Supertagging-based Parsing with Linear Context-free Rewriting Systems

Thomas Ruprecht and **Richard Mörbitz**
Faculty of Computer Science
Technische Universität Dresden
01062 Dresden, Germany
{thomas.ruprecht,richard.moerbitz}@tu-dresden.de

Abstract

We present the first supertagging-based parser for linear context-free rewriting systems (LCFRS). It utilizes neural classifiers and outperforms previous LCFRS-based parsers in both accuracy and parsing speed by a wide margin. Our results keep up with the best (general) discontinuous parsers, particularly the scores for discontinuous constituents establish a new state of the art. The heart of our approach is an efficient lexicalization procedure which induces a lexical LCFRS from any discontinuous treebank. We describe a modification to usual chart-based LCFRS parsing that accounts for supertagging and introduce a procedure that transforms lexical LCFRS derivations into equivalent parse trees of the original treebank. Our approach is evaluated on the English Discontinuous Penn Treebank and the German treebanks Negra and Tiger.

1 Introduction

In NLP, constituency parsing is a task that assigns – usually tree-shaped – syntactic structures to sentences. Formalisms such as context-free grammars (CFG) are used in this setting because they are conceptually simple, interpretable, and parsing is tractable (cubic in sentence length).

Discontinuous constituents span non-contiguous sets of positions in a sentence. The resulting phrase structures do not take the shape of a tree anymore, as they contain crossing branches (cf. the left of Fig. 1), and cannot be modeled by CFG. As a countermeasure, many treebanks, e.g. the Penn Treebank (PTB; Marcus et al., 1994), denote these phrase structures as trees nevertheless and introduce designated notations for discontinuity, which is then often ignored in parsing. However, discontinuity occurs in about 20 % of the sentences in the PTB and to an even larger extent in German treebanks such as Negra and Tiger. For parsing discontinuous constituents, so-called *mildly context-sensitive* grammar formalisms have been investigated, e.g. tree-adjoining grammars (TAG; Joshi et al., 1975) and linear context-free rewriting systems (LCFRS; Vijay-Shanker et al., 1987). An LCFRS derivation of a discontinuous phrase is shown in the right of Fig. 1. The increased expressiveness of these formalisms comes at the cost of a higher parsing complexity: given a sentence of length n, parsing is in $O(n^6)$ for TAG and $O(n^{3\cdot \mathrm{fo}(G)})$ for a binary LCFRS G. The grammar-specific *fanout* $\mathrm{fo}(G)$ indicates that G can parse constituents spanning up to n non-contiguous sets of positions. TAG have the same expressiveness as LCFRS with fanout 2 (Seki et al., 1991), which accounts for 96.67 % of the sentences in Negra and 96.83 % of the sentences in Tiger (Maier and Søgaard, 2008). Previous publications have established mildly context-sensitive formalisms in the field of statistical constituent parsing, and found methods to tame the high parsing complexity (Evang and Kallmeyer, 2011; Kallmeyer and Maier, 2013; Angelov and Ljunglöf, 2014; van Cranenburgh, 2012).

One approach for making parsing with mildly context-sensitive grammars tractable is *supertagging*, which was originally introduced for lexical TAG (Bangalore and Joshi, 1999). A TAG is lexical if each rule contains exactly one lexical item, i.e. word in the parsed language. The supertagger is a (often discriminative) classifier that selects for each position of the input sentence a subset of the rules of the TAG; these are the so-called supertags. Parsing is then performed with the much smaller grammar of supertags. Research on supertagging has also been conducted in the context of combinatory categorial grammars (CCG; Clark, 2002), but not yet for LCFRS. The use of recurrent neural networks (RNN) as classifiers in supertagging has improved their accuracy by far (Vaswani et al., 2016; Kasai et al., 2017; Bladier et al., 2018; Kadari et al., 2018).

Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pages 2923–2935
June 6–11, 2021.

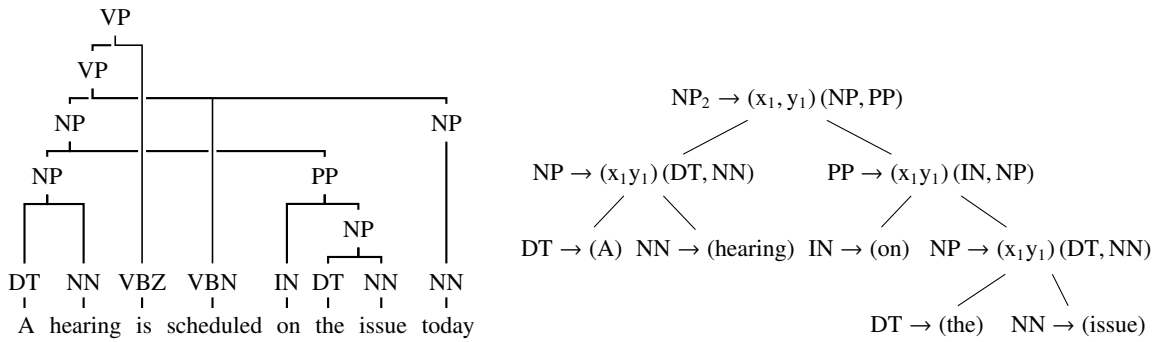


Figure 1: A discontinuous phrase structure tree of the sentence *A hearing is scheduled on the issue today* (left) and a corresponding LCFRS derivation of the discontinuous noun phrase *A hearing on the issue* (right).

Recently, Mörbitz and Ruprecht (2020) introduced a lexicalization procedure for probabilistic LCFRS[1], paving the way to employ supertagging for parsing with this formalism. Early experiments showed that the approach is infeasible in realistic settings: the set of rules explodes in a step of the construction where new rules are introduced for pairs of terminals in the grammar. To mitigate this problem, we conduct the procedure for single derivations. Consequently, we only have to construct rules for pairs of terminals that occur in sibling nodes of a derivation (cf. step (4) in Section 4). Moreover, we consider unweighted LCFRS, as weights of underlying grammar structures are usually not considered in supertagging-based approaches.

In this paper, we present the first supertagging-based parser for LCFRS. Section 3 extends the usual chart-based parsing approach for LCFRS to account for supertagging with lexical LCFRS. Section 4 adapts the lexicalization procedure by Mörbitz and Ruprecht (2020) to efficiently induce a lexical LCFRS from any given treebank. We implemented and evaluated the approach. Section 5 describes the experimental setups of our evaluation using three discontinuous treebanks (one English and two German). Section 6 compares our results to recent LCFRS-based parsers and other state-of-the-art parsers for discontinuous constituents. The implementation of our approach is published on GitHub.[2]

[1]Their work is an instance of the lexicalization of (unweighted) multiple context-free tree grammars by Engelfriet et al. (2018).

