A Derivational Model of Discontinuous Parsing

Mark-Jan Nederhof[1] and Anssi Yli-Jyrä[2]

[1] School of Computer Science, University of St Andrews, UK
[2] Department of Modern Languages, University of Helsinki, Finland

Abstract. The notion of latent-variable probabilistic context-free derivation of syntactic structures is enhanced to allow heads and unrestricted discontinuities. The chosen formalization covers both constituency parsing and dependency parsing. By the new framework, one obtains a probability distribution over the space of all discontinuous parses. This lends itself to intrinsic evaluation in terms of cross-entropy. The derivational model is accompanied by an equivalent automaton model, which can be used for deterministic parsing.

Keywords: parsing, grammars, weighted automata

1 Introduction

Much of traditional parsing theory [21] considers a syntactic structure to be a constituency tree in which siblings are linearly ordered, and a sentence is formed by the labels of the leaves from left to right. Whereas most English sentences can be given syntactic analyses that satisfy these constraints, other languages, especially those with more flexible word order such as German or Czech, do not let themselves be described easily, if at all, by using trees of this form. At the very least, these languages require types of syntactic trees with 'crossing edges', a phenomenon which is known formally as *discontinuity.* In dependency parsing this corresponds to *non-projectivity.*

In the theory of constituency parsing, leaf nodes in a parse tree are commonly words and punctuation tokens, and non-leaf nodes represent categories; one may also assign a special role to the nodes one level above the words, to represent parts of speech. In the theory of dependency parsing however, each node corresponds to a word or punctuation token, and can also be tagged with a part of speech; moreover, the parent-child edges are typically labeled by dependency relations.

Specifying a space of discontinuous constituency structures requires extensions to traditional parsing theory. Moreover, one generally needs to specify a probability distribution over this space, so one may disambiguate an input string that allows more than one parse. Rather than probabilities, one may more generally use weights to favour or disfavour one parse relative to another.

One approach to specifying a space of discontinuous constituency structures is to use grammatical formalisms that distinguish between derived trees and derivation trees, with discontinuity introduced through the interaction between the two kinds of trees. This idea has been explored for tree adjoining grammars

(TAGs) and linear context-free rewriting systems (LCFRSs). For TAG, see [17]. For LCFRS applied to dependency parsing see [20] and for LCFRS applied to constituency parsing see [8]. In all of these cases, probability models may remain attached to grammar rules, as in the case of traditional probabilistic context-free parsing [15].

Another approach takes shift-reduce automata as starting point, with an additional mechanism for swapping elements in the stack. This has been explored both in constituency parsing [22] and in dependency parsing [28]. Weights may be associated with possible next parser actions by applying a classifier on the current state of the parser. For a sequence of steps of the automaton, these weights may be added. Beam-search may then restrict the number of alternative computations to be considered at each input position, to reduce computational costs. With a beam size of 1, one obtains an entirely deterministic algorithm. Even with larger beam sizes, parsing tends to be fast.

This paper presents a form of discontinuous parsing that connects grammar-based approaches and automaton-based approaches. For the former, we introduce a class of grammars that can describe a probability distribution over a set of discontinuous parses. For the latter, we show how an automaton can be trained to deterministically construct discontinuous parses, in such a way that the computations of the automaton represent derivations of a corresponding grammar and vice versa.

Before we can define a form of discontinuous shift-reduce parsing that is directed by a grammar, we first need to develop a suitable grammatical formalism that allows discontinuity. For this, we will retain the main principle of traditional context-free derivation, that is, the process of repeatedly rewriting a nonterminal to a string of terminals and nonterminals, starting from the start symbol, until only terminals remain. By attaching latent variables to these nonterminals, obtained through EM or spectral methods, one may build upon recent developments for continuous parsing that have pushed the state-of-the-art in parsing accuracy.

We depart from existing frameworks in a redefinition of derivation steps. In our form of derivation, when a nonterminal is rewritten to a string of grammar symbols, only one of these grammar symbols, the head of the used rule, needs to remain in the same location in the sentential form as the rewritten symbol. The other symbols from the rule may 'jump over' existing symbols in the sentential form. This 'jumping over' is what achieves the desired discontinuity. Our work has elements in common with other approaches that redefine context-free derivations, by allowing descendants of a node in the tree to 'jump over' descendants of that node's siblings [35, 19, 7]. However, this was typically combined with Boolean constraints to avoid arbitrary word order.

Our approach uses probability distributions to favour common word order over pathological word order, with parameters that can be estimated automatically. Firstly, probabilities are attached to our rules, much as in traditional probabilistic (latent variable) grammars. Secondly, additional probability distributions are used to govern the introduction of discontinuity as part of the application of these rules. The parameters of these probability distributions can be obtained through relative frequency estimation on the basis of a corpus of trees. This is by virtue of constraints we place on discontinuous derivations, which leave exactly one derivation for each discontinuous tree.

Many other approaches to discontinuous parsing involve a grammar constructed out of a training corpus. There are often no restrictions per se on the kinds of discontinuities that are covered by the trained grammar, provided similar discontinuities were present in the training corpus. This holds for the approaches involving TAG and LCFRS discussed earlier. This also holds for hybrid grammars [26], pseudo-projectivity [16, 31], and the reversible splitting conversion of [2]. Our work differs from this in that our model provably allows for any discontinuity in unseen input, although very discontinuous trees will become very improbable if most of the training corpus is continuous.

A shorter version of the current article has appeared in LATA 2017. The current version further includes:

– extended explanations and examples
– elaboration of the theory of deterministic discontinuous parsing
– extended evaluation using cross-entropy
– evaluation of deterministic parsing

2 Trees

In the following, we define a type of trees that is able to represent both discontinuous constituency structures and nonprojective dependency trees, using the notion of *heads*. Heads are an inherent component of most definitions of dependency structures [12, 11]. Most older definitions of constituency structures avoid the notion of heads altogether, whereas some more recent literature tends to at least involve heads in some way [6].

We let $\mathbb{N}^+$ denote the set of natural numbers excluding 0, and we define $[n] = \{1, 2, \ldots, n\}$ for $n \in \mathbb{N}^+$.

Let Σ be a finite set of *terminals* and let N be a finite set of *labels*. Terminals correspond naively to tokens, although in reality they can represent open classes of distributionally similar tokens. Labels could represent categories, parts of speech, semantic roles, or even a combination of these.

The set $H(\Sigma, N)$ of *headed trees* over Σ and N is defined inductively as follows. We have a *leaf* $(a, i) \in H(\Sigma, N)$ for each $a \in \Sigma$ and $i \in \mathbb{N}^+$. We also have $A(s_1 \cdots s_k, h, t_1 \cdots t_\ell) \in H(\Sigma, N)$ for each $A \in N$, $k, l \geq 0$, and $s_1, \ldots, s_k, h, t_1, \ldots, t_\ell \in H(\Sigma, N)$. Nothing else is in $H(\Sigma, N)$.

Two headed trees are *structurally identical* if they differ at most in their input positions. Inductively, any two leaves (a, i) and (a, j) are structurally identical, and two headed trees $A(s_1, \ldots, s_k, h, t_1, \ldots, t_\ell)$ and $A(s'_1, \ldots, s'_k, h', t'_1, \ldots, t'_\ell)$ are structurally identical if $s_1, \ldots, s_k, h, t_1, \ldots, t_\ell$ are structurally identical to $s'_1, \ldots, s'_k, h', t'_1, \ldots, t'_\ell$, respectively.

For each tree t, we define its *head leaf* (a, i) where a is called the *head terminal* of t and i is called the *head position* of t. For a leaf $t = (a, i)$, its head leaf is

t itself. For a non-leaf tree $t = A(s_1 \cdots s_k,\ h,\ t_1 \cdots t_\ell)$, its head leaf is defined recursively as the head leaf of h.

The set of all positions in a headed tree t is denoted by $pos(t)$, that is, $pos((a,i)) = \{i\}$ and $pos(A(s_1 \cdots s_k, h, t_1 \cdots t_\ell)) = \bigcup_{i \in [k]} pos(s_i) \cup pos(h) \cup \bigcup_{j \in [\ell]} pos(t_j)$. A headed tree is a *positioned headed tree* (ph-tree) if:

1. no two leaves share the same position, and
2. for every subtree $t = A(s_1 \cdots s_k, h, t_1 \cdots t_\ell)$, the sequence of the head positions of $s_1, \ldots, s_k,\ h,\ t_1, \ldots, t_\ell$ is strictly increasing.

The *yield* of a ph-tree whose leaves are $(a_1, k_1),\ \ldots,\ (a_m, k_m)$, arranged such that $k_1 < \ldots < k_m$, is the string $a_1 \cdots a_m$. If two ph-trees are *structurally identical*, then for each node in the first tree there is a corresponding node in the second tree. Because head positions of siblings need to be strictly increasing, corresponding children from corresponding parents have the same relative order. The two trees may differ however in how the descendents of these children are interleaved with other nodes in the tree. If two ph-trees are structurally identical, then the yield of the first is a permutation of the second. The converse does not hold however: for a given ph-tree and a permutation of its yield, it is not guaranteed that there is a structurally identical ph-tree whose yield is that permutation.

A ph-tree t is *complete* if $pos(t) = [n]$ for some $n \in \mathbb{N}^+$. A ph-tree t is *continuous* if:

1. $pos(t) = \{m, m+1, \ldots, n-1, n\}$ for some $m, n \in \mathbb{N}^+$, and
2. each immediate subtree is continuous.

The set of ph-trees is denoted by $P(\Sigma, N)$, the set of complete ph-trees by $C(\Sigma, N)$, and the set of continuous and complete ph-trees by $C_c(\Sigma, N)$.

