Tractable Parsing for CCGs of Bounded Degree

Lena Katharina Schiffer
Leipzig University
Faculty of Mathematics and
Computer Science
schiffer@informatik.uni-leipzig.de

Marco Kuhlmann
Linköping University
Department of Computer and
Information Science
marco.kuhlmann@liu.se

Giorgio Satta
University of Padua
Department of Information Engineering
satta@dei.unipd.it

Unlike other mildly context-sensitive formalisms, Combinatory Categorial Grammar (CCG) cannot be parsed in polynomial time when the size of the grammar is taken into account. Refining this result, we show that the parsing complexity of CCG is exponential only in the maximum degree of composition. When that degree is fixed, parsing can be carried out in polynomial time. Our finding is interesting from a linguistic perspective because a bounded degree of composition has been suggested as a universal constraint on natural language grammar. Moreover, ours is the first complexity result for a version of CCG that includes substitution rules, which are used in practical grammars but have been ignored in theoretical work.

1. Introduction

Combinatory Categorial Grammar (CCG; Steedman and Baldridge 2011) is one of the standard grammar formalisms in computational linguistics and natural language processing. It is usually counted among the so-called mildly context-sensitive formalisms, which were introduced by Joshi (1985). This classification is based on two celebrated theoretical results: First, in a result about *generative power*, Weir and Joshi (1988) and Vijay-Shanker and Weir (1994) proved that CCG generates the same string languages as three other extensions of context-free grammars: Head Grammars, Linear

Action Editor: Carlos Gómez-Rodríguez. Submission received: 22 December 2021; accepted for publication: 25 February 2022.

https://doi.org/10.1162/coli_a_00441

© 2022 Association for Computational Linguistics
Published under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International (CC BY-NC-ND 4.0) license

Indexed Grammars, and Tree Adjoining Grammars (TAGs)—the perhaps best-known mildly context-sensitive formalism. Second, in a result about *computational power*, Vijay-Shanker and Weir (1990, 1993) showed that CCG can be parsed in polynomial time with respect to the length of the input sentence, which is one of the characteristic properties of mild context-sensitivity. These two results long defined the view of CCG's place in the landscape of grammar formalisms. In this article, we contribute to a growing body of work that refines and partially changes this view. We briefly summarize the main results of this recent work before stating our own contributions.

1.1 Generative Power

One important insight in recent work on the generative power of CCG concerns the role of *lexicalization*. Modern CCG proposes a radically lexicalized theory of grammar in which all linguistic structure is projected from a language-specific lexicon by a small set of universal rules (Steedman and Baldridge 2011). We call this formalism **pure** CCG. By contrast, the formalism studied in the classical equivalence result of Weir and Joshi (1988) and Vijay-Shanker and Weir (1994) allows restricting the applicability of rules on a per-grammar basis. This makes a significant difference in terms of expressiveness: Kuhlmann, Koller, and Satta (2010, 2015) show that the class of string languages generated by pure CCG is not equivalent to but properly included in the class of languages generated by TAG. More specifically, they show that every language generated by a pure CCG contains a permutation-equivalent, context-free subset—a property not shared by the TAG languages. The intuition behind this formal result is that the universal availability of combinatory rules entails a certain loss of control in the derivation process. For example, every pure CCG that admits infinitely many strings of the (non-context-free) form $a^n b^n c^n$ will *necessarily* also admit permutations of the context-free form $(ab)^n c^n$ or $a^n (bc)^n$. This closure property has concrete implications in linguistic modeling. For instance, Kuhlmann, Koller, and Satta (2015) show that any pure CCG that models the canonical, cross-serial word order in Swiss German subordinate clauses (Shieber 1985) also predicts alternative, non-crossing word orders. For the more general class of *prefix-closed* CCGs, Kuhlmann, Koller, and Satta (2015) prove that it is the availability or unavailability of a specific type of rule restrictions called **target restrictions** that separates TAG-equivalent CCGs from CCGs with the permutation closure property. Related to this, the authors discuss the conditions under which modern *multimodal* variants of the CCG formalism—which are able to express certain but not all rule restrictions—can be weakly equivalent to TAG. We shall return to multimodal CCG in Section 6.3.

All known results on the weak generative power of CCG concern grammars that only use functional application and composition. It is generally accepted that *substitution rules*, which we will discuss in detail later on, do not change the generated string languages, although we are not aware of any full proof of this supposition. Steedman (2000, page 210) sketches how the simulation of CCG by Linear Indexed Grammar in the proof of Vijay-Shanker and Weir (1994) can be extended to cover substitution rules, which shows that adding these rules does not *increase* the generative power of CCG.

The impact of lexicalization on expressive power also extends to notions of strong generative capacity. In particular, CCG has been studied as a generator of tree languages via its **derivation trees**. The standard conception of a derivation tree in CCG, illustrated in Figure 1, is that of a binary tree whose leaves are labeled with lexical categories and whose inner nodes are labeled with derived categories produced by combinatory rules

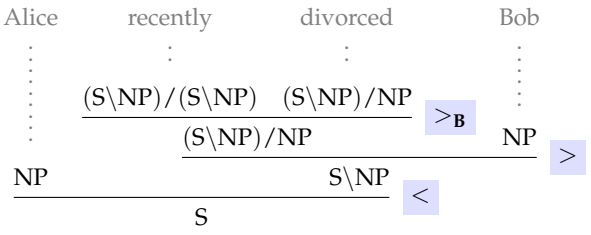


Figure 1
A CCG derivation tree for the sentence *Alice recently divorced Bob* (from Kuhlmann, Maletti, and Schiffer 2022). Rules are annotated with their conventional abbreviations (highlighted): $>$ forward application; $<$ backward application; $>_{\mathbf{B}}$ forward harmonic composition.

such as functional application and composition.[1] An alternative conception that is more convenient from a formal point of view is to let the derived categories remain implicit and consider derivation trees where the inner nodes are labeled with the names of the combinatory rules.[2] (In Figure 1, abbreviated rule names are highlighted.) Under this view, the set of derivation trees licensed by a given CCG forms a tree language in the technical sense of the term. Then, just as in the string case, the generative power of CCG depends on the availability of rule restrictions. The restricted formalism of Weir and Joshi (1988) and Vijay-Shanker and Weir (1994) generates the same tree languages as TAG (Schiffer and Maletti 2021). This holds even when lexical entries for the empty word—so-called ε-entries—are forbidden. That case is particularly interesting because such entries contradict one of the fundamental linguistic principles of CCG, the Principle of Adjacency (Steedman 2000, page 54). The same class of tree languages is generated by simple monadic context-free tree grammar, as proven by Kepser and Rogers (2011); an equivalent model is the linear top–down push-down tree automaton (Fujiyoshi and Kasai 2000). On the other hand, pure CCG cannot even generate all regular tree languages (i.e., the sets of trees generated by context-free grammars), and CCG that allows application rules and composition rules of degree 1 can generate exactly the regular tree languages (Kuhlmann, Maletti, and Schiffer 2019, 2022).

Table 1 summarizes several results on the string-generative and tree-generative power of CCG. Other work has studied the sets of **dependencies** that can be expressed by the formalism. Hockenmaier and Young (2008) compare a version of CCG that includes the type-raising rule with Lexicalized TAG and identify specific cases of scrambling that can only be expressed in the former but not in the latter. Another result is that of Koller and Kuhlmann (2009), who inspect the classes of dependency trees induced by CCG and TAG and show that these are incomparable. In a similar direction, Stanojević and Steedman (2021) study cross-linguistic word order variations in two major constructions with a "natural order of dominance" of their elements and show that CCG can characterize exactly the **separable permutations** of this order. Separable permutations are those for which one can construct a binary tree where the leaves of each subtree form a contiguous subset of the original order. By contrast, TAG can characterize even non-separable permutations. In particular, as Stanojević and Steedman (2021) point out, the example of Koller and Kuhlmann (2009, Figure 8b) of a dependency tree that can be

1 Later on, we will consider the input words as nodes of the derivation tree as well.
2 This conception of CCG derivation trees was considered already by Weir and Joshi (1988).

Table 1
Overview of results on the generative power of CCG, relating it to context-free languages (CFL), regular tree languages (RTL), and tree-adjoining languages (TAL). Unless stated otherwise, rule restrictions are allowed. References: (a) Bar-Hillel, Gaifman, and Shamir (1960); (b) Buszkowski (1988); (c) Weir and Joshi (1988); (d) Vijay-Shanker and Weir (1994); (e) Fowler and Penn (2010); Kuhlmann, Koller, and Satta (2010); (f) Kuhlmann, Koller, and Satta (2015); (g) Steedman (2000, page 210); (h) Schiffer and Maletti (2021); (i) Kuhlmann, Maletti, and Schiffer (2022). Bar-Hillel, Gaifman, and Shamir (1960) and Buszkowski (1988) studied the generative capacity of classical categorial grammar, which is pure, but the results for CCG with rule restrictions easily follow (see also Kuhlmann, Maletti, and Schiffer 2022). We have omitted further conclusions that can be drawn by combining the reported results.

CCG variant	ε-entries	string languages	tree languages
(pure) with application rules only	yes/no	$= \mathrm{CFL}^{(a)}$	$\subsetneq \mathrm{RTL}^{(b)}$
pure with composition degree $d = 1$	yes/no	$= \mathrm{CFL}^{(i)}$	$\subsetneq \mathrm{RTL}^{(i)}$
composition degree $d = 1$	yes/no	$= \mathrm{CFL}^{(e)}$	$= \mathrm{RTL}^{(i)}$
pure with composition degree $d \geq 2$	yes/no	$\subsetneq \mathrm{TAL}^{(f)}$	$\subsetneq \mathrm{TAL}^{(i)}$
prefix-closed without target restrictions	yes/no	$\subsetneq \mathrm{TAL}^{(f)}$	
prefix-closed with target restrictions	yes	$= \mathrm{TAL}^{(f)}$	
composition degree $d \geq 2$	no	$= \mathrm{TAL}^{(h)}$	$= \mathrm{TAL}^{(h)}$
composition degree $d \geq 2$	yes	$= \mathrm{TAL}^{(d)}$	
composition/substitution degree $d \geq 2$	yes	$= \mathrm{TAL}^{(g)}$	
generalized composition of unlimited degree	yes	$\supsetneq \mathrm{TAL}^{(c)}$	

expressed by TAG but not by CCG corresponds to a non-separable permutation. This marks yet another recent result on generative power that separates CCG from other mildly context-sensitive grammar formalisms.

1.2 Computational Power

In this article, we are primarily concerned with the computational power of CCG, and in particular with their parsing complexity. As already mentioned, polynomial-time parsing is one of the characteristic properties of mildly context-sensitive grammars as they were originally proposed by Joshi (1985). However, it is important to note that this property can refer to two different computational problems. The result by Vijay-Shanker and Weir (1993) mentioned above refers to the **membership problem**, where parsing complexity is measured relative to the length of the input string. Under this conceptualization, CCG and TAG are equal in their computational power.