[2]https://github.com/truprecht/lcfrs-supertagger

2 Notation

We start by introducing some basic notation that will be used throughout Sections 3 and 4. The set of *non-negative* (resp. *positive*) *integers* is denoted by $\mathbb{N}$ (resp. $\mathbb{N}_+$). We abbreviate $\{1, ..., n\}$ by $[n]$; for each $n \notin \mathbb{N}_+$, the set $[n]$ is the empty set. An *alphabet* Σ is a finite and non-empty set; the *set of (finite) strings over* Σ is denoted by Σ^*. The symbol ε denotes an empty string or sequence.

Compositions. *Linear context-free rewriting systems (LCFRS)* extend the rule-based string rewriting mechanism of CFG to string tuples; we describe the generation process by *compositions*. Let $k \in \mathbb{N}$ and $s_1, \ldots, s_k, s \in \mathbb{N}_+$; one can think of k as the number of arguments of a function mapping string tuples of the sizes $s_1, \ldots, s_k$ to a string tuple of size s. A *Σ-composition* is a tuple $(u_1, \ldots, u_s)$ where each $u_1, \ldots, u_s$ is a non-empty string over Σ and variables of the form x_i^j with $i \in [k]$ and $j \in [s_i]$. Each of these variables must occur exactly once in $u_1 \cdots u_s$ and they are ordered such that x_i^1 occurs before x_{i+1}^1 and x_i^j occurs before x_i^{j+1} for each $i \in [k-1]$ and $j \in [s_i - 1]$. The set of all such compositions is denoted by $\mathrm{C}^{\Sigma}_{(s_1 \cdots s_k, s)}$.

As usual in the literature, we will only consider *binary* compositions (where $k \leq 2$) in the following. Variables of the form x_1^i and x_2^j are abbreviated by x_i and y_j, respectively.

We associate with each composition $(u_1, \ldots, u_s) \in \mathrm{C}^{\Sigma}_{(s_1 \cdots s_k, s)}$ a function from k string tuples, where the i-th tuple is of arity s_i, to a string tuple of arity s. This function is denoted by $[\![(u_1, \ldots, u_s)]\!]$. Intuitively, it replaces each variable of the form x_i in $u_1, \ldots, u_s$ by the i-th component of the first argument, and y_j by the j-th component of the second argument. The *identity composition* $(\mathrm{x}_1, \ldots, \mathrm{x}_s)$ is denoted by id_s.

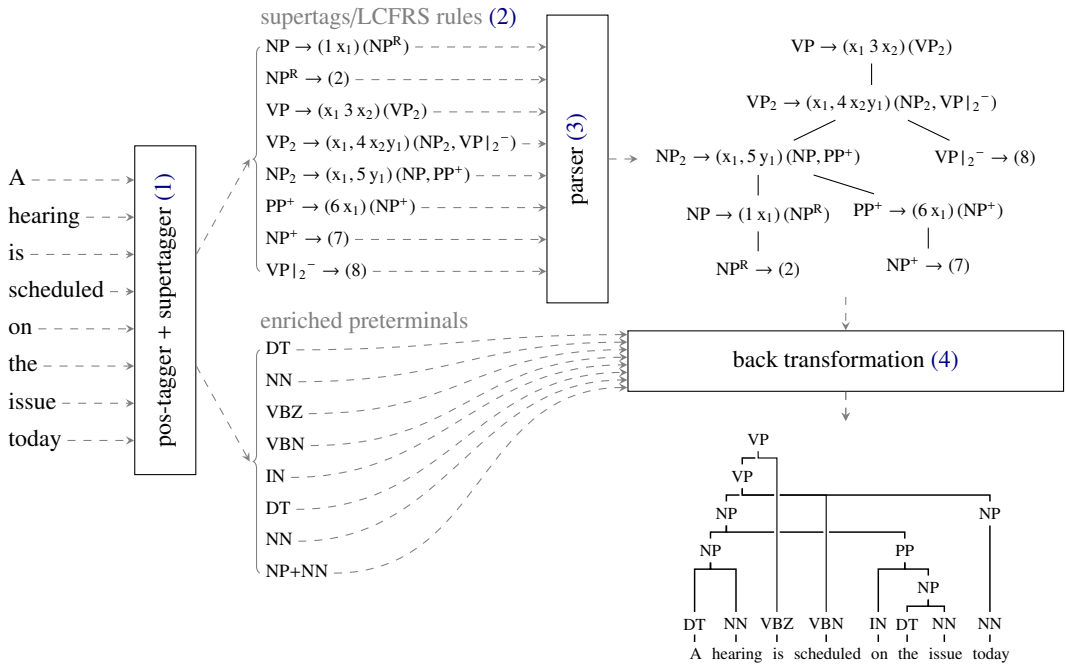


Figure 2: An overview over the supertagging-based parsing procedure. A sequence tagger predicts k (here, $k = 1$) supertags and one enriched preterminal (cf. step (4) in Section 4) for each sentence position. The supertags are rules of a uni-lexical LCFRS (the annotation of the nonterminals is explained in Section 4). The terminal of each rule is the sentence position it was predicted for (rather than the word at that position). The range of sentence positions is parsed and finally the resulting derivation is transformed into a parse tree. The transformation requires the predicted nonterminals, which are used as preterminals.

Let $c \in \mathrm{C}^{\Sigma}_{s_1 \cdots s_k, s}$ be a composition where $k \in [2]$, $i \in [k]$ such that $s_i = 1$, and $w \in \Sigma^*$. We obtain the *partial application of c to w as i-th argument*, denoted by $[\![c]\!]_i(w)$ as follows:

- $[\![c]\!]_2(w) \in \mathrm{C}^{\Sigma}_{(s_1, s)}$ is obtained from c by replacing y_1 by w and
- $[\![c]\!]_1(w)$ is obtained from c by replacing x_1 by w and each variable y_j by x_j. If $k = 1$, then $[\![c]\!]_1(w) \in \mathrm{C}^{\Sigma}_{\varepsilon, s}$, otherwise $[\![c]\!]_1(w) \in \mathrm{C}^{\Sigma}_{s_2, s}$.

LCFRS. A (binary) *LCFRS* is a tuple $G = (N, \Sigma, S, R)$ where

- N is a finite set (*nonterminals*) where each nonterminal $A \in N$ has a *fanout* $\mathrm{fo}(A) \in \mathbb{N}_+$,
- Σ is an alphabet (*terminals*),
- $S \subseteq N$ (*initial nonterminals*) such that $\mathrm{fo}(A) = 1$ for each $A \in S$, and
- R is a finite set (*rules*); each *rule* in R is of the form $A \to c(B_1, \ldots, B_k)$, where $k \in \{0, 1, 2\}$, $A, B_1, \ldots, B_k \in N$, and $c \in \mathrm{C}^{\Sigma}_{(\mathrm{fo}(B_1) \cdots \mathrm{fo}(B_k), \mathrm{fo}(A))}$. The function $[\![c]\!]$ maps k string tuples (of sizes $\mathrm{fo}(B_1), \ldots, \mathrm{fo}(B_k)$) to a string tuple of size $\mathrm{fo}(A)$. We call A the *left-hand side* (*lhs*), $B_1, \ldots, B_k$ the *right-hand side* (*rhs*) and c the rule's *composition*. We drop the parentheses around the rhs if $k = 0$.