Figure 1(a) shows an example of a complete ph-tree t, with $pos(t) = [8]$. Note in particular that every position between 1 and 8 occurs in precisely one leaf. An example of a subtree t' of t is, in linearized form:

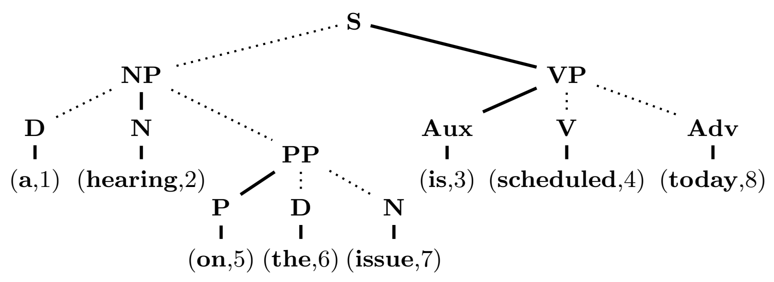
$$\mathbf{NP}(\ \mathbf{D}(\varepsilon,\ (\mathbf{a}, 1),\ \varepsilon),\quad \mathbf{N}(\varepsilon,\ (\mathbf{hearing}, 2),\ \varepsilon),\quad \mathbf{PP}(\varepsilon,\ \mathbf{P}(\varepsilon, (\mathbf{on}, 5), \varepsilon),\ \ldots)\ )$$

where ε denotes the empty string. The head positions of the immediate subtrees are 1, 2, 5, respectively, which is a strictly increasing sequence as required.

In the example in Figure 1(a), the subtree rooted in the node labeled **PP** is continuous, but the subtrees rooted in the nodes labeled **NP** and **VP** are not. The set of positions in this last subtree is $\{3, 4, 8\}$, which contains a 'gap' between 4 and 8. Because of this, the tree in Figure 1(a) as a whole is discontinuous.

The critical reader may argue that one could also formalize a ph-tree $A(s_1 \cdots s_k,\ h,\ t_1 \cdots t_\ell)$ using set notation as $A(h, \{s_1 \cdots s_k, t_1 \cdots t_\ell\})$, omitting the relative order of the immediate subtrees, but keeping the explicit identification of the head. This is without loss of information, as the order can be reconstructed on the basis of the head positions of the immediate subtrees. We prefer to keep the earlier notation however, as it simplifies making a connection with grammars in Section 3.

(a)



(b)

$$\begin{aligned}
&\widehat{\mathbf{S}} \Rightarrow_{rm} \mathbf{NP}\ \widehat{\mathbf{VP}} \Rightarrow_{rm}\\
&\mathbf{NP\ Aux\ V}\ \widehat{\mathbf{Adv}} \Rightarrow_{rm}\\
&\widehat{\mathbf{NP}}\ \mathbf{Aux\ V\ today} \Rightarrow_{rm}\\
&\mathbf{D\ N\ Aux\ V}\ \widehat{\mathbf{PP}}\ \mathbf{today} \Rightarrow_{rm}\\
&\mathbf{D\ N\ Aux\ V\ P\ D}\ \widehat{\mathbf{N}}\ \mathbf{today} \Rightarrow_{rm}\\
&\mathbf{D\ N\ Aux\ V\ P}\ \widehat{\mathbf{D}}\ \mathbf{issue\ today} \Rightarrow_{rm} \ldots
\end{aligned}$$

Fig. 1. Complete ph-tree with yield "*a hearing is scheduled on the issue today*" and corresponding rightmost derivation, assuming π is the identity function. Solid edges lead to heads and dotted edges to dependents. Examples of headed rules used here are $\mathbf{NP} \to \mathbf{D}\ \langle\mathbf{N}\rangle\ \mathbf{PP}$ and $\mathbf{PP} \to \varepsilon\ \langle\mathbf{P}\rangle\ \mathbf{D}\ \mathbf{N}$ with ε here indicating absence of left dependents.

We call a ph-tree *unilexical* if each subtree is either a leaf or of the form $A(s_1 \cdots s_k, h, t_1 \cdots t_\ell)$, where h is a leaf and none of $s_1, \ldots, s_k$, $t_1, \ldots, t_\ell$ are leaves. A complete ph-tree that is unilexical would more commonly be called a dependency structure.

Figure 2(a) presents an example of a unilexical ph-tree, in which the terminals are parts of speech; such ph-trees will be discussed again in Section 7. Figure 2(b) presents an equivalent but more common representation of a dependency structure, with occurrences of parts of speech arranged in textual order, and edges leading towards dependents. Note the additional edge from an artificial zeroth input token to the head word of the sentence, here with part of speech **Aux**. If we always assume such an edge for each dependency tree, then existence of crossing edges breaks the projectivity condition [23, 39]. In the dependency tree depicted in the figure, there are some crossing edges. Hence the dependency tree is non-projective.

3 Headed Context-free Grammars

In this section, we give a formalization of headed grammars that differs somewhat from related definitions in the literature, for example those of [1, 6]. This is motivated by the need for a streamlined presentation that allows formulation of

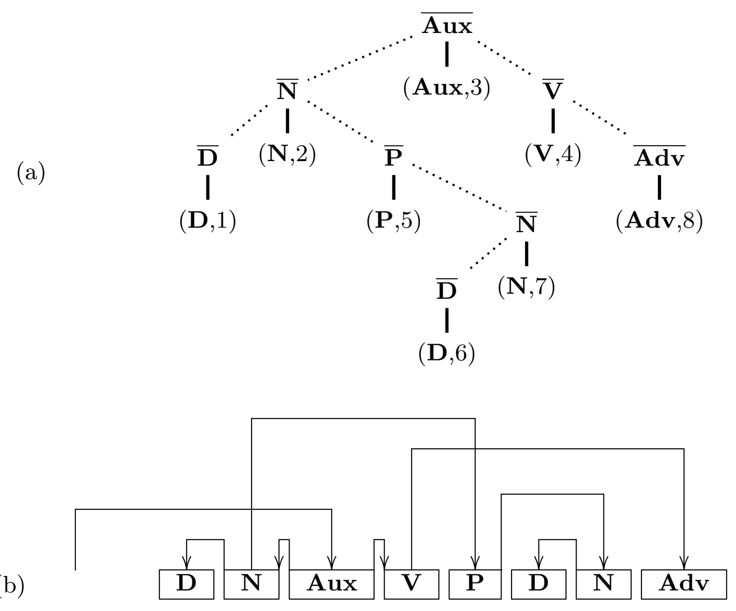


Fig. 2. Unilexical ph-tree and a more common representation of a dependency structure.

both discontinuous constituency parsing and non-projective dependency parsing. We also wish to incorporate the notion of latent variables, as this is an essential component of some state-of-the-art parsers [32, 24].

A *headed context-free grammar* (HCFG) is a 6-tuple $(\Sigma, Q, q_{init}, R, N, \pi)$, where Σ is a finite set of terminals as before, Q is a finite set of *states*, $q_{init} \in Q$ is the *initial state*, and R is a set of *headed rules*. Also as before, N is a finite set of labels. The function π maps states to labels. Several states may be mapped to the same label, to accommodate for latent variables. However, in the examples and in the current experiments (Section 8), π is always the identity function.

A headed rule has the form $q \to \alpha\langle Z\rangle\beta$, where $q \in Q$, $Z \in \Sigma \cup Q$ and $\alpha, \beta \in (\Sigma \cup Q)^*$. Here q is called the *left-hand side*, and $\alpha\langle Z\rangle\beta$ the *right-hand side*, in which Z is the *head*, and the symbols in α and β are the left and right dependents, respectively. The set R of headed rules is potentially infinite, in which case we assume finite descriptions for the sets of strings α and β that may appear in rules of the form $q \to \alpha\langle Z\rangle\beta$, for given q and Z. Such descriptions would typically be finite automata.

The derivations of a HCFG are defined by the binary relation $\Rightarrow$ that has, for every rule $q \to X_1 \cdots X_k\langle Z\rangle Y_1 \cdots Y_\ell$, and for every $\gamma_0, ..., \gamma_k, \delta_0, ..., \delta_\ell \in (\Sigma \cup Q)^*$:

$$\gamma_0\gamma_1 \cdots \gamma_k q \delta_0 \delta_1 \cdots \delta_\ell \quad \Rightarrow \quad \gamma_0 X_1 \gamma_1 X_2 \cdots X_k \gamma_k Z \delta_0 Y_1 \delta_1 Y_2 \cdots Y_\ell \delta_\ell$$

The reflexive, transitive closure of $\Rightarrow$ is denoted by $\Rightarrow^*$.

Note that after a rewrite step of $\Rightarrow$, the head Z must be in the same position as q relative to the symbols in $\gamma_0, \ldots, \gamma_k, \delta_1, \ldots \delta_\ell$. The other symbols in the right-hand side of the rule may be placed somewhere else, as long as the relative order of $X_1 \cdots X_k Z Y_1 \cdots Y_\ell$ is preserved. Further note that if we were to restrict $\gamma_1 \cdots \gamma_k$ and $\delta_0 \cdots \delta_{\ell-1}$ to be always ε, then we obtain the familiar notion of context-free derivation. We call such a derivation a *continuous derivation.*

Where we speak of '*a derivation* $q_{init} \Rightarrow^* w$', we implicitly assume a certain sequence of rule applications that leads us from q_{init} to w, via intermediate *sentential forms.* This includes not only the identity of each applied rule, but also the occurrence of the state on which it is applied (a sentential form may contain several occurrences of the same state), and the locations where the left and right dependents are placed among the existing elements in the sentential form.

A derivation $q_{init} \Rightarrow^* w$ maps to a complete ph-tree. To make this precise, we first enhance $w = a_1 \cdots a_n$ to $w' = (a_1, 1) \cdots (a_n, n)$, so that each terminal is coupled to its position in the derived string. In the same way, we construct a set of *enhanced rules*, on the basis of the rules that occur in the derivation. Roughly speaking, the enhanced rules are chosen in such a way that their positions are consistent with the positions of the leaf nodes needed to derive the sentence w'. Concretely, for a rule ρ of the form $q \to X_1 \cdots X_k \langle Z \rangle Y_1 \cdots Y_\ell$, a corresponding enhanced rule is of the form

$$(q, i) \to (X_1, i_1) \cdots (X_k, i_k) \langle (Z, i) \rangle (Y_1, j_1) \cdots (Y_\ell, j_\ell),$$

where $1 \leq i_1 < \ldots < i_k < i < j_1 < \ldots < j_\ell \leq n$. The set of such enhanced rules for given rule ρ will be denoted by $\rho^{(n)}$. We now *extend* the derivation $q_{init} \Rightarrow^* w$ in a unique way to an *enhanced derivation* $(q_{init}, i_0) \Rightarrow^* w'$, for some i_0, where we replace an application of a rule ρ by an application of an enhanced rule $\rho' \in \rho^{(n)}$ in which the positions are uniquely determined by the corresponding leaf positions in the derivation.