Looking at the **universal recognition problem**, however, where runtime is measured as a function of both the length of the input string and the size of the grammar, the parsing complexities of CCG and TAG are much different: Whereas the universal recognition problem of TAG remains polynomial (Schabes 1990), Kuhlmann, Satta, and Jonsson (2018) show that for CCG, this problem is EXPTIME-complete. This means that, in the worst case, the runtime of any parsing algorithm for CCG will grow exponentially with the size of the grammar. However, the exact effect of different grammar-related features on parsing complexity have been hard to pin down so far. Kuhlmann and Satta (2014) present an analysis that refers to several rather specialized properties, including the maximum degree of a composition rule, the maximal arity of an argument in the lexicon, and the maximum number of arguments. Kuhlmann, Satta, and Jonsson (2018)

discuss additional features and their relevance for their EXPTIME result. As they point out, the availability of ε-entries is crucial: Without this feature, the universal recognition problem becomes NP-complete. The other features that they discuss are lexical ambiguity (the nondeterministic assignment of categories to input words via the lexicon), the use of restricted rules that may contain variables, and the absence of a fixed bound on the maximal degree of composition.[3] Dropping any of these features would break the EXPTIME proof, but it remained unclear whether it would also yield a formalism whose universal recognition problem is solvable in polynomial time.

1.3 Contributions of the Present Article

The present article takes a further step toward a better understanding of the complexity of CCG parsing. Our first contribution is to radically simplify earlier complexity analyses, and in particular that of Kuhlmann and Satta (2014): As we will show, the universal recognition problem is exponential *only* in the maximum rule degree; bounding this degree by a constant leads to a problem that is polynomial in the grammar size. Such a bound is interesting both from a theoretical and a practical point of view. From the equivalence proof of Vijay-Shanker and Weir (1994), it is clear that rule degree $d \leq 2$ is sufficient to obtain the full generative power in terms of string languages. More recently, Schiffer and Maletti (2021) showed that this result even holds when considering CCGs as generators of tree languages. In the linguistic work on CCG, a bounded degree of composition has repeatedly been discussed as a syntactic universal. In particular, for English, Steedman (2000, page 42) posits the bound $d \leq 3$.

Our second contribution is the extension of existing parsing algorithms to the full class of practically relevant CCGs, by including the substitution rule. This rule is used to model linguistic phenomena such as parasitic gaps, but was not part of the formalism considered by Vijay-Shanker and Weir (1994). Figure 2 shows an example from Steedman (2000, page 51) that requires backward crossed substitution to obtain a category for the phrase *file without reading*. Steedman argues that this expression should be considered a constituent because it can be used in coordinate structures. The gaps behind *without reading* and behind *file* refer to the same resource, expressed by arguments /NP, which collapse into a single argument /NP through substitution. Although it is accepted that substitution rules do not affect the weak generative power of CCG, their effect on the computational complexity of the parsing problem is much less clear. Therefore, the generalization of the parsing algorithm to substitution rules is an important contribution. To the best of our knowledge, our result is the first one that explicitly features substitution. As we will show, the maximum degree of substitution has a similar impact on the parsing complexity as the maximum degree of composition.

A full complexity analysis of practical CCG would cover not only application, composition, and substitution, but also *type-raising*. However, here we follow Steedman's assumption that this rule cannot be used recursively, and can therefore be compiled into the lexicon of the grammar (Steedman 2011, page 81). Similarly, we assume a purely

3 Each individual CCG defines a maximum degree for its composition rules, since its rule set is finite. However, the analysis by Kuhlmann, Satta, and Jonsson (2018) assumes that there is no *universal* bound on the maximum degree of composition rules across all possible CCGs. As a side remark, if composition rules are generalized in such a way that their degree is unbounded within an individual CCG, the resulting formalism becomes more powerful (Weir and Joshi 1988) and is actually Turing-complete when ε-entries are allowed (Kuhlmann, Satta, and Jonsson 2018).

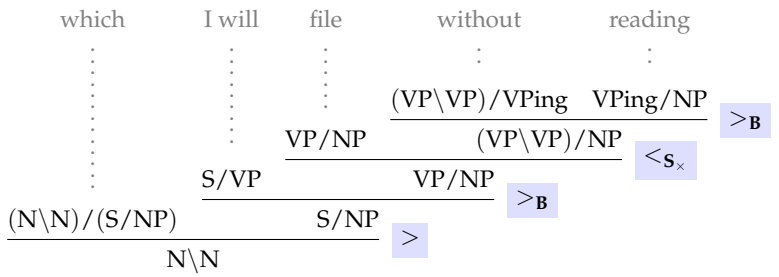


Figure 2
A CCG derivation tree (from Steedman 2000, page 51) that uses backward crossed substitution ($<\mathbf{S}_\times$) to combine the categories corresponding to *file* and *without reading*. Note that *I will* with associated category S/VP is not an actual lexicon entry but is obtained through prior combination of the individual words (abbreviated here).

lexical treatment of *coordination* via (appropriately restricted) lexicon entries such as *and* $:= X\backslash X/X$ (Steedman 2011, page 91).

1.4 Structure of the Article

The remainder of this article is structured as follows. Section 2 provides an introduction to the CCG formalism that we use. Section 3 is dedicated to the new parsing algorithm. After explaining the central idea of the *factorization* of derivations in an informal way, we define the basic notions required for our presentation, and then provide a formal specification of the algorithm and an illustrating example. The section concludes with a comprehensive runtime analysis. Section 4 contains the correctness proof of the algorithm, divided into the proofs of soundness and completeness. Section 5 describes how to construct a CCG derivation tree from the parse trees produced by our algorithm. Section 6 explores several potential extensions and improvements of the algorithm, including the elimination of spurious ambiguity, support for rule restrictions, support for multimodal CCG, and an algorithm for the universal recognition problem running in polynomial runtime if all secondary input categories in the rule set of the grammar are instantiated. Section 7 concludes the article.

2. Preliminaries

The CCG formalism that we consider in this article is based on that of Vijay-Shanker and Weir (1994). A grammar in this formalism consists of two parts: a *lexicon* that specifies syntactic information in the form of *categories*, and a set of *rules* that specify how to project the lexical information onto longer sentences. In the formalism of Vijay-Shanker and Weir (1994), these rules included rules of *application* and *composition*; here we also add rules of *substitution*.

2.1 Categories

Categories are syntactic types that can either be primitive, such as S ("sentence") or NP ("noun phrase"), or complex, such as $(\mathrm{S}\backslash\mathrm{NP})/\mathrm{NP}$ or $\mathrm{S}\backslash\mathrm{NP}$. The intended interpretation of a complex category of the general form X/Y or $X\backslash Y$ is that it takes an argument

of category Y and returns an object of category X. Thus, complex categories constitute function types. Formally, given an alphabet A of **atomic categories** and the set of **slashes** $D = \{/, \backslash\}$, the set of **categories** is inductively defined as the smallest set $\mathcal{C}(A)$ such that (1) $A \subseteq \mathcal{C}(A)$ and (2) $(X/Y) \in \mathcal{C}(A)$ and $(X\backslash Y) \in \mathcal{C}(A)$ for $X, Y \in \mathcal{C}(A)$. The slashes are left-associative by convention, so we can write each category as

$$c = a|_1 c_1 \cdots |_k c_k$$

where $k \geq 0$, $a \in A$, and $|_i \in D$, $c_i \in \mathcal{C}(A)$ for all $1 \leq i \leq k$. The atomic category a is called the **target** of c and denoted by $\text{target}(c)$. The slash–category pairs $|_i c_i$ are called **arguments** and their number k is called the **arity** of c and written as $\text{arity}(c)$.

Example 1
The category S\NP/NP is identical to (S\NP)/NP due to left-associativity of slashes. It has target S and two arguments \NP and /NP, thus its arity is 2. Note that this category is different from S\(NP/NP), which has arity 1.

2.2 Rules

Combinatory Rules. Categories can be combined using **combinatory rules**. We propose the following compact notation that covers both composition and substitution rules in a uniform manner. Each combinatory rule has one of the forms

$$X/Y\alpha \quad Y\alpha\beta \quad \Rightarrow \quad X\alpha\beta \qquad \text{(forward rule)}$$

$$Y\alpha\beta \quad X\backslash Y\alpha \quad \Rightarrow \quad X\alpha\beta \qquad \text{(backward rule)}$$

where α and β are sequences of slash–variable pairs such that $|\alpha| \leq 1$ and $|\beta| \geq 0$. The rules describe how a **primary input category** (highlighted) can be combined with a **secondary input category** to produce an **output category**. We shall refer to the slash–category pairs in $|Y\alpha$ as **bridging arguments** and to the sequence $\alpha\beta$ as the **excess**. The primary input category expects the bridging arguments to be provided by the secondary input category; the leading slash of the argument $|Y$ specifies the direction where the secondary input category is found. The output category follows the form of the primary input category with the bridging arguments replaced by the excess. We note that we can write down combinatory rules more explicitly as

$$X/Y\alpha \quad Y\alpha\,|_1 C_1 \cdots |_b C_b \quad \Rightarrow \quad X\alpha\,|_1 C_1 \cdots |_b C_b \qquad \text{(forward rule)}$$

$$Y\alpha\,|_1 C_1 \cdots |_b C_b \quad X\backslash Y\alpha \quad \Rightarrow \quad X\alpha\,|_1 C_1 \cdots |_b C_b \qquad \text{(backward rule)}$$

where $\alpha \in \{\varepsilon, |_0 C_0\}$, and for $i \in \{0, \ldots, b\}$, we have $|_i \in D$, and X, Y, and C_i are category variables that range over all categories in $\mathcal{C}(A)$. We can optionally restrict the ranges of some of the variables as described below.

Let $a = |\alpha|$ and $b = |\beta|$. Rules with $a = 0$ are **composition rules**, and rules with $a = 1$ are **substitution rules**. A special form of composition rules are **application rules**, which have $a = b = 0$. The length $d = a + b$ is called the **degree** of the rule. This means that substitution rules have degree at least 1 and application rules are composition rules of degree 0. Figure 3 lists the (unrestricted) combinatory rules of degree 0 and degree 1.

Application ($a = 0, b = 0$)

$$\begin{array}{llll} X/Y & Y & \Rightarrow & X \quad \text{(forward application)} \\ Y & X\backslash Y & \Rightarrow & X \quad \text{(backward application)} \end{array}$$

Composition of degree 1 ($a = 0, b = 1$)

$$\begin{array}{llll} X/Y & Y/Z & \Rightarrow & X/Z \quad \text{(forward harmonic composition)} \\ X/Y & Y\backslash Z & \Rightarrow & X\backslash Z \quad \text{(forward crossed composition)} \\ Y\backslash Z & X\backslash Y & \Rightarrow & X\backslash Z \quad \text{(backward harmonic composition)} \\ Y/Z & X\backslash Y & \Rightarrow & X/Z \quad \text{(backward crossed composition)} \end{array}$$

Substitution of degree 1 ($a = 1, b = 0$)

$$\begin{array}{llll} X/Y/Z & Y/Z & \Rightarrow & X/Z \quad \text{(forward harmonic substitution)} \\ X/Y\backslash Z & Y\backslash Z & \Rightarrow & X\backslash Z \quad \text{(forward crossed substitution)} \\ Y\backslash Z & X\backslash Y\backslash Z & \Rightarrow & X\backslash Z \quad \text{(backward harmonic substitution)} \\ Y/Z & X\backslash Y/Z & \Rightarrow & X/Z \quad \text{(backward crossed substitution)} \end{array}$$

Figure 3
Combinatory rules of degree 0 and degree 1.