In our examples, whenever the fanout of a nonterminal is greater than 1, the fanout is the subscript of the nonterminal. For instance, VP_2 denotes a verbal phrase with fanout 2. The *fanout of G* is $\mathrm{fo}(G) = \max_{A \in N} \mathrm{fo}(A)$.

Rules of the form $A \to c$, $A \to c(B)$, and $A \to c(B_1, B_2)$ are called *terminating*, *monic*, and *branching*, respectively. A rule is called *(uni-/double-)lexical*, if its composition contains *at least one* terminal (resp. *exactly one* terminal/*exactly two* terminals). The LCFRS G is called *(uni-)lexical*, if each rule is (uni-)lexical.

A *derivation in G (starting with $A \in N$)* is a tree over rules $d = r(d_1, \ldots, d_k)$ such that r is of the form $A \to c(B_1, \ldots, B_k)$ and each d_i is a derivation in G starting with B_i. The *set of derivations in G* is denoted by D^G. The *string tuple computed by d* is defined recursively as $w = [\![c]\!](w_1, \ldots, w_k)$ where $w_1, \ldots, w_k$ are the string tuples computed by $d_1, \ldots, d_k$; in the following we also call d *a derivation for w*.

3 Supertagging-based parsing

Our parsing model consists of two components: a uni-lexical LCFRS and a discriminative sequence

tagger which we henceforth call *supertagger*. The LCFRS is induced from a treebank by an adaptation of the construction of Mörbitz and Ruprecht (2020); the interested reader may find a detailed description of this procedure in Section 4. After the induction, we replace every terminal of the LCFRS by the wildcard symbol "_", and we refer to the resulting rules as *supertags*.

Our parsing pipeline is depicted in Fig. 2.

(1) Given a sentence w, the supertagger predicts for each position of w the k best supertags, where k constitutes a hyperparameter of our approach.

(2) We combine the predicted supertags to a new LCFRS which we call G_w. In doing so, we replace the wildcard of each supertag by the sentence position it was predicted for.

(3) We employ a usual chart-based parsing algorithm to parse the sequence $1\ 2\ \cdots\ |w|$ of sentence positions with G_w.

(4) We transform the resulting derivation in G_w into a parse tree of the same form as those in the original treebank.

As G_w only resembles a fraction of all supertags, this approach shifts a huge amount of work from parsing with grammars to predicting the rules. Thus its success is mainly determined by the quality of the supertagger.

4 Inducing Lexical LCFRS

Our lexicalization scheme is based on Mörbitz and Ruprecht (2020). However, we ignore all weights and perform lexicalization on individual derivations rather than on a grammar induced from the entire treebank. More specifically, we directly read off a set of uni-lexical rules from each tree in the treebank; then the union of these rules forms our uni-lexical LCFRS G_{lex}. In contrast to that, Mörbitz and Ruprecht (2020) first induce an LCFRS G from the entire treebank and then lexicalize G. Thus G_{lex} may have a different language than the lexicalization of G.

We obtain a set of uni-lexical rules from each tree t in the treebank by the following procedure.

(1) Binarize the tree. The symbol | is appended to constituents that result from binarization (this reflects Markovization with a vertical context of 1 and a horizontal context of 0).

(2) Transform the tree into an LCFRS derivation using the standard technique for induction of LCFRS (Maier and Søgaard, 2008).

(3) Collapse every chain of monic rules; the nonterminals of each chain are combined to a new nonterminal.

(4) Remove every terminating rule that has a parent and insert the terminal from its composition into the parent.

(5) Propagate terminals from double-lexical terminating rules into non-lexical branching rules. All rules in the resulting derivation are lexical.

(6) Split all remaining double-lexical terminating rules into two uni-lexical rules. All rules in the resulting derivation are uni-lexical. The resulting derivation is called $d_{\text{lex}}(t)$.

(7) Read off the rules of $d_{\text{lex}}(t)$; call them $R(t)$.

These steps are defined such that in the LCFRS formed by $R(t)$, $d_{\text{lex}}(t)$ is a derivation for the sentence of t. Moreover, we are able to reconstruct t from $d_{\text{lex}}(t)$ by reverting steps (6) to (1) (we will give the details later).

Finally, we obtain the uni-lexical LCFRS G_{lex} by combining the rules $R(t)$ for each tree t. The initial nonterminals of G_{lex} are all left-hand sides of roots of $d_{\text{lex}}(t)$.

Let t be a tree in the treebank. Steps (1) and (2) and their reversal are standard techniques for trees and LCFRS. After applying them to t, we obtain an LCFRS derivation in which each occurring rule is either of the form

- $A \to (\sigma)$, where σ is a terminal and A is the part-of-speech tag of σ,
- $A \to c(B_1)$ where $\text{fo}(A) = \text{fo}(B)$ and $c = \text{id}_{\text{fo}(A)}$, or
- $A \to c(B_1, B_2)$ where c contains no terminals and none of B_1, B_2 is an initial nonterminal.

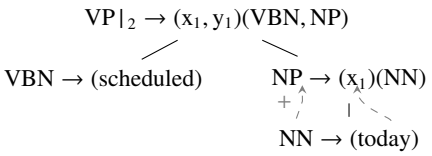
Let us denote this derivation by d.

In the following, we describe steps (3) to (6) of the above procedure in more detail (showing examples in Figs. 3 to 6) and also glimpse at how the individual steps are reverted.

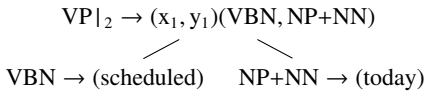
Step (3). We *repeatedly* replace parts in d of the form $A \to c(B)(B \to c')$ by $A{+}B \to c'$, and $A \to c(B)\big(B \to c'(C_1, C_2)(\dots)\big)$ by $A{+}B \to c'(C_1, C_2)(\dots)$, until there is no monic rule in d left. If the occurrence of $A \to c(B)$ is not the root of d, then the corresponding nonterminal in the parent's rhs is replaced by $A{+}B$.[3] After this step, there are only branching rules and terminating rules in d. Figure 3 shows an example for this step.

This step is easy to reverse, as the composition of every removed rule is $c = \text{id}_{\text{fo}(B)}$. We give the

[3]Note that root nodes in the derivation may be collapsed, this is why we use LCFRS with *multiple* initial nonterminals.



(a) A derivation for *scheduled today*. Gray arrows show how the bottom-most composition is chained with the monic rule on top.



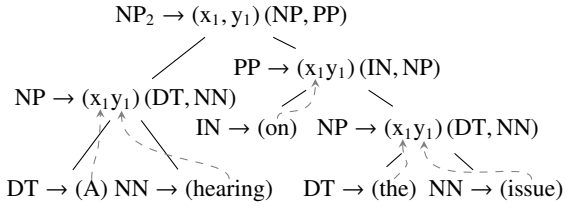
(b) The derivation resulting from applying step (3) to the derivation in Fig. 3a.

Figure 3: Example for step (3).