We illustrate this for the example in Figure 1(a). Relevant enhanced rules are $(\mathbf{S}, 3) \to (\mathbf{NP}, 2)\ \langle (\mathbf{VP}, 3) \rangle\ \varepsilon$ and $(\mathbf{D}, 6) \to \varepsilon\ \langle (\mathbf{the}, 6) \rangle\ \varepsilon$. With $i_0 = 3$, we may construct an enhanced derivation:

$(\mathbf{S}, 3) \Rightarrow$
$(\mathbf{NP}, 2)\ (\mathbf{VP}, 3) \Rightarrow$
$(\mathbf{D}, 1)\ (\mathbf{N}, 2)\ (\mathbf{VP}, 3)\ (\mathbf{PP}, 5) \Rightarrow \ldots \Rightarrow$
$(\mathbf{a}, 1)\ (\mathbf{hearing}, 2)\ (\mathbf{VP}, 3)\ (\mathbf{on}, 5)\ (\mathbf{the}, 6)\ (\mathbf{issue}, 7) \Rightarrow$
$(\mathbf{a}, 1)\ (\mathbf{hearing}, 2)\ (\mathbf{Aux}, 3)\ (\mathbf{V}, 4)\ (\mathbf{on}, 5)\ (\mathbf{the}, 6)\ (\mathbf{issue}, 7)\ (\mathbf{Adv}, 8) \Rightarrow \ldots \Rightarrow$
$(\mathbf{a}, 1)\ (\mathbf{hearing}, 2)\ (\mathbf{is}, 3)\ (\mathbf{scheduled}, 4)\ (\mathbf{on}, 5)\ (\mathbf{the}, 6)\ (\mathbf{issue}, 7)\ (\mathbf{today}, 8)$

Next, we interpret the enhanced derivation as a tree structure, with the right-hand side elements of an enhanced rule being the children of the left-hand side. On this tree structure, we apply π, which amounts to replacing each (q, i) by $\pi(q)$. In particular, if $\pi(q) = q$, as in the case of Figure 1(a), then each (q, i) is simply replaced by q. Hereby, a derivation maps to a unique string w as well as to a unique complete ph-tree.

For a given HCFG G, the string language $SL(G)$ generated by G is the set of all strings that can be derived and the tree language $TL(G)$ is the corresponding set of complete ph-trees. $TL_c(G)$ is the tree language that results if $\Rightarrow$ is restricted to continuous derivation.

If we restrict HCFG rules to have precisely two members in the right-hand side, then the formalism is close to the non-projective context-free dependency grammars of [14]. A more distant grammatical formalism for describing languages of discontinuous structures is [33]. Another idea that is distantly related is given by [36], which separates tree structure from surface realization. See also [34].

4 Relation to Context-Free Grammars

In this section, we investigate formal properties of HCFGs and relate them to CFGs. To make the comparison easier, we assume that the states are also used as the labels and π is the identity relation over Q. A CFG is a 4-tuple (Σ, Q, q_{init}, R), consisting of the finite set Σ of terminals, the finite set Q of nonterminals, the start symbol $q_{init} \in Q$ and the set $R \subseteq Q \times \{\Sigma \cup Q\}^*$ of context-free rules. A CFG is *epsilon-free* if no rule derives the empty string, and *linear* if its rules are of the form $q \to xZy$ where $q \in Q$, $x, y \in \Sigma^*$ and $Z \in \Sigma \cup Q$. An epsilon-free CFG is *right-linear* if its rules are of the form $q \to ar$ or of the form $q \to a$, where $q, r \in Q$ and $a \in \Sigma$.

A homomorphism over Σ^* is a function $f : \Sigma^* \to \Sigma^*$ such that $f(xy) = f(x)f(y)$ for all $x, y \in \Sigma^*$. A homomorphism f is length-preserving if $f(a) \in \Sigma$ for all $a \in \Sigma$. We extend a function on strings to a function on sets of strings in the natural way, i.e. $f(L) = \{f(x) \mid x \in L\}$.

Theorem 1 *The class of languages generated by HCFGs is closed under length-preserving homomorphisms but not under all homomorphisms.*

Proof. Given a HCFG G and a length-preserving homomorphism f, we can construct a HCFG whose language is $f(L(G))$, by replacing each occurrence of a terminal a in a rule by $f(a)$. However, if we are given a HCFG H containing just the rule $S \to \langle a \rangle$ and a homomorphism $g : a^* \to a^*$ such that $g(a) = \epsilon$, then $g(L(H)) = \{\epsilon\}$, and no HCFG can generate the empty string.

Theorem 2 *All regular subsets of Σ^+ are included in the languages generated by HCFGs.*

Proof. Let G be a right-linear CFG generating a regular subset of Σ^+. We can construct a HCFG G' generating this language as follows. For each CFG rule $q \to ar$, construct the HCFG rule $q \to \langle a \rangle r$, and for each $q \to a$, construct $q \to \langle a \rangle$.

Theorem 3 *Some linear context-free subsets of Σ^+ are not generated by any HCFG.*

Proof. Let G be the linear CFG grammar with rules $S \rightarrow aSa$, $S \rightarrow bSb$, and $S \rightarrow c$. This grammar generates the set of strings of the form xcx^R where $x \in \{a, b\}^*$ and x^R represents the reversal of x. Assume this language is generated by a HCFG. Then there must be a nonterminal B, and four strings y, y', z, z' such that $S \Rightarrow^+ ayBy'a \Rightarrow^+ aybzcz'by'a$. This can be seen by investigating derivations of strings of the form $ab^n c b^n a$. We may assume such a derivation is continuous. (If not, then we can step-wise rearrange nodes until it is continuous. Each step in effect moves a substring leftward or rightward. If this does not preserve the generated string, then another string is obtained that is not of the form xcx^R, which is a contradiction.)

On paths from S to the center c we can identify minimal subderivations of the form $D \Rightarrow^* z_1 E z_2$ such that $z_1 \neq \varepsilon$ and $z_2 \neq \varepsilon$. Note that the length of z_1 in subderivations $D \Rightarrow^* z_1 E$ as well as the length of z_2 in subderivations $D \Rightarrow^* E z_2$ are bounded, because we could otherwise generate strings not of the form xcx^R. For the same reason, the lengths of z_1 and z_2 in minimal subderivations of the form $D \Rightarrow^* z_1 E z_2$ with $z_1 \neq \varepsilon$ and $z_2 \neq \varepsilon$ are bounded. By choosing n greater than this bound gives us the required subderivation $B \Rightarrow^+ bzcz'b$ near the end of the path.

If the head leaf of B is c or is to the right of c, then some node on the path from B to the leftmost b in the subderivation is the leftmost left dependent of its parent. This node can be moved to the left, such that a different string is obtained, starting with b, while still ending on a, which would be a contradiction. The remaining case, with the head leaf of B being to the left of c, is symmetric.

Theorem 4 *Some languages generated by HCFGs are not context-free.*

Proof. Let G be the HCFG with rules $S \rightarrow a\langle S \rangle a$, $S \rightarrow b\langle S \rangle b$, and $S \rightarrow \langle c \rangle$. This grammar generates the strings of the form xcy where $x, y \in \{a, b\}^*$ and y is a permutation of x. If this were a context-free language, then the intersection with $a^*b^*ca^*b^*$ would also be context-free. However, this intersection is $a^n b^m c a^n b^m$, which is known not to be context-free.

Corollary 5 *The classes of languages generated by CFGs and by HCFGs are incomparable.*

The *permutation closure* of a string language $L \subseteq \Sigma^*$ is defined to be the language $perm(L)$ of strings that are permutations of a string in L. We extend permutation closure to HCFG rules as follows. If ρ is a rule $q \rightarrow \alpha\langle Z \rangle\beta$, then $perm(\rho)$ is the set of rules of the form $q \rightarrow \alpha'\langle Z \rangle\beta'$, where $\alpha' Z \beta'$ is a permutation of $\alpha Z \beta$; note that α' and β' need not be of the same length as α and β. The permutation closure of a HCFG G, denoted by $perm(G)$, is obtained by replacing its set of rules by the union of their permutation closures.

Theorem 6 *The class of languages generated by HCFGs is closed under permutation closure.*

Proof. It is easy to see that a language is a permutation closure of an epsilon-free context-free language (and thereby of an epsilon-free regular language) if and only if it is $SL(perm(G))$ for some HCFG G.

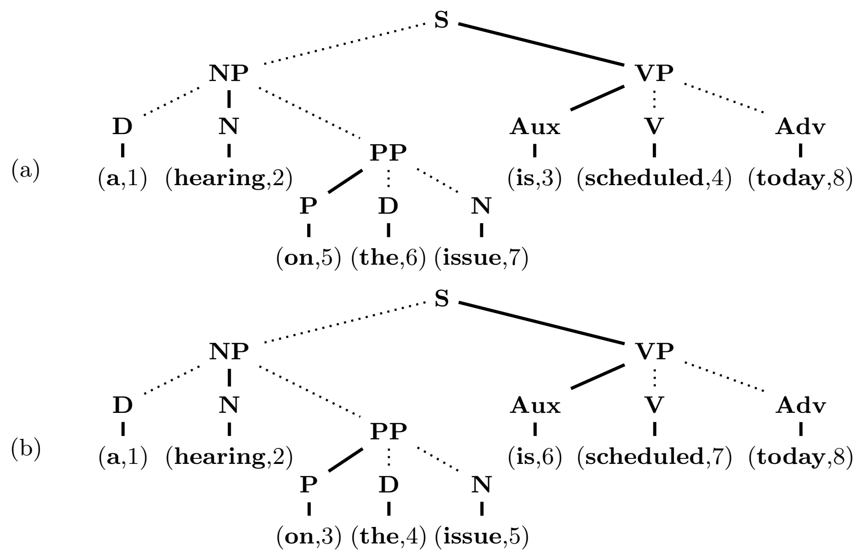


Fig. 3. The construction in the proof of Theorem 7, producing the continuous complete ph-tree in (b) from the complete ph-tree in (a), by renumbering the leaves from left to right.