Rule Restrictions. There are two types of **rule restrictions**. A **target restriction** can restrict the target of the variable X. More specifically, it can restrict X to range over a set of the form $\{c \in \mathcal{C}(A) \mid \text{target}(c) \in T\}$ with $T \subseteq A$. A **secondary input restriction** can restrict the variable Y and any of the variables C_i with $i \in \{0, \ldots, b\}$ to a subset of $\mathcal{C}(A)$.

Instantiation of Rules. Before combinatory rules can be applied, they first have to be **instantiated** by replacing the variables X, Y, and the variables C_i for $i \in \{0, \ldots, b\}$ by concrete categories from $\mathcal{C}(A)$, yielding a **ground instance** of the combinatory rule. We use the compact notation of combinatory rules also for ground instances, so that α and β denote actual argument sequences. It will always be clear from the context whether α and β are sequences of slash–variable pairs or else sequences of actual arguments.

We would like to point out that for a fixed CCG there is only a finite set of categories that the variables Y and C_i for $i \in \{0, \ldots, b\}$ can be instantiated with. This is because all arguments of categories occurring in derivations, and therefore all arguments occurring in the applied ground instances, already appear in the lexicon (Vijay-Shanker and Weir 1994, Lemma 3.1). However, even if each of these variables is restricted to match only a single category, a combinatory rule can still have infinitely many ground instances due to X ranging over infinitely many categories.

Example 2
We consider the forward harmonic composition rule $X/Y, Y/Z \Rightarrow X/Z$, where $a = 0$, $b = 1$, and $\beta = /Z$.[4] We can either leave it unrestricted, such that X, Y, and Z can be instantiated with any category, or we can use rule restrictions. For instance, we can restrict Z to $\{\text{NP}\}$ using a secondary input restriction and we can restrict X to take a target from $\{S\}$ using a target restriction. In this scenario, $(\text{S}\backslash\text{NP})/(\text{S}\backslash\text{NP}), (\text{S}\backslash\text{NP})/\text{NP} \Rightarrow (\text{S}\backslash\text{NP})/\text{NP}$ with bridging argument $/(\text{S}\backslash\text{NP})$ and excess $/\text{NP}$ is a valid ground instance, whereas $\text{NP}/\text{NP}, \text{NP}/\text{NP} \Rightarrow \text{NP}/\text{NP}$ is not, as it violates the target restriction.

2.3 Grammars

A CCG is defined as a tuple $G = (\Sigma, A, :=, R, S)$. It consists of

1. the finite vocabulary Σ, from which the input words are taken,
2. the atomic categories A,
3. the **lexicon**, which is a finite relation $:=$ between $\Sigma \cup \{\varepsilon\}$ and $\mathcal{C}(A)$,
4. a finite set R of combinatory rules, and
5. the **distinguished category** $S \in A$.

Note that the lexicon can assign categories not only to the possible input words from Σ, but also to the empty word ε. These lexicon entries are called **ε-entries**. A category X that occurs in some lexicon entry $\sigma := X$ is called **lexical category**.

Pure CCG. A CCG that allows all combinatory rules without rule restrictions, up to a fixed degree, is called **pure**. Note how "non-pure" is different from "no rule restrictions": For example, a CCG that uses only forward rules up to some degree where the ranges of variables are not restricted is not pure although it has no rule restrictions. In the present article, we will assume that the CCG we are given is pure. An extension of our algorithm to CCG that uses only a subset of combinatory rules up to some degree and specifically allows rule restrictions is discussed in Section 6.2.

Derivation. We start with an informal description of a derivation. Let $G = (\Sigma, A, :=, R, S)$ be a CCG and let $w = w_1 \cdots w_n$ be an input string, where $w_i \in \Sigma$ for $i \in \{1, \ldots, n\}$. First, the lexicon $:=$ assigns categories c_i to the input words such that $w_i := c_i$. This assignment is nondeterministic, since several categories might be associated with a single input word via the lexicon. Optionally, we may use ε-entries. In that case we start with a sequence of lexical categories that is longer than the input string. The combinatory rules of R successively combine neighboring categories: If two neighboring categories are the primary and secondary input category, respectively, of a ground instance of some combinatory rule in R, the associated output category can be derived. If through iteration of this process the distinguished category S is derived and under the condition that the complete category sequence corresponding to the input is incorporated, we say that the input string is generated by G.

4 In running text, combinatory rules are displayed with the input categories separated by a comma.

Derivation Tree. Following most of the CCG literature, we view CCG derivations as **trees**. In accordance with standard notation for trees, given some alphabet Δ, we write α for the tree consisting of only one (leaf) node with label $\alpha \in \Delta$, and we write $\sigma(t_1, \ldots, t_n)$ to denote the tree with root label $\sigma \in \Delta$ and subtrees $t_1, \ldots, t_n$ as children of the root. Given a tree t, we write $\text{root}(t)$ to access its root label. We define the **arity** of a node u as the number of children of u. Finally, we define the **size** of a tree t as the number of nodes in t.

Given a CCG $G = (\Sigma, A, :=, R, S)$, a **derivation tree** is a tree consisting of nodes with arity at most 2 such that internal nodes are labeled by categories $\mathcal{C}(A)$, leaves are labeled by elements from $\Sigma \cup \{\varepsilon\}$, and the following conditions are fulfilled. First, unary nodes are labeled by lexical categories that, together with their only child (which is labeled by an input word or by ε), constitute a lexicon entry. Second, binary nodes together with their children constitute valid ground instances of combinatory rules in R. Formally, the set of derivation trees is the smallest set $\mathcal{D}(G)$ such that (1) for all $\sigma := X$, we have $X(\sigma) \in \mathcal{D}(G)$, and (2) for all $t_1, t_2 \in \mathcal{D}(G)$ and ground instances $\text{root}(t_1), \text{root}(t_2) \Rightarrow X$ of combinatory rules in R, we have $X(t_1, t_2) \in \mathcal{D}(G)$.

For internal nodes u, we write $\text{cat}(u)$ to access the category that labels u. For leaf nodes u, we write $\text{input}(u)$ to access the input word or the symbol ε that labels u.

Following standard conventions for CCG, we draw derivation trees with the root at the bottom. If the word–category mapping specified by the lexicon is indicated, we visualize it using dotted lines (see Figure 1). However, when drawing derivation trees we will usually omit lexical entries.

Yield and Generated Language. The **yield** of a derivation tree is the sequence of input words associated with its leaves, read from left to right. It is recursively defined as follows. If t consists of a single node u, let $\text{yield}(t) = \text{input}(u)$. If $t = X(t')$, let $\text{yield}(t) = \text{yield}(t')$. If $t = X(t_1, t_2)$, let $\text{yield}(t) = \text{yield}(t_1) \cdot \text{yield}(t_2)$, where $\cdot$ denotes string concatenation.

The **language generated by** G is the set of strings that are the yield of some derivation tree whose root is labeled by the distinguished category S. Formally,

$$L(G) = \{\text{yield}(t) \mid t \in \mathcal{D}(G), \text{root}(t) = S\}\,.$$

Example 3
Figure 1 shows a derivation tree using forward application (denoted by $>$), backward application (denoted by $<$), and forward composition (denoted by $>_{\mathbf{B}}$). The composition rule combines the primary input category $(\text{S}\backslash\text{NP})/(\text{S}\backslash\text{NP})$, which is associated with the lexical item *recently*, and the secondary input category $(\text{S}\backslash\text{NP})/\text{NP}$, which is associated with *divorced*. We can observe that the bridging argument $/(\text{S}\backslash\text{NP})$ of the primary input category is replaced by the excess $/\text{NP}$, resulting in output category $(\text{S}\backslash\text{NP})/\text{NP}$. This operation is similar to function composition. The subsequent application rules incorporate the remaining input words and remove the two arguments from $(\text{S}\backslash\text{NP})/\text{NP}$, resulting in distinguished category S.

3. Parsing Algorithm

In this section we develop a tabular algorithm for parsing based on CCGs. Our algorithm extends the approach of Kuhlmann and Satta (2014) by including substitution rules. In spite of its extended power, we will see that the new algorithm facilitates a sharper analysis of its runtime complexity with respect to the grammar size.

3.1 Basic Idea

Before moving on with the technical presentation, we informally discuss here the key ideas at the core of our result.

Let w be some input string of length $|w|$. Recall that a CCG consists of a finite number of lexical categories, and hence a finite number of arguments. However, in a CCG derivation of w, the arity of produced categories can grow with $|w|$. This means that a naive tabular method for CCG parsing, which records in its parsing table each node of each possible derivation for w, may result in exponential time complexity in $|w|$, because of the combinatorial explosion of CCG categories. This has been already pointed out in the literature, for instance by Kuhlmann and Satta (2014, §3), who avoid this combinatorial explosion by resorting to factorization techniques for CCG derivations. Informally, each CCG derivation is broken into pieces such that each piece uses at most g of the topmost arguments in its categories, where g is a grammar constant that does not depend on $|w|$. The arguments that are not used by a derivation piece are not stored in the parsing table, so that the abovementioned combinatorial explosion of CCG categories only involves argument sequences of length at most g. This results in polynomial time (and space) parsing in $|w|$. In this article we generalize these techniques in order to factorize CCG derivations that also include substitution rules, which were not considered by Kuhlmann and Satta (2014).

As a second technical point, consider an instance of the composition rule having the form $X|Y, Y\alpha \Rightarrow X\alpha$, where α is some sequence of arguments. Because of the factorization of CCG derivations that we have mentioned above, the number of arguments in α is bounded by our constant g. However, the arity of category Y is bounded by the maximum arity g' of an argument from the lexicon, which is independent of g. In pathological cases, where g' is much larger than g, we again face the problem of combinatorial explosion of CCG categories. This problem is not dealt with well in the parsing algorithm of Kuhlmann and Satta (2014), which exhaustively produces all possible categories $Y\alpha$ with arity bounded by $g' + g$. As a solution for this we observe here that the category Y must match a finite number of possible arguments from the lexicon of the CCG. Using this restriction, we avoid the additional exponential factor of g' in the running time of our parsing algorithm.

3.2 Definitions and Notation

We start with some auxiliary definitions and notation that we use in the development of our algorithm.

String. We are given a CCG G and a string w to be parsed. We write $w[i, j]$ to denote the substring of w from (fencepost) position i to position j, for $0 \leq i \leq j \leq |w|$, and assume $w[i, i] = \varepsilon$. We also write w_i for $w[i-1, i]$, the one-element substring containing the i-th word in w.

Lexical Arguments. We write Args for the set of all arguments in the lexical categories of G. As already noted in Section 2.2, successful derivations can only contain categories consisting of lexical arguments (Vijay-Shanker and Weir 1994, Lemma 3.1). We write Args^* for the set of all sequences of lexical arguments. Further, for $d \geq 0$, we write $\text{Args}^{\leq d}$ for the sequences of at most d lexical arguments. These sequences

may also have length zero. As usual for strings, the empty sequence of arguments is denoted by ε.