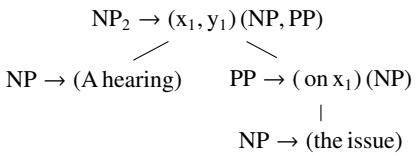
formal description in Appendix A.3.

Step (4). We remove every non-root occurrence r of a terminating rule $A \to (\sigma)$ in d. Let r be the ith child of its parent (with $i \in [2]$), then we replace the parent's composition c by $[\![c]\!]_i(\sigma)$ and remove the ith nonterminal in the parent's rhs.

We note that the parent becomes lexical, and after this step, every rule in d is either branching or lexical. Moreover, every terminal rule in d is either double-lexical (if both children were removed) or the root of d (and thus its only node). Figure 4 shows an example for this step.



(a) A derivation for the string tuple (*A hearing*, *on the issue*). Gray arrows show the terminals that are put into binary non-lexical rules during step (4).



(b) The derivation resulting from applying step (4) to the derivation in Fig. 4a.

Figure 4: Example for step (4).

Clearly, this step loses information, namely the left-hand sides of the removed rules. These nonterminals are part-of-speech tags (that may be enriched with nonterminals of collapsed monic rules from the previous step). For the reversal of this step, we opted to predict them along with the supertags as part of the supertagger. The formal description of the reversal is given in Appendix A.2.

Step (5). For each occurrence r of a branching rule $A \to c(A_1, A_2)$ in d, let us consider the occurrence t of the leftmost terminating rule (i.e. t is a leaf) that is reachable via the second successor of r. For example, in Fig. 5a, the two binary rules (r) are end points of gray arrows; these arrows start at the mentioned leaves (t). Our goal is to remove one terminal from t and propagate it all the way up to r. For this, at each node s on the path from t to r (from bottom up):

- If s is t, we remove the leftmost terminal in the rule's composition at s.
- If s is neither t nor r, we insert the last removed terminal right before the variable x_1 and then remove the leftmost terminal in the rule's composition at s.
  We note that if the rule at s is monic and the variable x_1 occurs right of the terminal in its composition, then we propagate a different terminal than the one received from the child. In order to be able to reverse this step, we need to remember whether the terminal in the rule's composition stayed the same or was swapped with the terminal received from the child. In the following, we consider this information as part of the rule (cf. the gray annotation $^{\text{swapped}}$ in Fig. 5).
- If s is r, we insert the last removed terminal right before the variable y_1 in the rule's composition at s.

If $s \neq r$, let s' be the parent of s and s the ith child of s'. If, after removal of a terminal at s, the first component in the composition is empty:

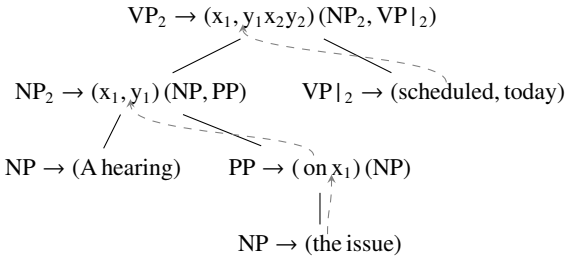
- we annotate the lhs nonterminal at s and the ith rhs nonterminal at s' with $^-$ and remove the empty component, and
- if $i = 1$ (resp. $i = 2$), we remove x_1 (resp. y_1) and replace every other occurrence of x_i by x_{i-1} (resp. y_j by y_{j-1}) at s'.

Otherwise, we annotate the nonterminals with $^+$.

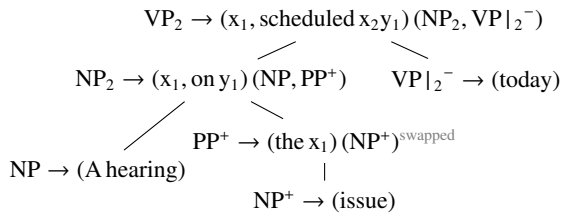
We note that the rule at r is uni-lexical and branching now, the rule at t is uni-lexical and terminating, and the number of terminals in each rule between them did not change. After this step, every rule in d is lexical. Figure 5 shows an example for this step.

There is a suitable leaf t for every branching rule

r. Intuitively, this holds since (a) after step (4) every leaf of d is a double-lexical rule and (b) for each branching rule we first go to its second successor and then follow the path of first successors until we reach a leaf. Here, (a) guarantees that there exists a double-lexical rule for each branching rule and (b) guarantees that each double-lexical rule is "assigned" to at most one branching rule, thus at most one terminal is removed from it. We refer the more interested reader to consult the proof of correctness by Engelfriet et al. (2018); this proof also applies to our method.



(a) A derivation for the string tuple (*A hearing*, *scheduled on the issue today*). Gray arrows show how terminals will be propagated through the derivation to lexicalize branching rules during step (5).



(b) The derivation resulting from applying step (5) to the derivation in Fig. 5a. A gray annotation $^{\text{swapped}}$ marks a monic rule whose terminal changed.

Figure 5: Example for step (5).

The reversal of this step removes all annotation ($^{+}$, $^{-}$, and $^{\text{swapped}}$) and restores each composition in d to its original form. We note that the original composition can be obtained deterministically; the construction is given in Appendix A.1.

Step (6). We replace the rightmost terminal σ_2 in the composition of each double-lexical terminating rule by a variable and add a new nonterminal A^{R} to the rule's right-hand side (making it a uni-lexical monic rule). Then we insert the rule $A^{\text{R}} \to (\sigma_2)$ as a child. After this step, every rule in d is uni-lexical. Figures 5b and 6 show an example for this step. The reversal of this step is straightforward.

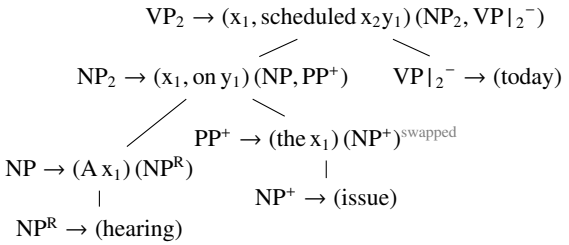


Figure 6: The derivation resulting from applying step (6) to the derivation in Fig. 5b. Each rule in the derivation contains exactly one terminal.

5 Experiments

Implementation. The induction of uni-lexical LCFRS and parsing was implemented by extending `disco-dop` (van Cranenburgh et al., 2016), from which we could borrow the generic LCFRS extraction and statistical parsing implementation. Moreover, we used the computation of evaluation scores in `disco-dop`.

The supertagger was implemented using the `flair` framework (Akbik et al., 2019). We report results for three types of architectures:

- *bert* – the output of the four topmost layers of a pretrained BERT[4] model (Devlin et al., 2019), which is fine-tuned during training,
- *flair* – the concatenation of language-specific fasttext (Mikolov et al., 2018) and flair embeddings (Akbik et al., 2018), which is fed through a two layered Bi-LSTM (Hochreiter and Schmidhuber, 1997), and
- *supervised (small/large)* – word embeddings (one-hot embeddings and character-based Bi-LSTM outputs) are trained with the model, and fed through a two layered Bi-LSTM. The *small* model adopts its size parameters from Stanojević and Steedman (2020); Coavoux and Cohen (2019) and the *large* model from Corro (2020).