In practice, it is undesirable for a model of natural language syntax to allow indiscriminate permutation of left dependents or of right dependents, let alone indiscriminate swapping of left and right dependents. This motivates considering the following weaker alternative to permutation closure. It involves shuffling the descendents of a node with descendents of other nodes, while preserving the relative order of immediate subtrees.

The *shuffle closure* of a set T of complete ph-trees $C(\Sigma, N)$, denoted by *shuffle*(T), is obtained by replacing every tree $t \in T$ by the set of structurally identical complete ph-trees.

Theorem 7 *$TL(G) = shuffle(TL_c(G))$, for each HCFG G.*

Proof. Given $t \in TL(G)$, one can construct a continuous, complete ph-tree that is structurally identical, and thereby $t \in shuffle(TL_c(G))$. This is done by drawing the tree in the traditional way, without crossing edges, and then renumbering leaves 1, 2, ..., from left to right, as illustrated in Figure 3.

The converse direction is immediate by the definition of HCFG derivation.

We have thus seen two ways of relating HCFG to continuous context-free grammar, one in terms of string languages, using the permutation closure, and one in terms of tree languages, using the shuffle closure. The string languages generated by HCFGs thus include all permutation closures of context-free languages, such as MIX, which is the language of all strings over $\{a, b, c\}$ in which the number of a's, b's and c's is equal. MIX is not context-free, and not even a tree-adjoining language [18].

Some generated languages are not in this class however. A simple example is the language of all strings over $\{a, b\}$, with equal numbers of a's and b's, but such that for any prefix, the number of a's is greater than or equal to the number of b's. This language is generated by the HCFG with rules $S \rightarrow ab\langle S\rangle$ and $S \rightarrow a\langle b\rangle$.

The ability of HCFGs to deal with shuffle closure is of great benefit to robust parsing of unrestricted natural language text. This is because idiosyncratic word order may be handled that was not seen in training data.

5 Leftmost and Rightmost Derivations

Just as in established theory of context-free grammars, there can be several derivations for the same string and the same tree that differ only in the order in which states are rewritten. For the purpose of designing effective parsers and formulating consistent probability distributions, one traditionally restricts derivations to be either leftmost or rightmost. The behavior of top-down parsers most closely matches leftmost derivations, whereas bottom-up parsers typically match (reversed) rightmost derivations.

Restricting our discontinuous derivations to be either leftmost or rightmost is more involved than in established theory, because of the potentially non-local behavior of derivation steps. Although we could define leftmost and rightmost derivations for arbitrary HCFGs, the definitions become simpler if we restrict ourselves to HCFGs that are separated; a HCFG is called *separated* if terminals only occur in rules of the form $q \rightarrow \varepsilon\langle a\rangle\varepsilon$, where $a \in \Sigma$, and all other rules are of the form $q \rightarrow \alpha\langle r\rangle\beta$, where $r \in Q$ and $\alpha, \beta \in Q^*$. The former will be called *lexical* rules and the latter *non-lexical* rules.

With a separated HCFG, we can split every sentential form into a prefix and a suffix, such that either the prefix or the suffix consists only of terminals, while the other consists only of states. In the case of a leftmost derivation, it is the prefix that consists only of terminals and in the case of a rightmost derivation, it is the suffix. As we will see later in this section, leftmost/rightmost derivations do not necessarily rewrite the leftmost/rightmost state.

For the set of states Q, we define the set of states $\hat{Q} = \{\hat{q} \mid q \in Q\}$, where each state is given a *hat*. In a rightmost derivation, the non-terminal prefix is in the set $\{\varepsilon\} \cup Q^*\hat{Q}Q^*$. In other words, either the entire sentential form consists of terminals, or there are one or more states at the beginning of the sentential form, of which precisely one has a hat. In a leftmost derivation, the states would be at the end of the sentential form.

The purpose of the hat is to link applications of rules together in a sequence of derivation steps: The state that has the hat is to be rewritten next. Which state receives the hat after the derivation step depends on whether the applied rule is lexical or non-lexical.

- If the rule is non-lexical, the hat is given to one of the states in the right hand side. With $\mathbf{NP} \rightarrow \mathbf{D}\langle\mathbf{N}\rangle\mathbf{PP}$ one possible step is:

$$\widehat{\mathbf{NP}}\ \mathbf{Aux}\ \mathbf{V}\ \mathbf{today} \quad \Rightarrow \quad \mathbf{D}\ \mathbf{N}\ \mathbf{Aux}\ \mathbf{V}\ \widehat{\mathbf{PP}}\ \mathbf{today}$$

- If the rule is lexical, the hat is given to any state in the sentential form, if there is at least one state left. With $\mathbf{Adv} \rightarrow \langle\mathbf{today}\rangle$ one possible step is:

$$\mathbf{NP}\ \mathbf{Aux}\ \mathbf{V}\ \widehat{\mathbf{Adv}} \;\Rightarrow\; \widehat{\mathbf{NP}}\ \mathbf{Aux}\ \mathbf{V}\ \mathbf{today}$$

Every derivation starts from the sentential form $\widehat{q_{init}}$. The distinction between leftmost and rightmost derivations is made by a restriction when we are allowed to apply a lexical rule, which is either when the hatted state occurs leftmost or when it occurs rightmost.

We now formally define rightmost derivations; the definition of leftmost derivations is symmetric. The binary derivation relation $\Rightarrow_{rm}$ ('rm' for 'rightmost') contains two kinds of derivation steps:

- $\gamma\hat{q}\delta w \Rightarrow_{rm} \gamma_0 q_1' \gamma_1 q_2' \cdots\ q_k' \gamma_k r' \delta_0 s_1' \delta_1 s_2' \cdots s_\ell' \delta_\ell w$ if $q \rightarrow q_1 \cdots q_k \langle r \rangle s_1 \cdots s_\ell$ is a rule, where $\gamma = \gamma_0 \cdots \gamma_k \in Q^*$, $\delta = \delta_0 \cdots \delta_\ell \in Q^*$, $w \in \Sigma^*$, q, q_1, …, q_k, r, s_1, …, $s_\ell \in Q$, and $q_1' \cdots q_k' r' s_1' \cdots s_\ell'$ is obtained from $q_1 \cdots q_k r s_1 \cdots s_\ell$ by placing the hat on exactly one of these states.
- $\gamma\hat{q}w \Rightarrow_{rm} \gamma' a w$ if $q \rightarrow \varepsilon\langle a \rangle\varepsilon$ is a rule, where $a \in \Sigma$, and $\gamma' = \gamma$ if $\gamma = \varepsilon$ and otherwise γ' is obtained from γ by placing the hat on exactly one of the states.

Perhaps counter-intuitively at first sight, our rightmost derivations do not necessarily rewrite the rightmost state. Instead, a rightmost derivation can be decomposed into several chains of rewrites, each of which ends in a step that is rightmost in the sense of adding one more terminal at the front of the suffix of terminals. After the first step in a single chain, each step rewrites a state introduced by the previous step. This constraint is enforced by placing a hat on a state to be rewritten next.

One chain in the running example starts with the hatted **NP** and ends with the addition of the token **issue** to the terminal suffix, after which a new chain starts with **D** obtaining the hat:

$$\begin{array}{l}
\widehat{\mathbf{NP}}\ \mathbf{Aux}\ \mathbf{V}\ \mathbf{today} \Rightarrow_{rm} \\
\mathbf{D}\ \mathbf{N}\ \mathbf{Aux}\ \mathbf{V}\ \widehat{\mathbf{PP}}\ \mathbf{today} \Rightarrow_{rm} \\
\mathbf{D}\ \mathbf{N}\ \mathbf{Aux}\ \mathbf{V}\ \mathbf{P}\ \mathbf{D}\ \widehat{\mathbf{N}}\ \mathbf{today} \Rightarrow_{rm} \\
\mathbf{D}\ \mathbf{N}\ \mathbf{Aux}\ \mathbf{V}\ \mathbf{P}\ \widehat{\mathbf{D}}\ \mathbf{issue}\ \mathbf{today}.
\end{array}$$

We can define SL_{lm}, TL_{lm}, SL_{rm} and TL_{rm} much as we defined SL and TL, now restricting the derivations to be leftmost or rightmost, respectively.

Of central importance to later sections is:

Theorem 8 *For each HCFG G, we have $SL(G) = SL_{lm}(G) = SL_{rm}(G)$, and $TL(G) = TL_{lm}(G) = TL_{rm}(G)$.*

A proof can be given in terms of enhanced derivations, which can be rearranged to become rightmost (or leftmost for the symmetric case). For a string of length n, there are chains of rewrites, one for each $i = n, \ldots, 1$. In the chain for a certain i, only state occurrences are rewritten whose corresponding subtrees have yields that include the i-th terminal but not the j-th terminal in the string, for any $j > i$.

Theorem 8 can be refined to formulate a surjective mapping from an arbitrary derivation to a rightmost derivation for the same tree. Moreover, if π is the identity function, then there is a bijective mapping from rightmost derivations to $TL(G)$.

6 The Shift-Reduce Automaton Model

This section defines a *discontinuous* shift-reduce parser, which computes (reversed) rightmost derivations of a separated HCFG.

6.1 General Model

A *configuration* is a pair consisting of a *stack*, which is a string in $\{\varepsilon\} \cup Q^*\hat{Q}Q^*$, and a *remaining input*, which is a string in Σ^*.

The binary relation $\vdash$ is defined by two allowable steps:

- **Shift.** A *shift* step is $(\gamma, aw) \vdash (\gamma'\hat{q}, w)$ if $q \to \varepsilon\langle a\rangle\varepsilon$ is a rule in the grammar, and γ' results from γ by removing the occurrence of the hat if there is one (if not, then $\gamma = \varepsilon$).
- **Reduction.** A *reduction* step is $(\gamma_0 q'_1 \gamma_1 q'_2 \cdots q'_k \gamma_k r' \delta_0 s'_1 \delta_1 s'_2 \cdots s'_\ell \delta_\ell, w) \vdash (\gamma_0 \cdots \gamma_k \hat{q} \delta_0 \cdots \delta_\ell, w)$ if $q \to q_1 \cdots q_k \langle r\rangle s_1 \cdots s_\ell$ is a rule in the grammar, and $q'_1 \cdots q'_k r' s'_1 \cdots s'_\ell$ contains exactly one hat, which is removed to give $q_1 \cdots q_k r s_1 \cdots s_\ell$.