Spine. The **spine** of a derivation tree t is the path that starts at the root node of t and at each node continues to that node's primary child (i.e., the child labeled by the primary input category) until it reaches some unary (pre-leaf) node in t. This node is called **lexical anchor**. We use the term **spinal node** to refer to any node on some (given) spine. Spinal nodes are clearly labeled by CCG categories, since the definition of spine excludes leaf nodes, which are labeled by a lexical item or else by the symbol ε. The **length** of a spine is defined as the number of its spinal nodes.

Node Arity. We extend the notion of arity to nodes labeled by categories. If u is a node of t, we write $\text{arity}(u)$ to denote the arity of the category labeling u.

We can consider a combinatory rule application as consisting of two phases, where in the first phase the bridging arguments of the primary input category are removed, and in the second phase the excess is added. Given a node u in t labeled by a primary input category X, the **downstep arity** of u is the arity of the category that is obtained from X by removing the bridging arguments, and is written $\text{downarity}(u)$.

Example 4
Consider node u with $\text{cat}(u) = A/B/C$ and parent node u' with $\text{cat}(u') = A/C\backslash D$, which is obtained by using an instance of the substitution rule $A\alpha/B/C, B/C\backslash D \Rightarrow A\alpha/C\backslash D$ with $\alpha = \varepsilon$. Then we have $\text{arity}(u) = \text{arity}(u') = 2$ and $\text{downarity}(u) = 0$.

3.2.1 Context. A **derivation context** or simply **context** is obtained from a derivation tree t with root node r by removing all proper descendants of some given spinal node $f \neq r$ under the following restrictions, where u is any node on the spine of t properly between f and r:

$$\text{downarity}(f) \leq \text{downarity}(u) \tag{1}$$

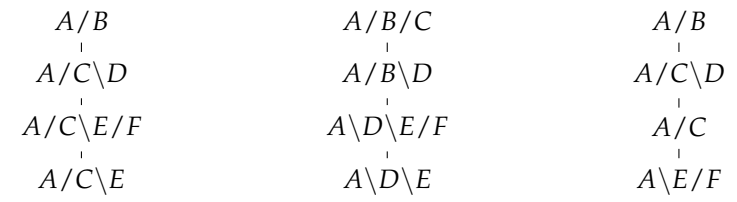
$$\text{arity}(r) \leq \text{arity}(u) \tag{2}$$

The node f is called the **foot node** of the context.

Example 5
Figure 4 shows some examples of spines in order to illustrate the context definition. For the sake of simplicity, the respective secondary input categories are omitted. According to our convention, the foot node is drawn at the top and the root node at the bottom.

Figure 4a depicts a valid context. In Figure 4b, the node labeled by $A/B\backslash D$ has downstep arity 0 due to substitution, but the downstep arity at the foot is 1. Therefore, the spine violates condition (1) of the context definition. Finally, in Figure 4c the node labeled by A/C has lower arity than root category $A\backslash E/F$. Therefore, the spine violates condition (2) of the context definition.

In what follows, α and γ denote sequences from Args^*. Let us write the category at f in the form $X\alpha$, where $\text{arity}(X) = \text{downarity}(f)$. This means that $1 \leq |\alpha| \leq 2$. Condition (1) implies that the category at each intermediate node u, as well as the category at node r, can be written as $X\gamma$. Note that only the arguments in γ may affect the derivation along the spine of the context. In other words, the arguments in X are not needed and we can represent the context without any record of X itself. We exploit



(a) valid context (b) violates Equation (1) (c) violates Equation (2)

Figure 4
Examples of spines complying with or violating the context definition.

this property later, to develop a dynamic programming algorithm where each context is stored in a compact form, and is shared among several CCG derivations.

Condition (2) is also very important for storing contexts in a compact form, and is at the basis of the proof of the following lemma, where r and f are defined as above.

Lemma 1
Let c be a context and let $\text{cat}(r) = X\beta$ with $\text{arity}(X) = \text{downarity}(f)$. Assume that the combinatory rules applied along the spine of c have degree at most d. Then $|\beta| \leq d$.

Proof. Let $\text{cat}(f) = X\alpha$ and let p be the parent node of f, with $\text{cat}(p) = X\gamma$. Because $\text{cat}(p)$ is obtained by applying a rule of degree at most d on primary input category $X\alpha$, we have $|\gamma| \leq d$. If $r = p$, we immediately have $\beta = \gamma$ and thus $|\beta| \leq d$. Otherwise, p is a node properly between f and r, and by condition (2) in the definition of context we have $\text{arity}(r) \leq \text{arity}(p)$. Observe that $\text{arity}(r) = \text{arity}(X) + |\beta|$ and $\text{arity}(p) = \text{arity}(X) + |\gamma|$. We can then write $|\beta| \leq |\gamma| \leq d$. □

Let α, β be defined as in the above proof. Extending our terminology from rules, we refer to α as the **bridging arguments** of the context and we refer to β as the **excess** of the context. We observe that, if a composition rule is used at the foot node, we have $|\alpha| = 1$, and if a substitution rule is used at the foot node, we have $|\alpha| = 2$.

3.2.2 Root Categories. To limit computational complexity, we introduce the set $\mathcal{G}$ of root categories of derivation trees that can be directly represented by our parsing algorithm. Derivation trees with root categories not in $\mathcal{G}$ will instead be represented in a factorized form. We define $\mathcal{G} = \mathcal{G}_1 \cup \mathcal{G}_2$, where $\mathcal{G}_1, \mathcal{G}_2$ are two not necessarily disjoint sets of categories specified as follows.

- $\mathcal{G}_1$ contains all categories $X\alpha$ such that X is some prefix (not necessarily proper) of a lexical category $X\beta$ from G, $\alpha \in \text{Args}^{\leq 2}$, and $\text{arity}(X\alpha) \leq \text{arity}(X\beta)$.
- $\mathcal{G}_2$ contains all categories $X\alpha$ such that X is some prefix (not necessarily proper) of an *instantiation* $X\beta$ of some secondary input category in a rule of G, $\alpha \in \text{Args}^{\leq 2}$, and $\text{arity}(X\alpha) \leq \text{arity}(X\beta)$.

Categories in $\mathcal{G}_1$ are used by the parser when derivation trees are introduced from lexical categories of G, and when topmost arguments of these categories are

"consumed" in a CCG derivation. Categories in $\mathcal{G}_2$ are instead used in the process of producing derivation trees that will serve as secondary input category in a CCG derivation. The α component of $X\alpha$ represents the bridging arguments. Later on, in the completeness proof of our parsing algorithm, it will become clear that this set does indeed suffice to represent all possible derivations.

3.3 Algorithm Specification

As usual in the natural language parsing literature, we formally specify our algorithm as a deduction system in the sense of Shieber, Schabes, and Pereira (1995).

3.3.1 Items. We use a logic with two types of items. Derivation trees whose root is labeled by a category in $\mathcal{G}$ can be represented by tree items directly. The additional context items follow our definition of context and can represent parts of derivation trees where arities of categories grow too large.

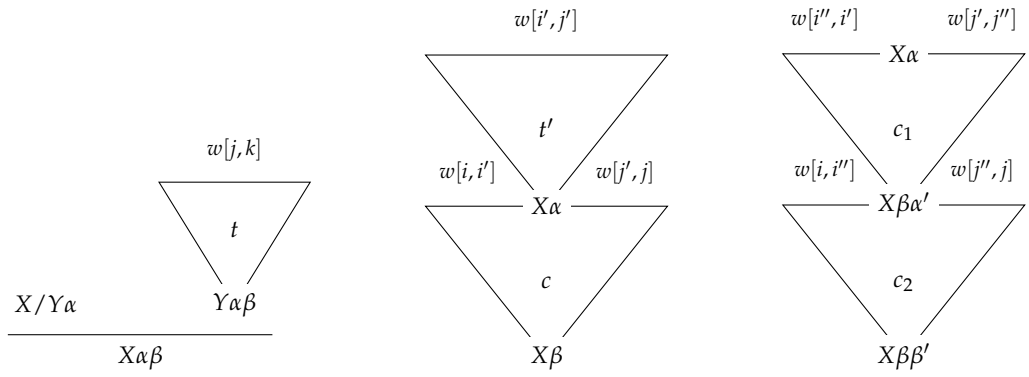
Tree Items. These have the form $[X, i, j]$, where $X \in \mathcal{G}$ and $0 \leq i \leq j \leq |w|$. The intended interpretation of such an item is: It is possible to build a derivation tree with yield $w[i, j]$ and root category X. The goal of the algorithm is the construction of an item of the form $[S, 0, |w|]$, which asserts the existence of a derivation tree that spans the entire input string and whose root node is labeled with S, where S is the distinguished category for sentences of G.

Context Items. These have the form $[\alpha, \beta, i, i', j', j]$, where $\alpha, \beta \in \text{Args}^{\leq d}$ with $1 \leq |\alpha| \leq 2$ and $0 \leq i \leq i' \leq j' \leq j \leq |w|$. The intended interpretation of these items is: For any choice of a category X, *if* it is possible to build a derivation tree t' with yield $w[i', j']$ and whose root node is labeled with $X\alpha$, *then* it is also possible to build a derivation tree t with yield $w[i, j]$ and whose root node is labeled with $X\beta$. In line with the usage of these terms for contexts, we refer to α as **bridging arguments** and to β as **excess**.

3.3.2 Axioms and Inference Rules. The inference rules and axioms of the algorithm can be classified along two dimensions, depending on whether the consequent item is a tree item or else a context item, and depending on whether the consequent item is obtained as the extension of an existing item of the same type or else it is newly introduced. Most importantly, in the following deduction system we implicitly assume that the inference rules are valid only if all of the involved items comply with the conditions in the definition of items provided above.

Introduce Derivation Tree. These are the axioms of the deduction system. For every word position $1 \leq i \leq |w|$ and every lexicon entry $w_i := X$, there is an axiom $[X, i-1, i]$. For every lexicon entry $\varepsilon := X$ and every fencepost position $0 \leq i \leq n$, there is an axiom $[X, i, i]$.

$$\frac{w_i := X}{[X, i-1, i]} \qquad \frac{\varepsilon := X}{[X, i, i]} \qquad \text{(rule 0)}$$



(a) context c (b) derivation tree t (c) context c

Figure 5
Decomposition of derivations. X and Y are categories; α, β, α', β' are (possibly empty) sequences of arguments satisfying the restrictions specified in the inference rules.

Introduce Context. For all valid items of the following form there are rules

$$\frac{[Y\alpha\beta, j, k]}{[/Y\alpha, \alpha\beta, i, i, j, k]} \quad \text{and} \quad \frac{[Y\alpha\beta, j, k]}{[\backslash Y\alpha, \alpha\beta, j, k, l, l]} \qquad \text{(rule 1)}$$

This rule type converts a tree item into a context item that models the effect of $Y\alpha\beta$ when applied as a secondary input category. The context has a spine of length 2 and is depicted in Figure 5a.