On top of each of those, there are two linear layers in parallel: one for the supertags and one for the nonterminals that were removed in step (4) of our lexicalization scheme (i.e. part-of-speech tags plus nonterminals from collapsed monic rules). The sequence tagger is trained to predict the gold supertag and the removed nonterminal for each sentence position via the sum of cross-entropy losses. More

[4]We used language-specific flavors of `bert-base` that were available in huggingface's `transformers` library; `bert-base-german-cased` for Tiger and Negra, and `bert-base-cased` for DPTB.

details with respect to hyperparameters for all models are shown in Appendix B.

During parsing, the predicted supertags are interpreted as a probabilistic grammar. At each sentence position, the weight of the rules is the softmax of the supertag's score among the k best scores. The parsing implementation that we borrow from `disco-dop` supports heuristics and early-stopping to speed-up the parsing process. For each intermediate parse that does not span all sentence positions, we use the best supertag probability for each position that does not belong to the parse as a heuristic to estimate the weight of a complete parse.

We extended the parser with a fallback mechanism that deals with parse fails, i.e. when it is not able to find parse trees for the whole sentence. It picks the largest partial derivations (for parts of the sentence) that it was able to find and combines them as children of artificial *NOPARSE* nodes. This is especially beneficial in settings with small k as there are many parse fails (cf. Table 1 column cov.). For example, if we did not use this mechanism, we would obtain prec. = 95.53, rec. = 46.21 and F1 = 62.29 for the development set of Negra and $k = 1$ (cf. first row in Table 1).

We use only the highest-scoring nonterminal predicted for the reversal of step (4).

Data. Following Coavoux and Cohen (2019), we use three treebanks for discontinuous constituent parsing in our evaluations: Negra (Skut et al., 1998), Tiger (Brants et al., 2004), and a discontinuous version of the Penn treebank (DPTB; Evang and Kallmeyer, 2011). The treebanks were split according to the usual standards into training, development and test sets.[5] During development, the lexicalization, tagging and parsing were mostly tested and optimized using Negra. We binarized each training set before extracting the LCFRS and supertags. Markovization with horizontal context h = 0 and vertical context v = 1 has yielded the best results; we thus extracted 3275 supertags from the training set of Negra, 4614 from Tiger and 4509 from DPTB. More context in Markovization lead to a blowup in the number of supertags which proved to be disadvantageous.

Baselines. We report labeled F1-scores, obtained from predicted and gold parse trees using `disco-dop` (with the usual parameters in `proper.prm`), for all constituents (*F1*) and all discontinuous constituents (*Dis-F1*). Additionally to the scores, parse speed[6] is reported in sentences per second (*sent/s*).

Our scores are compared to recent state-of-the-art parsers for discontinuous constituent trees in four categories:

- *grammar-based parsers* (van Cranenburgh et al., 2016; Gebhardt, 2020; Versley, 2016) – that directly rely on an underlying (probabilistic) grammar,
- *chart-based parsers* (Corro, 2020; Stanojević and Steedman, 2020) – that share parsing algorithms with LCFRS, but lack an explicit set of rules,
- *transition systems* (Coavoux and Cohen, 2019; Coavoux et al., 2019), and
- *neural systems* (Fernández-González and Gómez-Rodríguez, 2020; Vilares and Gómez-Rodríguez, 2020) – all other recent parsing approaches using neural classifiers.

Our approach is in the first category, as the supertags are clearly constructed from a grammar that was extracted from the treebank. Therefore, the local relations in the predicted derivations are restricted to those occurring in the treebank. The approaches by Corro (2020) and Stanojević and Steedman (2020), on the other hand, rank spans in the sentence for occurrence in the predicted parse tree and predict their nonterminal; both independently from previous spans and nonterminals. Hence, they allow any combination of parent/child nonterminals in the resulting derivations.

6 Results

Table 1 shows statistics of our parser on the development sets for different amounts (k) of supertags per sentence position. Specifically, we report the parsing speed (*sent/s*), the rate of sentence positions where the gold supertag was among the k predicted supertags (*tag accuracy*), the rate of sentences that was completely parsed (*coverage*) and parsing scores (labeled *precision*, *recall* and *F1*).

We see that the parsing speed gradually drops with rising k, but for $k > 10$ there are barely any gains in terms of parsing scores. As expected, with rising k, the recall increases drastically. The preci-

[5] We use the split for Negra by Dubey and Keller (2003), for Tiger by Seddah et al. (2013), and the standard split for DPTB (sections 2–21 for training, 22 for development, 23 for testing).

[6] We measured the parsing speed on a system with an Nvidia GeForce RTX 2080, two Intel Xeon Silver 4114 (20 cores/40 threads at 2.2 GHz) and 256 GB RAM.

Table 1: Results for different values for k, i.e. how many supertags for each sentence position are used for parsing, on development sets after training. Includes only the results for our *bert* model.

k	Negra					
	sent/s	tag acc.	cov.	prec.	rec.	F1
1	78	87.37	64.60	93.10	57.16	70.83
2	75	93.35	85.80	91.27	78.73	84.54
3	71	94.88	92.10	91.22	84.87	87.93
5	69	96.23	96.80	91.15	89.32	90.23
10	63	97.50	99.20	91.30	91.00	91.15
15	58	98.19	99.80	91.37	91.27	91.32
20	53	98.52	99.90	91.56	91.44	91.50
k	Tiger					
1	75	89.38	73.26	94.23	67.14	78.42
2	72	94.96	92.00	92.65	85.18	88.76
3	67	96.33	96.14	92.53	88.83	90.65
5	66	97.45	98.32	92.66	90.79	91.71
10	61	98.48	99.67	92.74	91.79	92.27
15	56	98.83	99.88	92.73	91.88	92.30
20	53	99.04	99.98	92.80	91.97	92.38
k	DPTB					
1	95	90.00	58.88	91.95	59.84	72.50
2	90	95.92	90.06	93.89	87.21	90.43
3	82	97.21	95.35	93.92	91.41	92.65
5	79	98.24	97.88	94.07	93.05	93.56
10	72	98.98	99.29	94.03	93.81	93.92
15	66	99.23	99.82	94.09	94.01	94.05
20	61	99.38	99.94	94.11	94.05	94.08

Table 2: Our results compared to other published supertaggers. (†) Bladier et al. (2018) used a slightly different split of Tiger. (‡) Tian et al. (2020) use CCGbank instead of DPTB, which is a digest of PTB specifically for CCG parsing. Recent publications for CCG parsing use a slightly different split (sec. 0 instead of sec. 22 for development) for CCGbank than we do for DPTB.