Our definition of the reduction step differs from the usual definition of reduction in continuous parsing by the fact that the occurrences of symbols in the right-hand side of the used rule can be arbitrarily deep in the stack (but in the same relative order as in the rule). The location in the stack where the head is found determines the location of the left-hand side of the rule after the reduction.

A *computation* recognizing a string w is a sequence of steps $(\varepsilon, w) \vdash^* (\widehat{q_{init}}, \varepsilon)$. Here (ε, w) is the *initial* configuration and $(\widehat{q_{init}}, \varepsilon)$ is the *final* configuration. As $\vdash$ can be seen as the reversal of $\Rightarrow_{rm}$, it is not difficult to see that a string can be recognized if and only if it is in $SL(G)$, using Theorem 8. Further, computations can be enhanced to construct corresponding trees.

For the example from Figure 1, the computation is:

$$\begin{array}{ll}
(\varepsilon & ,\ \mathbf{a\ hearing\ is\ scheduled\ on\ the\ issue\ today}) \vdash \\
(\widehat{\mathbf{D}} & ,\ \mathbf{hearing\ is\ scheduled\ on\ the\ issue\ today}) \vdash \\
(\mathbf{D}\ \widehat{\mathbf{N}} & ,\ \mathbf{is\ scheduled\ on\ the\ issue\ today}) \vdash \ldots \vdash \\
(\mathbf{D\ N\ Aux\ V\ P}\ \widehat{\mathbf{D}} & ,\ \mathbf{issue\ today}) \vdash \\
(\mathbf{D\ N\ Aux\ V\ P\ D}\ \widehat{\mathbf{N}} & ,\ \mathbf{today}) \vdash \\
(\mathbf{D\ N\ Aux\ V}\ \widehat{\mathbf{PP}} & ,\ \mathbf{today}) \vdash \\
(\widehat{\mathbf{NP}}\ \mathbf{Aux\ V} & ,\ \mathbf{today}) \vdash \\
(\mathbf{NP\ Aux\ V}\ \widehat{\mathbf{Adv}} & ,\ \varepsilon) \vdash \\
(\mathbf{NP}\ \widehat{\mathbf{VP}} & ,\ \varepsilon) \vdash \\
(\widehat{\mathbf{S}} & ,\ \varepsilon)
\end{array}$$

6.2 Parsing without Grammar

It is common for modern parsers to use an implicit and binarized grammar; cf. [6]. Conceptually, there is an infinite number of rules, which are each broken up into unary and binary rules, that is, rules with one or two members in the right-hand side. We will apply this idea to the shift-reduce model above, first for the purpose of constituency parsing. The states are then parts of speech or categories, and the allowable steps are the following, where r is a part of speech, q is a category, and s is either a part of speech or a category:

- **Shift.** $(\gamma, aw) \vdash (\gamma'\hat{r}, w)$, where as before γ' results from γ by removing the occurrence of the hat if there is one.
- **Reduce right to hat.** $(\gamma_0 s \gamma_1 \hat{q} \delta, w) \vdash (\gamma_0 \gamma_1 \hat{q} \delta, w)$.
- **Reduce left to hat.** $(\gamma \hat{q} \delta_0 s \delta_1, w) \vdash (\gamma \hat{q} \delta_0 \delta_1, w)$.
- **Reduce right from hat.** $(\gamma_0 \hat{s} \gamma_1 q \delta, aw) \vdash (\gamma_0 \gamma_1 \hat{q} \delta, w)$.
- **Reduce left from hat.** $(\gamma q \delta_0 \hat{s} \delta_1, w) \vdash (\gamma \hat{q} \delta_0 \delta_1, w)$.
- **Completion.** $(\gamma \hat{s} \delta, w) \vdash (\gamma \hat{q} \delta, w)$.

The ‘reduce right to hat’ step makes s a left dependent of q. This reduction acts as part of the application of an implied rule of the form $q \to \alpha_0 s \alpha_1 \langle q' \rangle \beta$, for some α_0, α_1, q', and β. The other three types of reduction are similar. The ‘completion’ step initiates the application of an implied rule of the form $q \to \alpha \langle s \rangle \beta$, for some α and β, where none of the symbols in α and β have been identified as yet.

In line with observations made in preceding sections, we can restrict the above steps to only produce continuous trees. For example, we could impose the restriction on ‘reduce right to hat’ that $\gamma_1 = \varepsilon$; similar restrictions could be imposed on the other three reductions. With those restrictions, the hat is always on the rightmost stack element, so that the model simplifies to a traditional (binary) shift-reduce automaton.

A very similar automaton model deals with dependency parsing. One difference is that we can dispense with states, so that the symbols on the stack are input tokens. The shift step then merely transfers a token from the remaining input to the top of the stack. Moreover, we no longer need a ‘completion’ step. If once more we are to restrict this model to continuous trees, then we obtain a familiar shift-reduce parsing algorithm for projective dependency parsing, known at least since [10]; see also [25]. [27] calls this 'arc-standard parsing'.

A common way to apply shift-reduce models is to use a classifier to repeatedly determine the next step, on the basis of features extracted from the current configuration [38, 29]. Although this is not apparent in our formalization above, a configuration can be extended to keep all information about descendents of states that are currently on the stack. This means that when a final configuration is reached, there is sufficient information to construct the corresponding derivation. Further, the information about descendents is available to the features that are used by the classifier. For example, a feature could be 'the category of the leftmost constituent so far that was made a left dependent of the state that currently has the hat'.

6.3 Stack Compression

One may object however that common classifiers have one shortcoming that prevents them from serving our purpose of allowing *any* derivation, namely that the set of possible response values is normally of bounded size. Consider for example the 'reduce right to hat' step $(\gamma_0 s \gamma_1 \hat{q} \delta, w) \vdash (\gamma_0 \gamma_1 \hat{q} \delta, w)$. Assume that q is the i-th state in the stack, or in other words $i = |\gamma_0 s \gamma_1| + 1$, and s is the j-th state, or in other words $j = |\gamma_0| + 1$. Then we call $j - i$ the *fellow index*, the *fellow* being the other stack element, next to the one with the hat, that is involved in the reduction. The fellow index is defined analogously for the other three reductions. If the fellow index is negative, the fellow is to the left of the hatted element, and if it is positive, then it is to the right.

A deterministic parser may determine the next step by a classifier that picks a shift, a 'reduce to hat', a 'reduce from hat', or (only in the case of constituency parsing) a 'completion'. If a reduction is chosen, then a second classifier may be used to determine the fellow index. In the worst case, a desired fellow index for a given configuration may be different from any of the fellow indices seen during training, which would imply the parser is incomplete, i.e. there are derivations it is unable to produce.

To mitigate this, we set a small constant N, which acts as the window size around the hatted stack symbol; in our experiments, $N = 2$. Further, we let the classifier for the fellow index return one of two kinds of objects:

- a non-zero number no smaller than $-N$ and no greater than N, or
- a pair (d, q), where $d \in \{\mathbf{left}, \mathbf{right}\}$, and q is a part of speech or (in the case of a constituency parser) a category.

If the classifier returns a non-zero number, then that uniquely identifies the fellow element, which must then be no further than N positions away from the hatted symbol. If however a pair (d, q) is returned, then the fellow is taken to be the first stack symbol in direction d, starting from the hatted position minus $N + 1$ if $d = \mathbf{left}$, or plus $N + 1$ if $d = \mathbf{right}$, that is labeled by q. Note that this may in principle correspond to an arbitrarily large fellow index.

To allow the parser to detect the existence of parts of speech or categories outside the window of size N around the hatted symbol, two sets will be constructed, namely the set of parts of speech or categories to the left of that window, and the set of parts of speech or categories to its right. We will refer to this as the *stack compression*, as it compresses the full stack into a bounded number of predictors, in combination with a bounded number of potential response values. It should be pointed out that stack compression mitigates the incompleteness of the parser, but cannot entirely remove it.

7 A Probabilistic Model of Derivations

There may be many derivations for a given string, each of which determines one tree. In order to disambiguate, and choose one out of those derivations and thereby one tree, one may impose a probability model on derivation steps. We investigate such a model here, as before for a fixed HCFG that is separated.

The task ahead is to define a probability distribution over the derivation steps that are possible for a sentential form $\gamma\hat{q}\delta w$. A derivation step is characterized first by a choice of a non-lexical rule $q \rightarrow \alpha\langle r\rangle\beta$ where $r \in Q$, $\alpha = q_1 \cdots q_k$ and $\beta = s_1 \cdots s_\ell$ or a lexical rule $q \rightarrow \varepsilon\langle a\rangle\varepsilon$ where $a \in \Sigma$; the latter is only possible if $\delta = \varepsilon$.

- When a non-lexical rule $q \rightarrow \alpha\langle r\rangle\beta$ is applied, we also need to choose the way in which γ and δ are broken up into $k+1$ and $\ell+1$ pieces, respectively, as $\gamma = \gamma_0 \cdots \gamma_k$ and $\delta = \delta_0 \cdots \delta_\ell$, to accommodate for the placement of the elements of α and β, and we need to choose, from $k+1+\ell$ possible states in $\alpha\, r\beta$, the m-th state that will have the hat next. The sentential form after this step will be $\gamma_0 q'_1\gamma_1 \cdots q'_k\gamma_k r'\delta_0 s'_1\delta_1 \cdots s'_\ell\delta_\ell w$, where $q'_1 \cdots q'_k r' s'_1 \cdots s'_\ell$ is obtained from $q_1 \cdots q_k r s_1 \cdots s_\ell$ by placing the hat on the m-th state.
- When a lexical rule $q \rightarrow \varepsilon\langle a\rangle\varepsilon$ is applied, we need to choose the state from among the $|\gamma|$ states in the sentential form $\gamma a w$ that will have the hat next. The sentential form after this step will be $\gamma' a w$, where γ' is obtained from γ by placing the hat on the m-th state.