Extend Derivation Tree. For all valid items of the following form there is a rule

$$\frac{[X\alpha, i', j'][\alpha, \beta, i, i', j', j]}{[X\beta, i, j]} \qquad \text{(rule 2)}$$

This rule type models the combination of a derivation tree and a context, such that the derivation tree rooted in $X\alpha$ is inserted at the foot node of the context, resulting in a derivation tree rooted in $X\beta$. This is depicted in Figure 5b.

Extend Context. For all combinations of valid context items of the following form with $|\beta'| \leq |\alpha'|$, there is a rule

$$\frac{[\alpha, \beta\alpha', i'', i', j', j''][\alpha', \beta', i, i'', j'', j]}{[\alpha, \beta\beta', i, i', j', j]} \qquad \text{(rule 3)}$$

This rule type models the combination of two contexts c_1 and c_2, represented by the left and right antecedent items, respectively. More precisely, c_1 is inserted at the foot node of c_2, resulting in a new context c represented by the consequent item. This is exemplified in Figure 5c.

The use of restriction $|\beta'| \leq |\alpha'|$ in rule 3 deserves some discussion here. This restriction guarantees that c fulfills condition (2) in the definition of context. Without this restriction, there might be nodes on the spine that have lower arity than the root. As a second observation, consider rule 3 as a means of extending context c_1 by "transferring" to this context the arguments from β'. Under this view, restriction $|\beta'| \leq |\alpha'|$ forbids such transferring whenever this results in an increase in the arity at the root of c, as compared with the arity at the root of c_1. One might then wonder whether forbidding the transferring of extra arguments from context to context might result in the loss of some valid derivations. As we will see in the completeness proof in Section 4, this strategy is safe, since we can always transfer extra arguments directly to some tree item later, by means of rules of type 2, rather than passing them through several intermediate contexts.

Example 6
As an example we consider a CCG with all composition and substitution rules of degree at most 2 and the input string $w_1 \cdots w_7$. The lexicon is specified as follows:

$$w_1 := A/E \qquad w_2 := C/E/F \qquad w_3 := S\backslash A/B \qquad w_4 := B\backslash C/E$$
$$w_5 := F \qquad w_6 := E/G \qquad w_7 := G$$

Figure 6 depicts a derivation of the CCG that shows that the input is generated by the grammar. We can observe that on the spine between the root and the lexical category $S\backslash A/B$ there are two categories that are not in $\mathcal{G}$ because they violate the arity restriction. Thus, they cannot be represented using tree items.

Figure 7 shows how the deduction system operates on the input. For each input symbol and matching lexicon entry, an axiom with the corresponding span is added by rule 0. These are the leaves of the deduction. Note that their order does not coincide with the order of the associated lexicon entries in the input string. To simulate the use of a combinatory rule, a secondary input category first has to be converted into a context item using deduction rule 1. For instance, in order to use it as a secondary input, tree item $[B\backslash C/E, 3, 4]$ is converted into context item $[/B, \backslash C/E, 2, 2, 3, 4]$, which describes its effect on a primary input category. Note that there is some freedom here concerning the indices, because there might be several choices regarding how this item wraps around other items. However, the deduction system does not allow the combination of items $[S\backslash A/B, 2, 3]$ and $[/B, \backslash C/E, 2, 2, 3, 4]$ because the corresponding output category

$$\dfrac{A/E \qquad \dfrac{\dfrac{C/E/F \qquad \dfrac{S\backslash A/B \quad B\backslash C/E}{S\backslash A\backslash C/E}\,>^2_{\mathbf{B}}}{S\backslash A/E/F}\,<^2_{\mathbf{S}} \qquad F}{S\backslash A/E}\,> \qquad \dfrac{E/G \quad G}{E}\,>}{S/E}\,<_{\mathbf{S}}}{S}\,>$$

Figure 6
CCG derivation with spine nodes in blue text. Categories in the spine that are not in $\mathcal{G}$ are highlighted with blue background.

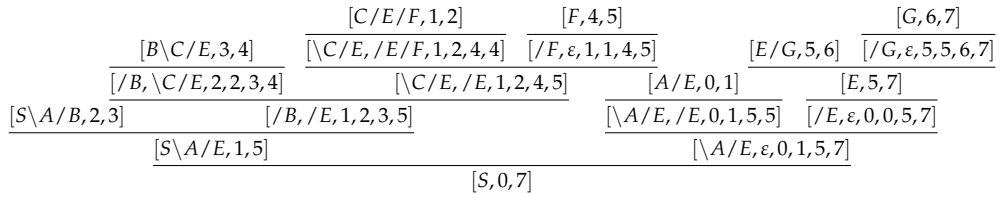


Figure 7
Example deduction.

is not in $\mathcal{G}$. Instead, the excess of the context item is reduced by combining it with another context item using deduction rule 3 first, resulting in $[/B, /E, 1, 2, 3, 5]$. This item models the effect that the categories of the three involved axioms have when applied successively to a category ending in $/B$. These are exactly the first three secondary input categories that are applied along the considered spine. The item covers the spans $[1, 2]$ and $[3, 5]$ of the input and has a gap at $[2, 3]$, such that $[S\backslash A/B, 2, 3]$ can be inserted using deduction rule 2, resulting in $[S\backslash A/E, 1, 5]$. The interpretation of this item is as follows: There exists a CCG derivation tree with root category $S\backslash A/E \in \mathcal{G}$ that involves the input symbols of the span $[1, 5]$. This can be verified in Figure 6. Note that it is also possible to change the combination order of context items and to first combine $[/B, \backslash C/E, 2, 2, 3, 4]$ with $[\backslash C/E, /E/F, 1, 2, 4, 4]$ and then to combine the resulting item with $[F, \varepsilon, 1, 1, 4, 5]$.

The spinal categories that are in $\mathcal{G}$ can be represented by tree items, but the categories in between two of these relatively short categories might have higher arity. In that case the operations taking place along that part of the spine are handled on the level of context items and their effect can only be added to the tree items if the excess is short enough. Not only each lexical category, but also each secondary input category (and the distinguished category) is represented by some tree item. We can uniquely decompose each CCG derivation tree into a set of maximal spines, meaning that each of these is the spine of a subtree and not contained in the spine of any larger subtree of the original tree. At least the lexical anchor and the root of each of these spines are represented by tree items. For example, category E as a secondary input category is the root of a maximal spine consisting of two nodes and accordingly represented by tree item $[E, 5, 6]$. Note that for trivial spines consisting of a single node, the lexical anchor and the root coincide.

3.4 Runtime Analysis

In this section we provide a computational analysis of our parsing algorithm. We consider the time and space complexity attributed to the execution of each of the deduction rule types. We also discuss the control flow for the deduction system, along with possible representations for rules and items.

We write $|w|$ for the length of the input string w. We also write $|G|$ for the size of the input grammar G, defined as the number of characters that we need to write down the lexicon and all the rules in some reasonable representation. Finally, we write d to denote the maximum degree of composition and substitution rules in G.

3.4.1 Grammar. We start our analysis by deriving bounds for the size of sets $\text{Args}^{\leq d}$ and $\mathcal{G}$, to be used later. Recall that Args is the set of all arguments in the lexical categories of G. Because every argument appears in our string representation of G, we have $|\text{Args}| \in$

$\mathcal{O}(|G|)$. Using the definition of $\text{Args}^{\leq d}$ we obtain $|\text{Args}^{\leq d}| \leq \sum_{i=0}^{d} |G|^i$. The right-hand side is the sum of the first $d+1$ terms of a geometric series. Using the closed-form formula for this sum we can write

$$\sum_{i=0}^{d} |G|^i = \frac{|G|^{d+1}-1}{|G|-1} < \frac{|G|^{d+1}}{|G|-1} = \frac{|G|}{|G|-1} \cdot |G|^d \leq 2 \cdot |G|^d$$

which holds for $|G| \geq 2$ and $d \geq 2$. We thus conclude that $|\text{Args}^{\leq d}| \in \mathcal{O}(|G|^d)$.

As for set $\mathcal{G}$, we separately analyze the two subsets $\mathcal{G}_1$ and $\mathcal{G}_2$, according to the definition in Section 3.2. Consider a category $X\alpha \in \mathcal{G}_1$, where X is a prefix of a lexical category and $\alpha \in \text{Args}^{\leq 2}$. Since every lexical category appears in our string representation of G, each prefix of a lexical category can be associated with some position within that string, representing the end position of the prefix. (The start position is uniquely determined, given the end position.) Therefore the number of prefixes of lexical categories does not exceed $|G|$. Because $|\text{Args}^{\leq 2}| \in \mathcal{O}(|G|^2)$, we conclude that $|\mathcal{G}_1| \in \mathcal{O}(|G|^3)$.

Consider now set $\mathcal{G}_2$, whose definition is based on the notion of instantiation of secondary input categories in rules of G. Recall that according to our convention, and ignoring the ordering of the antecedents, the general form of an instantiated rule of G is $Y|Z\alpha', Z\alpha'\beta' \Rightarrow Y\alpha'\beta'$, where $Z\alpha'\beta'$ is the instantiated secondary input category, $|Z \in \text{Args}$, $\alpha' \in \text{Args}^{\leq 1}$, and $\alpha'\beta' \in \text{Args}^{\leq d}$.

Let $X\alpha \in \mathcal{G}_2$, where X is a prefix, not necessarily proper, of some instantiated secondary input category $Z\alpha'\beta'$, and $\alpha \in \text{Args}^{\leq 2}$. We distinguish two cases on the basis of the arity of X.

- $\text{arity}(X) \leq \text{arity}(Z)$: In this case X is a prefix, not necessarily proper, of Z. Every category Y such that $|Y \in \text{Args}$ must appear in our string representation of G, and thus each prefix of such category can be associated with some position of the string. Therefore the number of prefixes of these categories does not exceed $|G|$. Because $\alpha \in \text{Args}^{\leq 2}$, we conclude that the number of $X\alpha \in \mathcal{G}_2$ such that $\text{arity}(X) \leq \text{arity}(Z)$ is in $\mathcal{O}(|G|^3)$.
- $\text{arity}(X) > \text{arity}(Z)$: In this case prefix X of $Z\alpha'\beta'$ spans over some of the arguments in $\alpha'\beta'$. We can then write X in the form $Z\gamma$ for some non-empty sequence of arguments γ. Let us associate $Z\alpha'\beta'$ with a sequence of arguments $|(Z)\alpha'\beta'$; similarly, we associate X with a sequence of arguments $|(Z)\gamma$. Since $|Z \in \text{Args}$ and $\alpha'\beta' \in \text{Args}^{\leq d}$, we have $|(Z)\alpha'\beta' \in \text{Args}^{\leq d+1}$. Using the condition $\text{arity}(X\alpha) \leq \text{arity}(Z\alpha'\beta')$ in the definition of $\mathcal{G}_2$, we also derive $|\gamma\alpha| \leq |\alpha'\beta'|$, and thus $|(Z)\gamma\alpha \in \text{Args}^{\leq d+1}$. We therefore conclude that the number of $X\alpha \in \mathcal{G}_2$ such that $\text{arity}(X) > \text{arity}(Z)$ is in $\mathcal{O}(|G|^{d+1})$.[5]

Putting everything together we conclude that, for $d \geq 2$, we have $|\mathcal{G}| \in \mathcal{O}(|G|^{d+1})$.