Model	Tiger		DPTB ‡	
	tags	tag acc.	tags	tag acc.
Bladier et al., 2018	3426	88.51†	–	–
Kasai et al., 2017	–	–	4727	89.71
ours (sup., small)	4614	74.35	4509	83.06
ours (sup., large)	4614	78.96	4509	86.73
ours (flair)	4614	81.50	4509	88.55
ours (bert)	4614	85.40	4509	90.14
Tian et al., 2020	–	–	1284	96.39

sion, on the other hand, only changes slightly. The drop in precision using Negra and Tiger may be explained by a significant decrease in parse fails from $k = 1$ to $k = 2$, then the effects of fewer parse fails and considering lower-scored supertags seem to balance each other out. We found $k = 10$ to be a good parameter for the rest of our experiments.

Table 3 shows the parsing scores and speed of our trained models on the test set compared to the scores reported in other recent publications on discontinuous constituent parsing. The experiments suggest that parsing using LCFRS can greatly benefit from supertagging with respect to both speed and accuracy. This, however requires a strong discriminative classifier for the sequence tagger to predict useful rules. Most notably, the prediction accuracy for discontinuous constituents seems to strongly benefit from pretrained word embeddings.

Compared to other parsing approaches, we obtain results that are on par with the state of the art; recently, this is rather unusual for grammar-based constituent parsing. We would like to especially highlight our results for discontinuous constituents, which surpass the previous state of the art by a wide margin.

Unfortunately, we can only compare our results to those of other supertagging-based parsers to a very limited extent, as authors seem to either report no parsing scores at all (Bladier et al., 2018), or give attachment scores for dependency relations (Kasai et al., 2017; Tian et al., 2020). However, Table 2 compares the accuracy of our supertagger to some recent publications. The CCG community is very active in the field of neural supertagging, achieving an improvement from 91.3% (Lewis and Steedman, 2014) to 96.4% accuracy (Tian et al., 2020) for predicted supertags in the last 6 years. We can not compete with those numbers, but this may be due to the fact that there are far fewer supertags trained in these approaches than in ours. In the case of TAG, the supertagger by Bladier et al. (2018) achieves a better accuracy than ours. But again, there are fewer tags to predict. Compared to Kasai et al. (2017), our models with pretrained embeddings seem to be on par in both the number of tags and the accuracy.

7 Conclusion

We described an approach to utilize supertagging for parsing discontinuous constituents with LCFRS and evaluated it. Compared to other parsers for the

Table 3: Our results on test sets compared to other published constituent parsers. (†) Van Cranenburgh et al. (2016) use a different split for Tiger. (‡) Parsing speeds marked with this symbol were measured on our system as processing times were not reported by the authors.

Model	Negra			Tiger			DPTB		
	F1	Dis-F1	sent/s	F1	Dis-F1	sent/s	F1	Dis-F1	sent/s
Grammar-based systems									
van Cranenburgh et al., 2016	76.8	–	2‡	78.2†	–	1‡	87.0	–	<1‡
Gebhardt, 2020	81.7	43.5	–	77.7	40.7	–	–	–	–
ours (supervised, small)	77.59	28.28	136	78.38	44.73	103	87.24	64.84	103
ours (supervised, large)	82.72	49.03	136	82.53	55.91	101	90.08	72.87	95
ours (flair)	86.54	61.89	104	85.12	61.00	80	91.77	76.14	86
ours (bert)	90.94	**72.58**	68	88.34	**69.02**	60	93.32	**80.53**	57
Versley, 2016	–	–	–	79.50	–	–	–	–	–
Chart-based systems									
Corro, 2020 (supervised)	86.3	56.1	478	85.2	51.2	474	92.9	64.9	355
Corro, 2020 (bert)	**91.6**	66.1	–	**90.0**	62.1	–	**94.8**	68.9	–
Stanojević and Steedman, 2020	83.3	50.7	15‡	83.4	53.5	9‡	90.5	67.7	–
Transition systems									
Coavoux and Cohen, 2019	84.0	54.0	–	87.6	52.5	64	91.4	70.9	38
Coavoux et al., 2019	83.2	54.6	–	82.7	55.9	126	91.0	71.3	80
Neural systems									
Fernández-G. and Gómez-R., 2020	86.1	59.9	12‡	86.3	60.7	11‡	–	–	–
Vilares and Gómez-R., 2020 (bilstm)	77.1	36.5	715	79.2	40.1	568	89.1	41.8	611
Vilares and Gómez-R., 2020 (bert)	84.2	46.9	81	84.7	51.6	80	91.7	49.1	80

same grammar formalism, we achieve state-of-the-art results, i.e. we are more accurate and also faster (cf. Table 3, Grammar-based systems). In contrast to previous parsers utilizing LCFRS, we can even keep up with other (neural) parsing approaches and establish a new state of the art for discontinuous constituents (cf. Table 3, columns for Dis-F1).

Recent publications by Corro (2020) and Stanojević and Steedman (2020) address discontinuous constituent parsing using approaches that share an algorithmic foundation with LCFRS parsing, but do not use an underlying grammar. Both of them restrict constituents to two non-contiguous spans (equivalent to an LCFRS with fanout 2), we have no such limitation. Considering the margin between our discontinuous F1-score and theirs, we suppose that this restriction is only benefiting the complexity, not the accuracy.

Future Work. Compared to previous approaches for supertagging, we utilize large sets of supertags. We are confident that the accuracy of the supertagger can be improved by appropriately reducing these sets. The approach how terminals are transported in derivations during step (5) of the extraction is quite technical and chosen such that there is no impact on the fanout of the grammar (Mörbitz and Ruprecht, 2020). Alternative techniques could conceivably result in smaller sets of supertags and/or improve parsing results.

To validate the benefit of LCFRS (compared to using TAG or CCG) for supertagging-based approaches to constituent paring, we aim for an in-depth comparison of our work to previous approaches. However, currently, these approaches lack of publicly available implementations for constituent parsing.

Acknowledgements

We thank Alex Ivliev for conducting early experiments during the development of our parser, and our colleague Kilian Gebhardt as well as the anonymous reviewers for their insightful comments on drafts of this paper.

References

Alan Akbik, Tanja Bergmann, Duncan Blythe, Kashif Rasul, Stefan Schweter, and Roland Vollgraf. 2019. FLAIR: An easy-to-use framework for state-of-the-art NLP. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (Demonstrations)*, pages 54–59, Minneapolis, Minnesota. Association for Computational Linguistics.

Alan Akbik, Duncan Blythe, and Roland Vollgraf. 2018. Contextual string embeddings for sequence

labeling. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 1638–1649, Santa Fe, New Mexico, USA. Association for Computational Linguistics.

Krasimir Angelov and Peter Ljunglöf. 2014. Fast statistical parsing with parallel multiple context-free grammars. In *Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics*, pages 368–376, Gothenburg, Sweden. Association for Computational Linguistics.

Srinivas Bangalore and Aravind K. Joshi. 1999. Supertagging: An approach to almost parsing. *Computational linguistics*, 25(2):237–265.

Tatiana Bladier, Andreas van Cranenburgh, Younes Samih, and Laura Kallmeyer. 2018. German and French neural supertagging experiments for LTAG parsing. In *Proceedings of ACL 2018, Student Research Workshop*, pages 59–66, Melbourne, Australia. Association for Computational Linguistics.