7.1 Independence Assumptions

In order to obtain a model that can be effectively trained, we need to make a number of independence assumptions. For the probability of a step $\gamma\hat{q}\delta w \Rightarrow_{rm} \gamma_0 q'_1\gamma_1 \cdots q'_k\gamma_k r'\delta_0 s'_1\delta_1 \cdots s'_\ell\delta_\ell w$ as above, we assume that the choice of the non-lexical rule $q \rightarrow \alpha\langle r\rangle\beta$ depends only on q, and on whether δ is empty, because if δ is empty, the probability mass may need to be shared with one or more lexical rules. We further assume that how γ is broken up into $k+1$ pieces depends only on k and $|\gamma|$. Similarly, how δ is broken up only depends on ℓ and $|\delta|$. Lastly, we assume that m, the index of the state from among $k+1+\ell$ states that is given the hat, depends only on k and ℓ.

Altogether, the probability of the step is approximated by:

$$\begin{array}{l} p_{rule}(q \to q_1 \cdots q_k \langle r \rangle s_1 \cdots s_\ell \mid q, B(\delta)) \cdot \\ p_{left}(|\gamma_0|, \ldots, |\gamma_k| \mid k, |\gamma|) \cdot \\ p_{right}(|\delta_0|, \ldots, |\delta_\ell| \mid \ell, |\delta|) \cdot \\ p_{rule\_hat}(m \mid k, \ell) \end{array}$$

where $B(\delta)$ is the truth value of $\delta = \varepsilon$.

Let us now consider a step $\gamma \hat{q} \delta w \Rightarrow_{rm} \gamma' \delta a w$, using a lexical rule. As lexical rules can only rewrite states that occur rightmost in the sentential form, the probability of such a step would be 0 if $\delta \neq \varepsilon$. If $\delta = \varepsilon$, we make the independence assumption that the choice of the lexical rule $q \to \varepsilon\langle a \rangle\varepsilon$ depends only on q, and that m, the index of the state from among $|\gamma|$ states that is given the hat, depends only on $|\gamma|$. Altogether, the probability of the step is approximated by:

$$\begin{array}{l} p_{rule}(q \to \varepsilon\langle a \rangle\varepsilon \mid q, B(\delta)) \cdot \\ p_{lex\_hat}(m \mid |\gamma|) \end{array}$$

7.2 Component Models

The component models pertaining to discontinuity are defined recursively, motivated by the assumption that the probability decreases exponentially, for each choice that introduces more discontinuity into the derivation under construction. The four parameters of these models are the probabilities P_{left}, P_{right}, P_{rule_hat}, and P_{lex_hat}. We further define their complements $\overline{P_{left}} = 1 - P_{left}$, $\overline{P_{right}} = 1 - P_{right}$, etc.

In the definition of the conditional probability distribution p_{left}, P_{left} is the probability that the next left dependent is *not* placed rightmost among the remaining states in the sentential form, provided there are any left. For non-negative integers k, i_0, ..., i_k, and $i = i_0 + \cdots + i_k$, we let:

$$p_{left}(i_0, \ldots, i_k \mid k, i) = \begin{cases} 1, & \text{if } k = 0, \\ P_{left}^{i_k}, & \text{if } k > 0 \wedge i_0 + \cdots + i_{k-1} = 0, \\ p_{left}(i_0, \ldots, i_{k-1} \mid k-1, i - i_k) \cdot \overline{P_{left}} \cdot P_{left}^{i_k}, & \text{if } k > 0 \wedge i_0 + \cdots + i_{k-1} > 0 \end{cases}$$

This is illustrated in Figure 4. In (a), q_2 is *not* placed rightmost, immediately next to r, which amounts to one factor P_{left}. It is placed at the next position, which amounts to a factor $\overline{P_{left}}$. Then q_1 is placed four positions further to the left, which amounts to four factors P_{left}, etc. In (b), q_2 is three positions further to the left than immediately next to r. After this, there is no remaining choice where q_1 and q_0 can be placed, so there are no further factors. Note that $\sum_{i_0,\ldots,i_k} p_{left}(i_0, \ldots, i_k \mid k, i) = 1$, for each k and i, as before under the constraint $i = i_0 + \cdots + i_k$. We define p_{right} symmetrically, where P_{right} is the probability

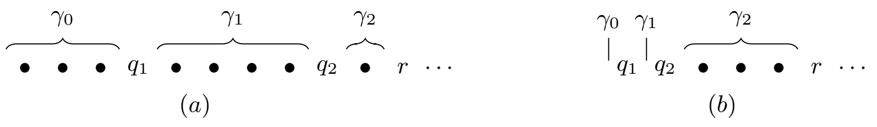


Fig. 4. (a) For $k = 2$, $|\gamma| = 8$, $|\gamma_0| = 3$, $|\gamma_1| = 4$, $|\gamma_2| = 1$, we have $p_{left}(3, 4, 1 \mid 2, 8) = \overline{P_{left}} \cdot P_{left}^4 \cdot \overline{P_{left}} \cdot P_{left}^1$. (b) For $k = 2$, $|\gamma| = 3$, $|\gamma_0| = 0$, $|\gamma_1| = 0$, $|\gamma_2| = 3$, we have $p_{left}(0, 0, 3 \mid 2, 3) = P_{left}^3$.

that the next right dependent is *not* placed leftmost among the remaining states in the sentential form, provided there are any left.

By the same reasoning, we define the conditional probability distribution p_{rule_hat}, using P_{rule_hat}, which is the probability that the hat is *not* given to the next state, from right to left. There is one such factor for each state that is skipped, and one more factor $\overline{P_{rule\_hat}}$ provided the hat is given to a state that is not leftmost. For non-negative integers k and ℓ and $1 \leq m \leq k + \ell + 1$, we let:

$$p_{rule\_hat}(m \mid k, \ell) = \begin{cases} P_{rule\_hat}^{k+\ell}, & \text{if } m = 1, \\ \overline{P_{rule\_hat}} \cdot P_{rule\_hat}^{k+\ell+1-m}, & \text{if } m > 1 \end{cases}$$

Note that $\sum_m p_{rule\_hat}(m \mid k, \ell) = 1$. The definitions of p_{lex_hat} and P_{lex_hat} are analogous.

7.3 Consistency

With the usual assumption of absence of useless rules, a sufficient condition for the above equations to specify a consistent probability model is that there is at least one rule $q \to \alpha\langle r\rangle\beta$ with non-empty β for each q. To illustrate the problem that is potentially caused if this requirement is not satisfied, assume the hat is placed on a state q in the sentential form that is not rightmost, and assume only lexical rules exist that have q as left-hand side. Then no rules at all are applicable, and probability mass is lost.

Such a problem in fact had to be solved for our experiments in Section 8 with dependency grammars, which are formalized in terms of rules $q_{init} \to \varepsilon\langle\overline{A}\rangle\varepsilon$, $\overline{A} \to \alpha\langle A\rangle\beta$, and $A \to \varepsilon\langle a\rangle\varepsilon$, where state A represents a part of speech and $\overline{A}$ is an auxiliary state for the same part of speech, α and β are strings of such auxiliary states for parts of speech, and a is a word. (Cf. our discussion about unilexical ph-trees in Section 2.) For each A there is one smoothed bigram model p_{ld} for possible choices of left dependents α and one such model p_{rd} for right dependents β. To avoid problems caused by out-of-vocabulary words and inflection, probabilities of rules $A \to \varepsilon\langle a\rangle\varepsilon$ are ignored, so that conceptually the input consists of a string of parts of speech.

In order to then obtain a probability distribution over derivations, one needs to ensure that the hat cannot be given to a part of speech that is non-rightmost in the sentential form. This is achieved by adjusting the definitions of p_{rule_hat}

and p_{lex_hat} to ignore parts of speech unless they are right-most in the sentential form. We further ensure that an auxiliary state $\overline{A}$ that occurs non-rightmost in the sentential form can only be rewritten using $\overline{A} \rightarrow \alpha\langle A\rangle\beta$ if $\alpha\beta \neq \varepsilon$, with probability $p_{ld}(\alpha \mid A) \cdot p_{rd}(\beta \mid A) \cdot C_{norm}(A)$, with the normalization factor $C_{norm}(A) = 1/(1 - p_{ld}(\varepsilon \mid A) \cdot p_{rd}(\varepsilon \mid A))$. For rightmost occurrences of $\overline{A}$ there is no such restriction on α and β and the normalization factor is not needed.

8 Evaluation

We carried out two kinds of investigation. First, we evaluated our shift-reduce model for the purpose of deterministic parsing. Second, we investigated the empirical behaviour of the probabilistic model of derivations. The first investigation was done for both dependency parsing and for constituency parsing, while the second has been done only for dependency parsing at this time. For constituency parsing, we considered the TIGER [3] and NEGRA [4] treebanks. Punctuation was transferred to a lower level in the trees using Treetools.[1]

For dependency parsing we have taken some of the largest corpora from Universal Dependencies v2.2[2], to be precise, those that contain at least 14,000 projective trees in the training sections. In addition we have taken treebanks for Ancient Greek and German, as these are known to have a large proportion of nonprojective trees, and are therefore central to our theory. Altogether, the considered languages are Arabic (ar; NYUAD), Czech (cs; CAC), German (de; GSD), Estonian (et; EDT), Ancient Greek (grc; PROIEL), Japanese (ja; BCCWJ), Korean (ko; Kaist), Norwegian (no; Bokmaal), Russian (ru; SynTagRus).

8.1 Experiments with Deterministic Parsing

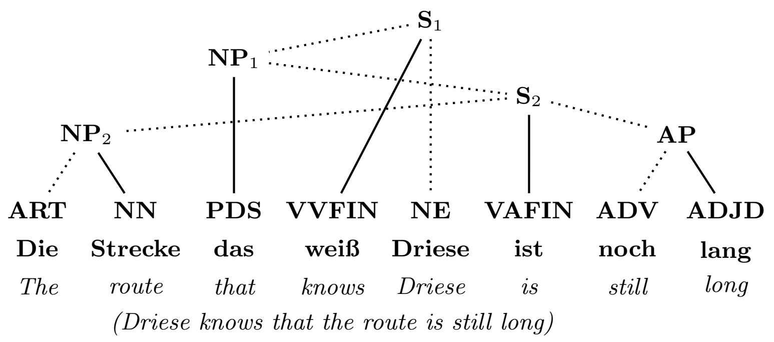
We implemented the nonprojective as well as the projective shift-reduce models described in Section 6, for both dependency parsing and constituency parsing. For comparison, we also implemented the model of [28, 30] with swapping transitions. Not considered were models with a very different architecture, such as that of [9].