5 The attentive reader may observe that we are overcounting the number of instantiations of secondary input categories. For instance, consider rule instantiations $D|(A|B), A|B|C \Rightarrow D|C$ and $D|A, A|B|C \Rightarrow D|B|C$, sharing the same secondary input category $A|B|C$. In the first rule, we associate $A|B|C$ with sequence $|(A|B)|C$, while in the second rule we associate the same category with sequence $|(A)|B|C$, counting the same secondary input category twice. Of course, this is not a problem for the construction of an upper bound.

3.4.2 Items. We derive here upper bounds for the total number of items that are produced in a run of our algorithm on string w. Since each tree item $[X, i, j]$ satisfies $X \in \mathcal{G}$, the total number of tree items must be in $\mathcal{O}(|G|^{d+1} \cdot |w|^2)$. Consider now a context item $[\alpha, \beta, i, i', j', j]$. Because $1 \leq |\alpha| \leq 2$ and $\beta \in \text{Args}^{\leq d}$, the total number of context items is in $\mathcal{O}(|G|^{d+2} \cdot |w|^4)$.

For future use, we also develop bounds on the number of arguments appearing in tree and context items. We have already observed above that, for a context item $[\alpha, \beta, i, i', j', j]$, we have $|\alpha\beta| \leq d + 2$. Regarding tree items, we need to introduce some auxiliary notation. Let ℓ be the maximum arity of a lexical category in G. Additionally, let r be the maximum arity of a category Z for all possible $|Z \in \text{Args}$. Consider a tree item $[X, i, j]$. According to the definition of $\mathcal{G}$, if $X \in \mathcal{G}_1$, then $\text{arity}(X) \leq \text{arity}(W)$ for some lexical category W. If $X \in \mathcal{G}_2$, then $\text{arity}(X) \leq \text{arity}(Z\alpha\beta)$ for some $|Z \in \text{Args}$ and $\alpha\beta \in \text{Args}^{\leq d}$. We thus conclude that, for every tree item $[X, i, j]$, $\text{arity}(X) \leq \max\{\ell, r + d\} = \rho$.

We assume that each element in Args is represented in $\mathcal{O}(1)$ space. Since we have $d \leq \rho$, we can conclude that the space requirement for each item constructed by the parsing algorithm is in $\mathcal{O}(\rho)$.

3.4.3 Deduction Rules. Using the analyses above, we can now consider a run of the algorithm on an input string w and provide upper bounds on the number of valid instantiations of each rule type in our deduction system.

Rule 0 is the simplest rule, producing tree items of the form $[X, i-1, i]$ or of the form $[X, i, i]$, with X a lexical category. The total number of lexical categories is in $\mathcal{O}(|G|)$, and we have $0 \leq i \leq |w|$. We then conclude that, in a run of the algorithm on w, the number of instantiations of rule 0 is in $\mathcal{O}(|G| \cdot |w|)$.

Considering rule 1, let us focus on the case of tree items of the form $[Y\alpha\beta, j, k]$ producing context items of the form $[/Y\alpha, \alpha\beta, i, i, j, k]$; a similar analysis can be carried out for the symmetrical case. Inspecting the consequent item, we observe that the number of possible choices is in $\mathcal{O}(|G|^{d+1} \cdot |w|^3)$, because of the duplicate occurrences of argument α and index i. Furthermore, the tree item in the premise is completely determined by the choice of the consequent item. We therefore conclude that in a run of the algorithm on w, the number of instantiations of rule 1 is in $\mathcal{O}(|G|^{d+1} \cdot |w|^3)$.

Rule 2 has premise items $[X\alpha, i', j']$ and $[\alpha, \beta, i, i', j', j]$, and consequent item $[X\beta, i, j]$. We have already established that the number of possible consequent items is in $\mathcal{O}(|G|^{d+1} \cdot |w|^2)$. Since $|\beta| \leq d$, category $X\beta$ can be split into X and β in at most $d + 1$ ways. Finally, recall that $1 \leq |\alpha| \leq 2$, thus the number of choices for α is in $\mathcal{O}(|G|^2)$. From all of the previous observations, and taking into account the extra indices i', j', we conclude that the number of instantiations of rule 2 is in $\mathcal{O}(d \cdot |G|^{d+3} \cdot |w|^4)$.

Finally, consider rule 3 with premise items $[\alpha, \beta\alpha', i'', i', j', j'']$, $[\alpha', \beta', i, i'', j'', j]$ and with consequent item $[\alpha, \beta\beta', i, i', j', j]$. We have already established in Section 3.4.2 that the number of possible consequent items is in $\mathcal{O}(|G|^{d+2} \cdot |w|^4)$. From the side conditions of rule 3, we know that the argument sequence $\beta\beta'$ in a consequent item must be split in such a way that $|\beta'| \leq 2$, which amounts to $\mathcal{O}(1)$ possible choices. Furthermore, we have $1 \leq |\alpha'| \leq 2$ and thus the number of choices for α' is in $\mathcal{O}(|G|^2)$. Accounting for the possible range of the indices i'', j'', we then conclude that the number of instantiations of rule 3 is in $\mathcal{O}(|G|^{d+4} \cdot |w|^6)$.

Putting everything together, and observing that $d \leq |G|$, we conclude that in a run of the parser on input w the total number of valid instantiations of deduction rules is dominated by rules of type 3 and is in $\mathcal{O}(|G|^{d+4} \cdot |w|^6)$.

3.4.4 Runtime. We have established bounds on the total number of items and valid instantiations of deduction rules for input w. We now provide an upper bound on the runtime of our algorithm. While our analysis is one of the main results of this article, it is based on a rather naive implementation of the algorithm, one that is only of theoretical significance. Alternative implementations of the algorithm can be developed that are slightly more involved, but will work more efficiently in practice.

Given as input a grammar G and a string w, we start by constructing a table $\mathcal{R}$ with all instantiations of deduction rules of types 1, 2, and 3, including those instantiations that are never used by the computation on w. Because there are $\mathcal{O}(|G|^{d+4} \cdot |w|^6)$ instantiations, and because the size of each item is $\mathcal{O}(\rho)$, the space requirement for $\mathcal{R}$ is in $\mathcal{O}(\rho \cdot |G|^{d+4} \cdot |w|^6)$.

Furthermore, for each item I we construct a list $\mathcal{L}(I)$ of all rules in $\mathcal{R}$ where item I occurs as an antecedent. Note that each rule in $\mathcal{R}$ can appear in at most two lists $\mathcal{L}(I)$. Therefore the total space requirement for all lists $\mathcal{L}(I)$ is in $\mathcal{O}(\rho \cdot |G|^{d+4} \cdot |w|^6)$. It is not difficult to see that the data structures $\mathcal{R}$ and $\mathcal{L}(I)$ can be constructed in time $\mathcal{O}(\rho \cdot |G|^{d+4} \cdot |w|^6)$ as a preprocessing of the grammar.

Our algorithm maintains a chart $\mathcal{C}$ where all items constructed by the parser while processing w are added. The total number of such items is in $\mathcal{O}(|G|^{d+2} \cdot |w|^4)$, and thus the space requirement for $\mathcal{C}$ is in $\mathcal{O}(\rho \cdot |G|^{d+2} \cdot |w|^4)$. We also use an agenda $\mathcal{A}$ where we store items that have been derived by the parser but have not yet been processed and added to $\mathcal{C}$.

We start parsing by initializing $\mathcal{A}$ with all items that can be produced by rules of type 0 applied to w. This phase can be executed in time $\mathcal{O}(\rho \cdot |G| \cdot |w|)$. We then iterate the following steps, until $\mathcal{A}$ becomes empty:

1. pop some item $I \in \mathcal{A}$ and add I to $\mathcal{C}$
2. mark as 'active' each occurrence of I appearing in $\mathcal{R}$ as an antecedent
3. if some rule $R \in \mathcal{R}$ gets all of its antecedents marked as active,
   (a) let I_R be the consequent item of R
   (b) if $I_R \notin \mathcal{A} \cup \mathcal{C}$, add I_R to $\mathcal{A}$
   (c) remove R from $\mathcal{R}$.

We start by observing that each item I is processed only once by the algorithm. This is certainly true in the initialization phase, since rules of type 0 always produce different items. Furthermore, we observe that in each iteration of the main loop the test condition in step (b) guarantees that items are never doubled within our agenda $\mathcal{A}$.

To analyze the time complexity of the main loop of the algorithm, we proceed by considering the execution time of each individual step. We then amortize this amount of time among the rules in $\mathcal{R}$ involved in the step itself in such a way that each rule gets charged an overall amount of time in $\mathcal{O}(\rho)$.

- We process item I at step 1 in time $\mathcal{O}(\rho)$, that is, in time proportional to the size of I itself, which we assume to be the time required for insertion into $\mathcal{C}$. We charge this amount of time to the rule that has added item I to $\mathcal{A}$.
- When processing item I at step 2, we retrieve list $\mathcal{L}(I)$ in time $\mathcal{O}(\rho)$. We then mark as active each occurrence of I in $\mathcal{L}(I)$. Furthermore, we collect

rules in $\mathcal{L}(I)$ having all of their antecedents marked as active at this time. The execution of step 2 can be amortized by charging time $\mathcal{O}(\rho)$ to each processed rule. We assume this is the time required for accessing the respective rule in $\mathcal{R}$.

- As for the inner loop at step 3, we assume that we can test membership of an item I in $\mathcal{A}$ and in $\mathcal{C}$ in time $\mathcal{O}(\rho)$. In the execution of this step, we charge time $\mathcal{O}(\rho)$ to each rule collected at step 2.

In the above analysis of the main loop of our algorithm, each rule in $\mathcal{R}$ is charged with an amount of time in $\mathcal{O}(\rho)$. Because $|\mathcal{R}| \in \mathcal{O}(|G|^{d+4} \cdot |w|^6)$, we conclude that the running time of the main loop is in $\mathcal{O}(\rho \cdot |G|^{d+4} \cdot |w|^6)$. This is also the dominating quantity in the execution of the whole algorithm. Note that ρ is negligibly small in comparison to the other factors.

4. Correctness

In this section we prove the correctness of the deduction system by first showing its soundness and then its completeness.

We start with some notation and definitions. Given a derivation tree t (respectively, context c) and a node u, we write $\mathrm{cat}(t, u)$ (respectively, $\mathrm{cat}(c, u)$) for the category labeling u in t (respectively, c). This is more precise than writing $\mathrm{cat}(u)$ and will help to clarify in which derivation tree or context a node occurs.

Next, we introduce the notion of signature, which associates some specific pieces of derivations with items of our deduction system.

Definition 1
A derivation tree t with root r has **signature** $[X, i, j]$ if

1. the yield of t is $w[i, j]$ and
2. $\mathrm{cat}(t, r) = X$ with $X \in \mathcal{G}$.

Definition 2
A derivation context c with root r and foot f has **signature** $[\alpha, \beta, i, i', j', j]$ if

1. the yield of c is $(w[i, i'], w[j', j])$ and
2. for some X we have $\mathrm{cat}(c, f) = X\alpha$ and $\mathrm{cat}(c, r) = X\beta$.