Sabine Brants, Stefanie Dipper, Peter Eisenberg, Silvia Hansen-Schirra, Esther König, Wolfgang Lezius, Christian Rohrer, George Smith, and Hans Uszkoreit. 2004. TIGER: Linguistic interpretation of a German corpus. *Research on language and computation*, 2(4):597–620.

Stephen Clark. 2002. Supertagging for combinatory categorial grammar. In *Proceedings of the Sixth International Workshop on Tree Adjoining Grammar and Related Frameworks (TAG+ 6)*, pages 19–24.

Maximin Coavoux and Shay B. Cohen. 2019. Discontinuous constituency parsing with a stack-free transition system and a dynamic oracle. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 204–217, Minneapolis, Minnesota. Association for Computational Linguistics.

Maximin Coavoux, Benoît Crabbé, and Shay B. Cohen. 2019. Unlexicalized transition-based discontinuous constituency parsing. *Transactions of the Association for Computational Linguistics*, 7:73–89.

Caio Corro. 2020. Span-based discontinuous constituency parsing: a family of exact chart-based algorithms with time complexities from O(nˆ6) down to O(nˆ3). In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2753–2764, Online. Association for Computational Linguistics.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Amit Dubey and Frank Keller. 2003. Probabilistic parsing for German using sister-head dependencies. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*, pages 96–103, Sapporo, Japan. Association for Computational Linguistics.

Joost Engelfriet, Andreas Maletti, and Sebastian Maneth. 2018. Multiple context-free tree grammars: Lexicalization and characterization. *Theoretical Computer Science*, 728:29–99.

Kilian Evang and Laura Kallmeyer. 2011. PLCFRS parsing of English discontinuous constituents. In *Proceedings of the 12th International Conference on Parsing Technologies*, pages 104–116, Dublin, Ireland. Association for Computational Linguistics.

Daniel Fernández-González and Carlos Gómez-Rodríguez. 2020. Discontinuous constituent parsing with pointer networks. In *Proceedings of the 34th AAAI Conference on Artificial Intelligence*. Association for the Advancement of Artificial Intelligence.

Kilian Gebhardt. 2020. Advances in using grammars with latent annotations for discontinuous parsing. In *Proceedings of the 16th International Conference on Parsing Technologies and the IWPT 2020 Shared Task on Parsing into Enhanced Universal Dependencies*, pages 91–97, Online. Association for Computational Linguistics.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.

Aravind K. Joshi, Leon S. Levy, and Masako Takahashi. 1975. Tree adjunct grammars. *Journal of Computer and System Sciences*, 10(1):136–163.

Rekia Kadari, Yu Zhang, Weinan Zhang, and Ting Liu. 2018. CCG supertagging via bidirectional LSTM-CRF neural architecture. *Neurocomputing*, 283:31–37.

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

Jungo Kasai, Bob Frank, Tom McCoy, Owen Rambow, and Alexis Nasr. 2017. TAG parsing with neural networks and vector representations of supertags. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1712–1722, Copenhagen, Denmark. Association for Computational Linguistics.

Mike Lewis and Mark Steedman. 2014. Improved CCG parsing with semi-supervised supertagging. *Transactions of the Association for Computational Linguistics*, 2:327–338.

Wolfgang Maier and Anders Søgaard. 2008. Treebanks and mild context-sensitivity. In *Proceedings of the 13th conference on Formal Grammar*, pages 61–76, Hamburg, Germany. CSLI Publications.

Mitch Marcus, 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 the workshop on Human Language Technology*, pages 114–119, Plainsboro, New Jersey. Association for Computational Linguistics.

Tomas Mikolov, Edouard Grave, Piotr Bojanowski, Christian Puhrsch, and Armand Joulin. 2018. Advances in pre-training distributed word representations. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, Miyazaki, Japan. European Language Resources Association (ELRA).

Richard Mörbitz and Thomas Ruprecht. 2020. Lexicalization of probabilistic linear context-free rewriting systems. In *Proceedings of the 16th International Conference on Parsing Technologies and the IWPT 2020 Shared Task on Parsing into Enhanced Universal Dependencies*, pages 98–104, Online. Association for Computational Linguistics.

Djamé Seddah, Reut Tsarfaty, Sandra Kübler, Marie Candito, Jinho D. Choi, Richárd Farkas, Jennifer Foster, Iakes Goenaga, Koldo Gojenola Galletebeitia, Yoav Goldberg, Spence Green, Nizar Habash, Marco Kuhlmann, Wolfgang Maier, Joakim Nivre, Adam Przepiórkowski, Ryan Roth, Wolfgang Seeker, Yannick Versley, Veronika Vincze, Marcin Woliński, Alina Wróblewska, and Eric Villemonte de la Clergerie. 2013. Overview of the SPMRL 2013 shared task: A cross-framework evaluation of parsing morphologically rich languages. In *Proceedings of the Fourth Workshop on Statistical Parsing of Morphologically-Rich Languages*, pages 146–182, Seattle, Washington, USA. Association for Computational Linguistics.

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

Wojciech Skut, Thorsten Brants, Brigitte Krenn, and Hans Uszkoreit. 1998. A linguistically interpreted corpus of German newspaper text. In *Proceedings of the ESSLLI Workshop on Recent Advances in Corpus Annotation*, Saarbrücken, Germany.

Miloš Stanojević and Mark Steedman. 2020. Span-based LCFRS-2 parsing. In *Proceedings of the 16th International Conference on Parsing Technologies and the IWPT 2020 Shared Task on Parsing into Enhanced Universal Dependencies*, pages 111–121, Online. Association for Computational Linguistics.

Yuanhe Tian, Yan Song, and Fei Xia. 2020. Supertagging Combinatory Categorial Grammar with attentive graph convolutional networks. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6037–6044, Online. Association for Computational Linguistics.

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. Association for Computational Linguistics.

Andreas van Cranenburgh, Remko Scha, and Rens Bod. 2016. Data-oriented parsing with discontinuous constituents and function tags. *Journal of Language Modelling*, 4(1):57–111.

Ashish Vaswani, Yonatan Bisk, Kenji Sagae, and Ryan Musa. 2016. Supertagging with LSTMs. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 232–237, San Diego, California. Association for Computational Linguistics.

Yannick Versley. 2016. Discontinuity (re)^2-visited: A minimalist approach to pseudoprojective constituent parsing. In *Proceedings of the Workshop on Discontinuous Structures in Natural Language Processing*, pages 58–69, San Diego, California. Association for Computational Linguistics.

Krishnamurti Vijay-Shanker, David Jeremy Weir, and Aravind K. Joshi. 1987. Characterizing structural descriptions produced by various grammatical formalisms. In *Proceedings of the 25th Annual Meeting on Association for Computational Linguistics*, ACL '87, pages 104–111, Stroudsburg, PA, USA. Association for Computational Linguistics.

David Vilares and Carlos Gómez-Rodríguez. 2020. Discontinuous constituent parsing as sequence labeling. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2771–2785, Online. Association for Computational Linguistics.