For constituency parsing, one still needs to decide how to determine the head children of nodes, as in the NEGRA and TIGER treebanks, the heads are only explicitly represented for a portion of the categories. There are three obvious strategies. In the first, the explicit representation of heads is complemented with hand-written heuristic rules. In the second and third strategies, the heads are always chosen to be the left-most child or the right-most child, respectively, even where the treebank has explicit representation of heads. In this section, we assume the first strategy, which appears to perform marginally better in the case of discontinuous parsing.

[1] `https://github.com/wmaier/treetools`, by Wolfgang Maier. Placing punctuation was done using `--trans root_attach`.

[2] `https://universaldependencies.org/`

(a)


S1
NP1
S2
NP2
AP
ART NN PDS VVFIN NE VAFIN ADV ADJD
Die Strecke das weiß Driese ist noch lang
The route that knows Driese is still long
(Driese knows that the route is still long)

(b) stack: $\mathbf{NP}_2$ $\mathbf{NP}_1$ $\mathbf{S}_1$ **NE** $\widehat{\mathbf{S}}_2$ remaining input: **noch** **lang**

Fig. 5. The correct structure is as in (a), ignoring punctuation for presentational convenience, and with added subscripts for ease of reference (the subscripts are *not* part of the category names). After a number of steps, the configuration in (b) is reached. The correct next step is a reduction of $\mathbf{NP}_2$ to $\mathbf{S}_2$. If we assume $N = 2$, then stack compression implies that the fellow is taken to be $\mathbf{NP}_1$ instead, as this is the first **NP** to the left of the window of size 2 around the hat.

As the classifiers used in our implementation were not advanced enough to reach state-of-the-art parsing accuracies, it was decided to omit accuracy figures. Instead, we have investigated the behaviour of stack compression in practice. A central question is how often gold structures can be outside the reach of the parser. This can happen if the index of the hatted element is i, and the fellow index j is smaller than $-N$ or greater than N. In this case the classifier should return the fellow in the form of a pair (d, q), where $d \in \{\mathbf{left}, \mathbf{right}\}$ and q is a part of speech or category. If say $d = \mathbf{left}$, then this is intended to refer to a fellow with negative fellow index j that is the first stack element starting at index $i - N - 1$, searching downward, that has label q. If however the correct fellow with label q is at stack element $i + j$, but there is another stack element between $i + j$ and $i - N - 1$ with the same label, then the fellow is misidentified. Figure 5 illustrates a typical case.

Table 1 shows the distribution of the fellow indices in the case of constituency parsing, with $N = 3$. As expected, the most frequent fellow index is -1, which typically corresponds to a reduction of the topmost two stack elements. Also indicated in the table is the relative frequency of the representation (d, q) not being information-preserving, leading to misidentification of the fellow. It turns out this is exceedingly rare but does happen.

Table 2 shows the distribution of the fellow indices in the case of dependency parsing, with $N = 2$, for six languages that show notable differences. As expected, the pervasive nonprojectivity in Ancient Greek corresponds to a fellow index which is frequently a value other than -1, whereas this is very infrequent in the case of Japanese. For Korean the fellow index is often a value other than -1,

Table 1. The distributions of the fellow indices, for the TIGER and NEGRA treebanks. Also indicated are the relative frequencies of fellow indices being translated to pairs (d, q), and relative frequency of this not being information-preserving, due to an intervening stack element with the same label q.

TIGER

index	'reduce to hat'	'reduce from hat'
-11		0.0002%
-10	0.0006%	0.0007%
-9	0.0003%	0.0023%
-8	0.0012%	0.0032%
-7	0.0088%	0.0086%
-6	0.0143%	0.0273%
-5	0.0326%	0.0744%
-4	0.0707%	0.2497%
-3	0.2722%	0.9624%
-2	2.7642%	0.6103%
-1	93.7668%	96.9776%
1	3.0605%	1.0718%
2	0.0006%	0.0104%
3	0.0058%	0.0004%
4	0.0009%	0.0002%
5	0.0003%	0.0005%
compressed	0.1299%	0.3672%
unpreserved	0.0055%	0.0157%

NEGRA

index	'reduce to hat'	'reduce from hat'
-10	0.0008%	
-9	0.0008%	0.0004%
-8	0.0055%	0.0026%
-7	0.0148%	0.0110%
-6	0.0203%	0.0287%
-5	0.0382%	0.0789%
-4	0.0647%	0.2298%
-3	0.2914%	0.8552%
-2	3.1022%	0.6351%
-1	93.5331%	97.1707%
1	2.9253%	0.9774%
2	0.0016%	0.0093%
3	0.0016%	0.0009%
compressed	0.1449%	0.3515%
unpreserved	0.0062%	0.0123%

but only in the case of 'reduce to hat', not 'reduce from hat'. The frequent value of -2 corresponds to a left dependent that is separated from the head by one other element, without further sources of non-projectivity. For all of these six languages, it is very rare for stack compression not to be information-preserving, even in the case of Ancient Greek.

8.2 Experiments with the Probabilitistic Model of Derivations

The probabilistic model of derivations was trained on the nine dependency treebanks mentioned before. The estimated parameters related to nonprojectivity are presented in Table 3. It is unsurprising that all of these parameters are high for languages with much nonprojectivity, such as Ancient Greek, and all are low for languages with largely projective structures, such as Japanese. Once more, Korean distinguishes itself from the other considered languages; we see that the fair amount of nonprojectivity is concentrated in P_{left}.

There are tabular parsing methods with polynomial time complexity that can be used to determine the most probable tree given a sentence and a probabilistic context-free grammar [15]. This does not seem possible for our model of discontinuous trees however. At best the model could evaluate (parts of) candidate parses provided by other models, possibly including the deterministic

Table 2. The distributions of the fellow indices, for the training sections of six dependency treebanks. (Zero entries are left blank.)

	cs		de		grc	
index	‘reduce to hat’	‘reduce from hat’	‘reduce to hat’	‘reduce from hat’	‘reduce to hat’	‘reduce from hat’
$\leq$ -4	0.0051%	0.0311%	0.0051%	0.0911%	0.0775%	0.2823%
-3	0.0828%	0.1160%	0.0238%	0.3837%	0.6058%	1.0507%
-2	1.6567%	0.5024%	0.4899%	1.3132%	5.0079%	4.6467%
-1	98.0630%	98.9989%	98.8159%	97.6579%	93.2501%	91.8049%
1	0.1916%	0.3352%	0.6633%	0.5522%	1.0256%	2.0958%
2	0.0008%	0.0164%	0.0019%	0.0018%	0.0217%	0.1151%
$\geq$ 3					0.0113%	0.0044%
compressed	0.0879%	0.1471%	0.0290%	0.4749%	0.6947%	1.3374%
unpreserved	0.0004%	0.0050%		0.0138%	0.0103%	0.0310%

	ja		ko		no	
index	‘reduce to hat’	‘reduce from hat’	‘reduce to hat’	‘reduce from hat’	‘reduce to hat’	‘reduce from hat’
$\leq$ -4	0.0023%	0.0002%	0.0030%		0.0039%	0.1368%
-3	0.0013%		0.0973%		0.0063%	0.2478%
-2	0.0139%		3.6465%		0.2327%	0.6246%
-1	99.9814%	99.9996%	96.2532%	100.0000%	99.2315%	98.1666%
1	0.0011%	0.0002%			0.5256%	0.7932%
2						0.0258%
$\geq$ 3						0.0052%
compressed	0.0036%	0.0002%	0.1002%		0.0102%	0.3897%
unpreserved	0.0009%		0.0005%			0.0172%

shift-reduce models in the previous section. For combining parsers, see also [13, 40].

We can however investigate how accurate a trained model is by measuring cross-entropy of a test corpus, which we take here to be the negative log likelihood of that corpus, normalized by the length of the corpus, or formally $\frac{-\sum_{t \in T} \log_2 p(t)}{\sum_{t \in T} |yield(t)|}$. Here T is the set of trees in the test corpus, p is the trained probability model of rightmost derivations, and $|yield(t)|$ is the number of tokens in t. A closely related concept is perplexity, which is 2 raised to the power of the cross-entropy. Perplexity was shown by [37] to be a good indicator of parsing accuracy.

Table 4 presents cross-entropy, labelled as p in the final column, measured on the testing sections of the treebanks. It is the sum of the preceding six columns. The first four correspond to the models governing nonprojectivity (cf. Table 3). In addition there are the normalization constant discussed in Section 7.3, and the model of producing the left and right dependents.

The largest portion of the cross-entropy is made up of the submodels of left and right dependents, with relatively minor contributions from the submodels that account for nonprojectivity. This could mean that nonprojectivity is rela-

Table 3. Estimated parameters pertaining to discontinuity.

	P_{lex_hat}	P_{rule_hat}	P_{left}	P_{right}
ar	$2.1*10^{-3}$	$3.1*10^{-3}$	$3.9*10^{-3}$	$8.9*10^{-3}$
cs	$2.0*10^{-3}$	$3.8*10^{-3}$	$1.4*10^{-2}$	$2.3*10^{-2}$
de	$5.1*10^{-3}$	$5.0*10^{-3}$	$9.7*10^{-3}$	$4.2*10^{-2}$
et	$5.9*10^{-4}$	$1.2*10^{-3}$	$4.5*10^{-3}$	$5.3*10^{-3}$
grc	$1.5*10^{-2}$	$3.8*10^{-2}$	$5.7*10^{-2}$	$1.5*10^{-1}$
ja	0.0	$2.1*10^{-5}$	$2.3*10^{-4}$	$4.0*10^{-5}$
ko	0.0	0.0	$6.1*10^{-2}$	0.0
no	$4.7*10^{-3}$	$9.6*10^{-3}$	$3.7*10^{-3}$	$4.5*10^{-2}$
ru	$1.2*10^{-3}$	$7.6*10^{-3}$	$8.0*10^{-3}$	$2.8*10^{-2}$

Table 4. Cross-entropy and its decomposition.

	p_{lex_hat}	p_{rule_hat}	p_{left}	p_{right}	C_{norm}	p_{rule}	p
ar	$2.0*10^{-2}$	$7.4*10^{-3}$	$8.7*10^{-3}$	$2.3*10^{-2}$	$-2.4*10^{-3}$	3.88	3.93
cs	$1.6*10^{-2}$	$1.5*10^{-2}$	$5.6*10^{-2}$	$2.6*10^{-2}$	$-2.7*10^{-3}$	3.78	3.89
de	$1.3*10^{-1}$	$2.3*10^{-2}$	$5.1*10^{-2}$	$1.1*10^{-1}$	$-8.9*10^{-3}$	3.86	4.16
et	$1.1*10^{-2}$	$3.6*10^{-3}$	$2.5*10^{-2}$	$1.2*10^{-2}$	$-8.2*10^{-4}$	4.13	4.18
grc	$9.7*10^{-2}$	$9.5*10^{-2}$	$1.7*10^{-1}$	$1.3*10^{-1}$	$-1.9*10^{-2}$	4.08	4.56
ja	0.0	$7.7*10^{-6}$	$2.6*10^{-3}$	$1.1*10^{-5}$	0.0	3.85	3.85
ko	0.0	0.0	$2.3*10^{-1}$	0.0	0.0	4.16	4.38
no	$4.4*10^{-2}$	$2.4*10^{-2}$	$1.9*10^{-2}$	$5.9*10^{-2}$	$-9.3*10^{-3}$	3.82	3.95
ru	$7.8*10^{-3}$	$2.4*10^{-2}$	$3.5*10^{-2}$	$2.6*10^{-2}$	$-3.0*10^{-3}$	4.07	4.16

tively well modeled already, and that for making the overall model more accurate, the most gain is to be expected from the submodels of left and right dependents.