4.1 Soundness

We will prove the soundness of our deduction system by induction on the number of inference steps applied. In particular, we will show that for each item inferred by the system, there exists a corresponding derivation tree or context. The base case is clear from the correctness of the axioms. For each input word or the empty string ε, and for the corresponding lexicon entry, there is a derivation tree consisting of one unary node labeled by that lexical category with its child labeled by the corresponding input word or ε. For the inductive case, we inspect the inference rules. Assuming that there exist valid derivation trees or contexts corresponding to the antecedent(s) of a rule, we establish that this also is the case for the consequent.

Deduction Rule 1. Given a derivation tree with signature $[Y\alpha\beta, j, k]$, we can construct a context c with signature $[/Y\alpha, \alpha\beta, i, i, j, k]$ (respectively, $[\backslash Y\alpha, \alpha\beta, j, k, l, l]$) by regarding $Y\alpha\beta$ as a secondary input category for a primary input category $X/Y\alpha$, where X is a placeholder for an arbitrary category (see Figure 5a). Depending on the length of α, the performed operation is either a composition or substitution rule.

Deduction Rule 2. Given a derivation tree t' with signature $[X\alpha, i', j']$ and a context c with signature $[\alpha, \beta, i, i', j', j]$, we can obtain the desired derivation tree t with signature $[X\beta, i, j]$ by inserting t at the foot node of c (see Figure 5b).

Deduction Rule 3. Given two contexts c_1 with signature $[\alpha, \beta\alpha', i'', i', j', j'']$ and c_2 with signature $[\alpha', \beta', i, i'', j'', j]$, we obtain a context c with signature $[\alpha, \beta\beta', i, i', j', j]$ by inserting c_1 at the foot node of c_2 (see Figure 5c). Let f be the foot node of c, let s be the node where c_1 was inserted (previously the foot node of c_2), and let r be the root node of c. Then we have $\text{cat}(c, f) = X\alpha$, $\text{cat}(c, s) = X\beta\alpha'$, and $\text{cat}(c, r) = X\beta\beta'$. To show that c is a valid context, we have to verify that context conditions (1) $\text{downarity}(f) \leq \text{downarity}(u)$ and (2) $\text{arity}(r) \leq \text{arity}(u)$ both hold, where u is an arbitrary node properly between f and r.

Condition (1) is immediately fulfilled for every node properly between f and s due to the fact that c_1 is a context. To see that the condition also applies to s, note that $\text{downarity}(f) = \text{arity}(X) \leq \text{arity}(X\beta) = \text{downarity}(s)$. Since for every node v properly between s and r we have $\text{downarity}(s) \leq \text{downarity}(v)$, it also follows that $\text{downarity}(f) \leq \text{downarity}(v)$, so the condition is fulfilled for each node properly between f and r.

For condition (2), note that rule 3 is restricted such that $|\beta'| \leq |\alpha'|$. As a result, we have $\text{arity}(r) \leq \text{arity}(s)$. Because c_2 is a context, each node properly between s and r has at least the arity of r as well. Further, since c_1 is a context, each node properly between f and s has at least the arity of s, which has lower bound $\text{arity}(r)$. In summary, for each node u properly between f and r the condition $\text{arity}(r) \leq \text{arity}(u)$ is satisfied.

Complete Derivation Tree. Finally, we can conclude that if the deduction system yields an item $[S, 0, |w|]$, there is a derivation tree rooted in the distinguished category S that comprises the entire input.

4.2 Completeness

In the following we prove the completeness of our deduction system; namely, we show that if there exists a CCG derivation rooted in the distinguished category S and generating the entire input string w, then item $[S, 0, |w|]$ can be inferred by our deduction system. In order to do this, we prove a stronger statement: We show that, for every derivation tree or derivation context having signature I, the deduction system can infer item I. The proof strategy is an induction on the number of nodes in the derivation tree or derivation context, but in the case of derivation contexts we explicitly exclude the foot node from this count.

Our deduction system has been specified in Section 3.3.2 by regarding the binary rules as operations that extend derivation trees or contexts using contexts whose signatures have already been inferred. In this perspective, derivation trees are assembled or composed out of smaller derivation parts. For the completeness proof, we take the opposite perspective: When starting with a sufficiently complex derivation

tree or context associated with some valid signature, we show that it can be split or decomposed into two smaller derivation parts associated with valid signatures.

In order to do this, given such a derivation tree or context, we identify a so-called **split node**, which always lies on the spine and constitutes the position where we can decompose the derivation. More precisely, in the case of contexts, the split node is chosen among all spinal nodes having smallest downstep arity as the one closest to the foot node. In the case of derivation trees, there are two different scenarios. If there are spinal nodes with arity lower than the root, the one closest to the root is chosen. Otherwise, if all spinal nodes have arity larger than or equal to the arity of the root, the split node is chosen among the nodes having smallest downstep arity as the one closest to the lexical anchor. In the latter case, the split node can be the lexical anchor itself.

Example 7
Figure 8 illustrates the splitting strategy for derivation trees and contexts. The diagram shows the arities and downstep arities along the spine of a derivation tree with lexical anchor f and root node r. Assume a sequence of spinal nodes denoted by $u_0, u_1, \ldots, u_n$, where $u_0 = f$ and $u_n = r$. Then each application of a combinatory rule at a node u_i can be viewed as having the following steps: the arity before the rule application ($\text{arity}(u_i)$), the arity after the removal of bridging arguments ($\text{downarity}(u_i)$), and the arity after adding the excess ($\text{arity}(u_{i+1})$). In this way, the progression of arities along the spine is plotted as the sequence $\text{arity}(u_0), \text{downarity}(u_0), \ldots, \text{arity}(u_{n-1}), \text{downarity}(u_{n-1}), \text{arity}(u_n)$. The vertical lines of the grid mark $\text{arity}(u_i)$ for $i \in \{0, \ldots, n\}$, whereas the respective downstep arities are placed in between two lines of the grid.

Given a derivation tree with the spine following the depicted pattern, the first split node is u_6. There are spinal nodes u_5, u_6 with arity lower than $\text{arity}(r)$, thus among these nodes, u_6 is chosen as the one closer to r. Accordingly, the derivation tree is split into a smaller derivation tree t' that is rooted in u_6 and a derivation context with foot node u_6. All spinal nodes in t' have arity at least $\text{arity}(u_6)$. Therefore we consider the nodes with lowest downstep arity, namely u_4, u_5, and choose as the split node the one closer to f, namely, u_4. As a result, t' is split into a derivation tree t'' with a spine from f to u_4 and a derivation context with a spine from u_4 to u_6. The split node of t'' is f, dividing it into a trivial derivation tree consisting only of the lexical anchor f, and a derivation context c with a spine from f to u_4.

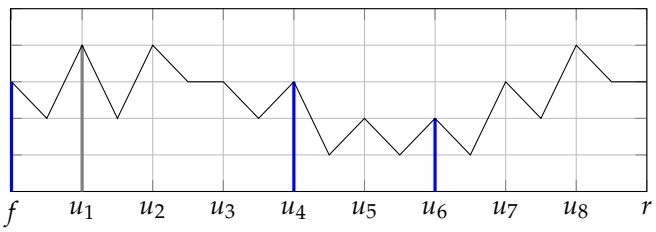


Figure 8
Diagram of arities and downstep arities along the spine of a derivation tree with interesting split nodes highlighted.

Like all other derivation contexts obtained so far, c is split at one of the nodes with lowest downstep arity, excluding the foot node. From the candidates u_1, u_3, the node u_1 is chosen as the one closer to f. All non-trivial contexts are handled in the same manner, until there is one context for each combinatory rule application.

4.2.1 Base Case: Derivation Trees Consisting of a Single Spinal Node. Consider a derivation tree t with spine length 1, root node r, and span $[i, j]$, in which $\text{cat}(t, r)$ is a lexical category. We shall write this category as X. The child of r is labeled by an input word w or by ε. We distinguish two cases depending on this label: Either there exists a lexicon entry $w := X$ and $j = i + 1$, or there exists a lexicon entry $\varepsilon := X$ and $j = i$. In either case, the item $[X, i, j]$ is one of the axioms of our deduction system.

4.2.2 Inductive Case 1: Contexts with Two Spinal Nodes. Consider a context c with spine length 2 and span $[i, i', j', j]$. This context takes one of the following two forms, where X is some category, $|Y$ is an argument, $\alpha \in \text{Args}^{\leq 1}$, and $\alpha\beta \in \text{Args}^{\leq d}$:

$$\frac{\overset{\bigtriangledown}{X/Y\alpha \quad Y\alpha\beta}}{X\alpha\beta} \qquad \frac{\overset{\bigtriangledown}{Y\alpha\beta \quad X\backslash Y\alpha}}{X\alpha\beta}$$

We write r for the root node of c, and f for the foot node. The category $\text{cat}(c, f)$ takes the form $X|Y\alpha$; the category $\text{cat}(c, r)$ takes the form $X\alpha\beta$. We assume that we already have a derivation for the subtree with root category $Y\alpha\beta$ that ends at the secondary child of the root node. Then with an application of some rule (forward or backward), we get a derivation for the complete context. We have $|\alpha| = 1$ if this rule is a substitution rule and $|\alpha| = 0$ if it is a composition rule. This operation is performed by rule 1 of the deduction system.

4.2.3 Inductive Case 2: Splitting Derivation Trees. This case corresponds to rule 2 of the deduction system. Assume a derivation tree t with root node r and lexical anchor f, having signature $[X\beta, i, j]$ and containing at least two spinal nodes. We will show how to identify the split node s at which t can be decomposed into a smaller derivation tree t' with signature $[X\alpha, i', j']$ and a context c with signature $[\alpha, \beta, i, i', j', j]$. Recall that $X\beta \in \mathcal{G}$ by the definition of signature. The derivation tree t' is the subtree of t rooted in s and the context c is the context that remains when all proper descendents of s are removed from t. Note that we can also choose f as the split node s. In this case the resulting tree is trivial and corresponds to an axiom of the deduction system. Also note that after splitting, there exist two copies of s, namely the foot node of c and the root node of t'. For the splitting of trees, we distinguish two subcases, depending on whether there is at least one spinal node with arity smaller than the root. As we will see below, these two subcases correspond to the two conditions $|\alpha| < |\beta|$ and $|\alpha| \geq |\beta|$. For each of these subcases we will show that t' has its root node labeled by some category from $\mathcal{G}$ and that c fulfills the context conditions.

Subcase 1. For this subcase, assume there is at least one spinal node n with $\text{arity}(n) < \text{arity}(r)$. We choose as split node s the node closest to r with this property (this can also be f). We can write $\text{cat}(t, s) = X\alpha$ and $\text{cat}(t, r) = X\beta$, where α are the bridging

arguments of the rule applied at s and $|\alpha| < |\beta|$ by the assumption of this subcase. Because we chose the node closest to r with the given property, the arguments in X are not modified between s and r.