A Unlexicalizing Derivations

In this Appendix we formally describe the reversal of selected steps of our lexicalization scheme (cf. Section 4). In each instance we assume a derivation like it would be obtained right after applying the corresponding step.

A.1 Reversal of step (5).

This step is applied to each occurrence r of branching rules of the form $A \to c(A_1, A_2)$ from the bottom to the top (i.e. it was already done for branching rules in the subtrees below a node before it is applied to the node itself). Let t be the leftmost occurrence of a terminating rule that is reachable from the second child of r. At each node s on the path from r to t (i.e. from top down) we perform three steps. (1) We transform the composition back into the original composition, (2) we remove all annotation ($^+$, $^-$, and $^{\text{swapped}}$), and (3) we pass a terminal to the child (if s is not t).

Obtaining the original composition. The composition at each node s is transformed back into the original composition depending on the type of the rule. We note that if s is a branching rule and $s \neq r$, the composition at s was already changed previously in this step and we leave it as it is.

Branching rule Let $B \to (u_1, \ldots, u_s)\,(B_1, B_2)$ be the rule at s and σ be the terminal in $(u_1, \ldots, u_s)$.

- If B_2 is annotated with $^-$ (i.e., its first component was removed during step (5)), we replace σ with y_0, and replace every occurrence of y_i by y_{i+1}.
- Otherwise, σ is removed from $(u_1, \ldots, u_s)$.

Moreover, if s occurs as a successor of the right child of some other branching rule, then the nonterminals B and B_1 have annotation as well.

- If B_1 and B are annotated with $^-$, then we replace $(u_1, \ldots, u_s)$ by $(\mathrm{x}_0, u_1, \ldots, u_s)$.
- If B_1 is annotated with $^-$ and B with $^+$, then we replace $(u_1, \ldots, u_s)$ by $(\mathrm{x}_0 u_1, \ldots, u_s)$.

If B_1 is annotated with $^-$, we replace every occurrence of x_i by x_{i+1} afterwards.

Monic rule Let $B \to c'(B_1)$ be the rule at s with c' of the form $(u_1, \ldots, u_s)$, σ_1 be the terminal received from the parent, and σ_2 be the terminal in c'.

1. If B is annotated with $^-$, then c' is replaced by $(\varepsilon, u_1, \ldots, u_s)$.
2. If B_1 is annotated with $^-$, then x_0 is inserted as the first symbol in the first component in c'. After that, every occurrence of x_i is replaced by x_{i+1}.
3. If the terminal was swapped during step (5), the terminal σ_2 is removed from c' and σ_1 is added as the first symbol in the first component of c.

We remark that if B is annotated with $^-$, then it must be the case that B_1 is annotated with $^-$ as well or the terminal was swapped during step (5). Hence we do not add empty components here.

Terminating rule Let $B \to (\sigma_2)$ be the rule at s and σ_1 be the terminal received from the parent. We replace the rule by $B \to (\sigma_1, \sigma_2)$ if B is annotated with $^-$ and by $B \to (\sigma_1\sigma_2)$ otherwise.

Passing the terminal to the child.

- If s is r, let σ be the terminal in c. We pass σ to the next node on the path to t.
- If s is neither r nor t, and there is a branching rule at s, we pass the terminal received from the parent to the next node on the path to t.
- If there is a monic rule of the form $B \to c'(B_1)$ at s, we let σ_1 be the terminal reveived from the parent and σ_2 the terminal in c'. If the terminal in this rule was swapped during step (5), we pass σ_2 to the next node on the path to t, otherwise we pass σ_1.

A.2 Reversal of step (4).

We recall that during step (4), certain nonterminals that occurred as the lhs of terminating rules were removed. For reverting this step, we assume that these nonterminals are predicted by an oracle. We replace every occurrence of a terminating rule of the form

- $A \to (\sigma_1\sigma_2)$ by
  $A \to (\mathrm{x}_1\mathrm{y}_1)(A_1, A_2)(A_1 \to (\sigma_1), A_2 \to (\sigma_2))$
  and
- $A \to (\sigma_1, \sigma_2)$ by
  $A \to (\mathrm{x}_1, \mathrm{y}_1)(A_1, A_2)(A_1 \to (\sigma_1), A_2 \to (\sigma_2))$,

where A_1 and A_2 are the predicted nonterminals for σ_1 and σ_2, respectively.

We replace every subderivation d' of the form $A \to c(B)(d'_1)$, where σ is the terminal in c and A_1 the predicted nonterminal for σ, as follows:

Table 4: Hyperparameters for the sequence tagger.

Parameter	supervised (small/large)	flair	bert
word embeddings	one-hot (32d/300d) + character-based (100d/100d)	fasttext + flair	top 4 bert layers
Bi-LSTM	2 layers, each 200/800 states	2 layers, each 800 states	no
linear layer	no. of supertags + no. of pos tags	same	same
dropout	0.5	0.5	0.1
loss	cross entropy (supertags + pos tags)	same	same
optimizer	Adam ($\beta_1 = 0.9, \beta_2 = 0.999$)	same	same
base learning rate (lr)	10^{-3}	10^{-3}	$5 \cdot 10^{-5}$
weight decay	0	0	0.1
lr scheduler	reduce on plateau (half lr if dev. F1 score decreases)	same as sup.	constant
batch size	32	32	32
training epochs	max. 100, or if lr $< 10^{-4}$	same as sup.	5
k-best supertags	10	10	10

- if σ is the first symbol in c, then c' is obtained from c by replacing, for each $i \in [\mathrm{fo}(B)]$, x_i with y_i and σ with x_1; d' is replaced by $A \to c'(A_1, B)(A_1 \to (\sigma), d'_1)$,
- otherwise, c' is obtained from c by replacing σ with y_1; d' is replaced by $A \to c'(B, A_1)(d'_1, A_1 \to (\sigma))$.

The composition c' is constructed such that $[\![c']\!]_1(\sigma) = c$ in the first case and $[\![c']\!]_2(\sigma) = c$ in the second case.

A.3 Reversal of step (3).

We *repeatedly* replace every occurrence r of the form $A{+}B \to c(\dots)(\dots)$ by

$$A \to \mathrm{id}_{\mathrm{fo}(B)}(B)\Big(B \to c(\dots)(\dots)\Big),$$

until there are no nonterminals of the form $A{+}B$ left in the derivation. If r has a parent, then we replace the corresponding nonterminal $A{+}B$ in the parent's rhs by A.

B Model parameters

Table 4 shows detailed parameters for our three reported models. The architecture of the *supervised (small)* models is the same as Stanojević and Steedman (2020); Coavoux and Cohen (2019), and *supervised (large)* the same as Corro (2020), to allow fair comparisons.

Note, that the time needed to train the models varies heavily: As the BERT embeddings are only fine-tuned for a small amount of iterations, training the *bert* model took less than an hour using the Negra corpus. The *bilstm* model benefits from the fixed word embeddings as they are only computed once; training it took ca. an hour. Both *supervised* models train a lot slower, training for each of those took ca. 6 hours.