9 Conclusion

We have introduced a model of syntax that captures discontinuous derivations. It allows the definition of probability distributions over the space of all discontinuous parses. We have shown that this allows evaluation in terms of cross-entropy. We have also introduced a new kind of automaton, which is such that each derivation corresponds in a natural way to a computation of an automaton, and vice versa. These automata may be used for deterministic parsing. Because our model captures both probabilistic grammatical formalisms and parsing algorithms, it is more comprehensive than previous models of discontinuous syntax.

Acknowledgments

Our implementation of the parsers built on unpublished work by Billy Brown [5]. Helpful comments from anonymous referees are gratefully acknowledged.

References

1. Alshawi, H.: Head automata and bilingual tiling: Translation with minimal representations. In: 34th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference. pp. 167–176. Santa Cruz, California, USA (Jun 1996)
2. Boyd, A.: Discontinuity revisited: An improved conversion to context-free representations. In: Proceedings of the Linguistic Annotation Workshop, at ACL 2007. pp. 41–44. Prague, Czech Republic (Jun 2007)
3. Brants, S., Dipper, S., Eisenberg, P., Hansen-Schirra, S., König, E., Lezius, W., Rohrer, C., Smith, G., Uszkoreit, H.: TIGER: Linguistic interpretation of a German corpus. Research on Language and Computation 2, 597–620 (2004)
4. Brants, T., Skut, W., Uszkoreit, H.: Syntactic annotation of a German newspaper corpus. In: Abeillé, A. (ed.) Treebanks: Building and Using Parsed Corpora, Text, Speech and Language Technology, vol. 20, chap. 5, pp. 73–87. Springer (2003)
5. Brown, B.: Hat method of non-projective dependency parsing: implementation and evaluation. Individual Masters project, University of St Andrews, School of Computer Science (2017)
6. Collins, M.: Head-driven statistical models for natural language parsing. Computational Linguistics 29(4), 589–637 (2003)
7. Daniels, M., Meurers, W.: Improving the efficiency of parsing with discontinuous constituents. In: Wintner, S. (ed.) Proceedings of NLULP'02: The 7th International Workshop on Natural Language Understanding and Logic Programming. Datalogiske Skrifter, vol. 92, pp. 49–68. Roskilde Universitetscenter, Copenhagen (2002)
8. Evang, K., Kallmeyer, L.: PLCFRS parsing of English discontinuous constituents. In: Proceedings of the 12th International Conference on Parsing Technologies. pp. 104–116. Dublin, Ireland (Oct 2011)
9. Fernández-González, D., Gómez-Rodríguez, C.: Non-projective dependency parsing with non-local transitions. In: Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. vol. 2, pp. 693–700. New Orleans, Louisiana (2018)
10. Fraser, N.: Parsing and dependency grammar. UCL Working Papers in Linguistics 1, 296–319 (1989)
11. Gaifman, H.: Dependency systems and phrase-structure systems. Information and Control 8, 304–337 (1965)
12. Hays, D.: Dependency theory: A formalism and some observations. Language 40(4), 511–525 (1964)
13. Henderson, J., Brill, E.: Exploiting diversity in natural language processing: Combining parsers. In: Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora. pp. 187–194. University of Maryland, USA (Jun 1999)
14. Holan, T., Kuboň, V., Plátek, M.: An implementation of syntactic analysis of Czech. In: Fourth International Workshop on Parsing Technologies. pp. 126–135. Prague and Karlovy Vary, Czech Republic (Sep 1995)

15. Jelinek, F., Lafferty, J., Mercer, R.: Basic methods of probabilistic context free grammars. In: Laface, P., De Mori, R. (eds.) Speech Recognition and Understanding — Recent Advances, Trends and Applications, pp. 345–360. Springer-Verlag (1992)
16. Kahane, S., Nasr, A., Rambow, O.: Pseudo-projectivity, a polynomially parsable non-projective dependency grammar. In: 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics. vol. 1, pp. 646–652. Montreal, Quebec, Canada (Aug 1998)
17. Kallmeyer, K., Kuhlmann, M.: A formal model for plausible dependencies in lexicalized tree adjoining grammar. In: Eleventh International Workshop on Tree Adjoining Grammar and Related Formalisms. pp. 108–116. Paris, France (2012)
18. Kanazawa, M., Salvati, S.: MIX is not a tree-adjoining language. In: 50th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference. pp. 666–674. Jeju Island, Korea (Jul 2012)
19. Kathol, A., Pollard, C.: Extraposition via complex domain formation. In: 33rd Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference. pp. 174–180. Cambridge, Massachusetts, USA (Jun 1995)
20. Kuhlmann, M.: Mildly non-projective dependency grammar. Computational Linguistics 39(2), 355–387 (2013)
21. Ljunglöf, P., Wirén, M.: Syntactic parsing. In: Indurkhya, N., Damerau, F. (eds.) Handbook of Natural Language Processing. Second Edition, pp. 77–91. CRC Press, Taylor and Francis Group (2010)
22. Maier, W.: Discontinuous incremental shift-reduce parsing. In: 53rd Annual Meeting of the Association for Computational Linguistics and 7th International Joint Conference on Natural Language Processing. vol. 1, pp. 1202–1212. Beijing (Jul 2015)
23. Marcus, S.: Algebraic Linguistics; Analytical Models, Mathematics in Science and Engineering, vol. 29. Academic Press, New York and London (1967)
24. Narayan, S., Cohen, S.: Optimizing spectral learning for parsing. In: 54th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference. vol. 1, pp. 1546–1556. Berlin (Aug 2016)
25. Nasr, A.: A formalism and a parser for lexicalised dependency grammars. In: Fourth International Workshop on Parsing Technologies. pp. 186–195. Prague and Karlovy Vary, Czech Republic (Sep 1995)
26. Nederhof, M.J., Vogler, H.: Hybrid grammars for discontinuous parsing. In: The 25th International Conference on Computational Linguistics: Technical Papers. pp. 1370–1381. Dublin, Ireland (Aug 2014)
27. Nivre, J.: Algorithms for deterministic incremental dependency parsing. Computational Linguistics 34(4), 513–553 (2008)
28. Nivre, J.: Non-projective dependency parsing in expected linear time. In: Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP. pp. 351–359. Suntec, Singapore (Aug 2009)
29. Nivre, J., Hall, J., Nilsson, J.: Memory-based dependency parsing. In: Proceedings of the Eighth Conference on Computational Natural Language Learning. pp. 49–56. Boston, Massachusetts (May 2004)
30. Nivre, J., Kuhlmann, M., Hall, J.: An improved oracle for dependency parsing with online reordering. In: Proceedings of the 11th International Conference on Parsing Technologies. pp. 73–76. Paris, France (Oct 2009)

31. Nivre, J., Nilsson, J.: Pseudo-projective dependency parsing. In: 43rd Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference. pp. 99–106. Association for Computational Linguistics, Ann Arbor, Michigan (Jun 2005)
32. Petrov, S., Barrett, L., Thibaux, R., Klein, D.: Learning accurate, compact, and interpretable tree annotation. In: Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics. pp. 433–440. Sydney, Australia (Jul 2006)
33. Plaehn, O.: Computing the most probable parse for a discontinuous phrase structure grammar. In: Proceedings of the Sixth International Workshop on Parsing Technologies. pp. 195–206. Trento, Italy (Feb 2000)
34. Ranta, A.: Grammatical framework: A type-theoretical grammar formalism. Journal of Functional Programming 14(2), 145–189 (Mar 2004)
35. Reape, M.: A logical treatment of semi-free word order and bounded discontinuous constituency. In: Fourth Conference of the European Chapter of the Association for Computational Linguistics, Proceedings of the Conference. pp. 103–110. Manchester, England (Apr 1989)
36. Samuelsson, C.: A statistical theory of dependency syntax. In: The 18th International Conference on Computational Linguistics. vol. 2, pp. 684–690. Saarbrücken, Germany (Jul–Aug 2000)
37. Søgaard, A., Haulrich, M.: On the derivation perplexity of treebanks. In: Proceedings of the Ninth International Workshop on Treebanks and Linguistic Theories. pp. 223–232. Tartu, Estonia (Dec 2010)
38. Yamada, H., Matsumoto, Y.: Statistical dependency analysis with support vector machines. In: 8th International Workshop on Parsing Technologies. pp. 195–206. LORIA, Nancy, France (Apr 2003)
39. Yli-Jyrä, A.: Approximating dependency grammars through intersection of star-free regular languages. International Journal of Foundations of Computer Science 16(3), 565–579 (2005)
40. Zeman, D., Žabokrtský, Z.: Improving parsing accuracy by combining diverse dependency parsers. In: Proceedings of the Ninth International Workshop on Parsing Technologies. pp. 171–178. Vancouver, British Columbia, Canada (Oct 2005)