Because $|\alpha| \in \{1, 2\}$ and $|\alpha| < |\beta|$, we have $|\beta| \geq 2$. From $X\beta \in \mathcal{G}$ it follows that X is a prefix of a lexical category ($X\beta \in \mathcal{G}_1$) or X is a prefix of an instantiation of a secondary input category ($X\beta \in \mathcal{G}_2$). From $|\alpha| \leq 2$ and $|X\alpha| < |X\beta|$ it follows that $X\alpha \in \mathcal{G}$, where $X\alpha \in \mathcal{G}_1$ if $X\beta \in \mathcal{G}_1$ and $X\alpha \in \mathcal{G}_2$ if $X\beta \in \mathcal{G}_2$. Consequently, $[X\alpha, i', j']$ is a valid tree item.

Furthermore, we need to verify that c is a valid context. Recall that we have to check the context conditions (1) $\text{downarity}(s) \leq \text{downarity}(u)$ and (2) $\text{arity}(r) \leq \text{arity}(u)$, for each node u properly between s and r. For the first condition, we observe that for two nodes v, w with $\text{arity}(v) < \text{arity}(w)$, we always have $\text{downarity}(v) \leq \text{downarity}(w)$. Each node u properly between s and r has the property $\text{arity}(s) < \text{arity}(u)$, and thus $\text{downarity}(s) \leq \text{arity}(u)$. For the second condition, we have $\text{arity}(r) \leq \text{arity}(u)$ because s is the node closest to r with $\text{arity}(s) < \text{arity}(r)$. We can conclude that c is a valid context.

Subcase 2. For this subcase, assume that every spinal node n has $\text{arity}(n) \geq \text{arity}(r)$. Among the spinal nodes with minimal downstep arity, we then choose the split node as the node s closest to f (this node can also be f itself). In other words, when starting at the lexical anchor and moving down toward the root node, we choose the last position where an argument of the lexical category is removed. Again, we can write $\text{cat}(t, s) = X\alpha$ and $\text{cat}(t, r) = X\beta$, where α are the bridging arguments of the rule applied at s and X is the prefix that is not modified between s and r. By the assumption of this subcase, $|\alpha| \geq |\beta|$.

By the choice of s, we know that X is a prefix of the lexical category labeling f. This is because for all nodes u properly between s and f we have $\text{downarity}(s) < \text{downarity}(u)$ and additionally $\text{downarity}(s) < \text{downarity}(f)$. Moreover, since nodes with lower arities than s have at most the downstep arity of s, if there existed such nodes closer to f, they would have been preferred as the split node. As a consequence, we have $\text{arity}(s) \leq \text{arity}(f)$. Thus, $\text{arity}(X\alpha)$ with $\alpha \in \{1, 2\}$ does not exceed the arity of the lexical category labeling f and it follows that $X\alpha \in \mathcal{G}_1 \subseteq \mathcal{G}$.

Next, we will show that c is a context. Let u be any node properly between s and r. Condition (1) $\text{downarity}(s) \leq \text{downarity}(u)$ is fulfilled because split node s was picked from the nodes with minimal downstep arity, so by this choice the statement is true. Condition (2) $\text{arity}(r) \leq \text{arity}(u)$ is already fulfilled by the assumption of this subcase. Therefore, c is a valid context item.

4.2.4 Inductive Case 3: Splitting Contexts. This case corresponds to deduction rule 3. Given a context c with more than two spinal nodes, having root node r, foot node f, and signature $[\alpha, \beta\beta', i, i', j', j]$, we will show that there is a spinal node s such that we can decompose c into two valid contexts c_1 and c_2, where c_1 has root node s, foot node f, and signature $[\alpha, \beta\alpha', i'', i', j', j'']$, whereas c_2 has root node r, foot node s, and signature $[\alpha', \beta', i, i'', j'', j]$. We can write $\text{cat}(c, f) = X\alpha$, $\text{cat}(c, s) = X\beta\alpha'$, and $\text{cat}(c, r) = X\beta\beta'$, where β might be empty.

We choose as the split node s from the spinal nodes properly between f and r among those with minimal downstep arity the one closest to f.

To show that c_1 is a context, let u be any node properly between f and s. Condition (1) $\text{downarity}(s) \leq \text{downarity}(u)$ already holds in c and thus also in c_1. For condition (2) $\text{arity}(s) \leq \text{arity}(u)$, first note that $\text{downarity}(s) < \text{downarity}(u)$, because s is the node closest to f with minimal downstep arity. Now it suffices to argue that if there

existed a node v with $\text{arity}(v) < \text{arity}(s)$ properly between f and s, this node would also have $\text{downarity}(v) \leq \text{downarity}(s)$, a contradiction to the previous observation.

As for c_2, let u be any node properly between s and r. First, we have condition (1) $\text{downarity}(s) \leq \text{downarity}(u)$ because s was chosen among the nodes with lowest downstep arity. Second, condition (2) $\text{arity}(r) \leq \text{arity}(u)$ is already fulfilled in c.

It remains to show that deduction rule 3 can actually be applied. For this, we only have to verify that $|\alpha'| \geq |\beta'|$ holds. This is easy to see since $\text{arity}(s) \geq \text{arity}(r)$ with $\text{cat}(c, s) = X\beta\alpha'$ and $\text{cat}(c, r) = X\beta\beta'$.

5. Construction of the CCG Derivation Tree

The algorithm presented in Section 3 is a recognition algorithm, that is, the algorithm decides whether a given input sentence w can be generated by some underlying CCG. However, in view of downstream natural language processing applications, we want to provide the syntactic analyses of w, here in the form of CCG derivation trees.

We call **parse tree** any tree structure whose nodes are labeled by items produced by our deduction system when processing w, and whose arcs connect each antecedent item to its consequent item. As in the case of several tabular parsing algorithms based on context-free grammars (Kallmeyer 2010, Chapter 3), one can easily adapt the algorithm of Section 3 to construct a compact representation for the forest of all parse trees for w, as described in what follows.

Whenever an item I is produced by our algorithm by means of some deduction rule, we create a tuple of so-called backpointers, referencing to the antecedent items used by the rule itself. Because I can be produced by several deduction rules, each item I is associated with a list $\mathcal{H}(I)$ of backpointer tuples. After a successful run of the algorithm on w, we can then extract any parse tree from the parsing table through the following procedure: Start at item $[S, 0, |w|]$, arbitrarily pick up a tuple t in $\mathcal{H}([S, 0, |w|])$, and recursively apply the procedure to all of the backpointers in t.[6]

As a side remark, we observe that for some item I it might happen that, in the process of following the backpointers stored in $\mathcal{H}(I)$, we end up reaching I itself. In other words, the constructed parse forest contains some cycles. This happens because our CCGs assign categories to ε, resulting in infinite ambiguity for some strings. In these cases the above procedure for extracting parse trees may never stop, for some specific choices of backpointers.

Example 8
Assume we parse input ab on the basis of a CCG with the lexicon entries $a := S/B$, $\varepsilon := B/B$, and $b := B$. Figure 9 depicts the resulting parse forest. Due to the ε-entry, we can use arbitrarily many copies of lexical category B/B that lead to cycles in the parse forest. The list of backpointer tuples is shown above the associated item. For axioms, this list starts with a nullpointer, drawn as a white circle. The other tuples in the list point to the antecedent item(s) of the considered item. There are three cycles in this parse forest. First, tree item $[S/B, 0, 1]$ can be combined with $[/B, /B, 0, 0, 1, 1]$, yielding the same tree item $[S/B, 0, 1]$ again. Second, context item $[/B, /B, 0, 0, 1, 1]$ can be combined with itself, yielding the same context item again. Third, context item $[/B, /B, 0, 0, 1, 1]$ can also be combined with $[/B, \varepsilon, 0, 0, 1, 2]$, resulting in consequent item $[/B, \varepsilon, 0, 0, 1, 2]$.

6 A backpointer tuple can also be viewed as a hyperedge. In this way the forest of all parse trees for w becomes a hypergraph; see Klein and Manning (2001).

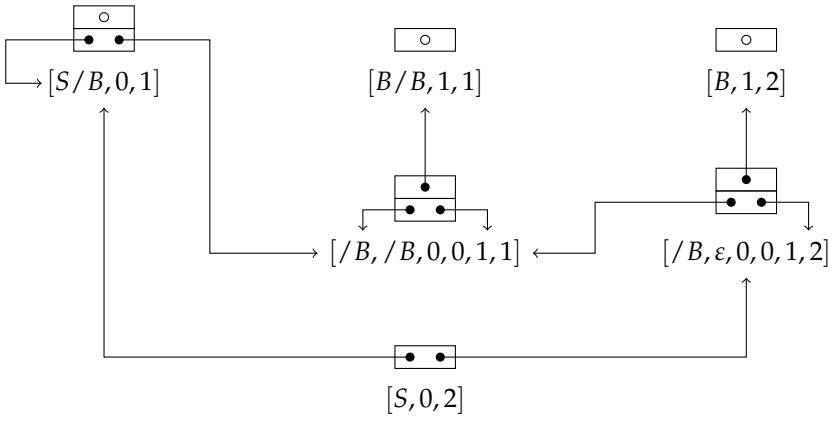


Figure 9
Parse forest containing cycles.

In practice, the extraction procedure is driven by a probabilistic model or by the use of other kinds of scores, in such a way that we can retrieve the most likely parse trees. While an item is a unique identifier for nodes of a parse forest, due to cycles, it can label several nodes of a parse tree. In what follows, we focus on the individual parse trees extracted from the parse forest, and describe how to transform these parse trees into CCG derivation trees.

5.1 Formal Definition

We present the construction of the CCG derivation tree using recursive functions that can be applied to a suitable parse tree after its extraction. The parse tree is processed in a top–down fashion, which enables us to immediately construct the derivation tree using the correct categories. More specifically, if a parse tree is rooted in a context item, without further information it is not clear what the prefix of the categories on the spine of the corresponding derivation context is. From this parse tree we can only reconstruct the combinatory rules applied along the spine, but not the exact categories labeling it. Because of this, we pass the intended root category, which should label the root of the derivation context in the complete derivation tree, as a parameter to the function that handles such parse trees. This category depends on the ancestor items and possibly on the sibling item of the regarded parse tree, which is why it can easily be determined by the top–down algorithm.

We start with some auxiliary notation. We use the standard notation for trees as introduced in Section 2.3. For simplicity, we use the special symbol $\Box$ to mark the foot node of a context instead of writing out its category explicitly. Given a context c and a tree t, we write $c\langle t\rangle$ for the tree that results from replacing $\Box$ by t in c. Similarly, when combining contexts c and c' by replacing $\Box$ by c' in c, we write $c\langle c'\rangle$. We access the category stored in a tree item by $\mathrm{category}([X\alpha, i, j]) = X\alpha$, and we access the yield corresponding to an item by $\mathrm{itemyield}([X\alpha, i, j]) = w[i, j]$, where w is the input string that was passed to the algorithm. Accordingly, the yield corresponding to an axiom is either an input word or ε. To attach the primary and secondary subtrees in the correct order, we require directionality information regarding the first bridging argument of each context item, so let $\mathrm{direction}([|Y\alpha, \beta, i', i, j, j']) = |$. Finally, given a

context item and the intended root category of the corresponding derivation context, let $\text{footcat}([\alpha, \beta, i, j, k, l], X\beta) = X\alpha$ be the corresponding foot category.

We define the recursive function **dtree**, which takes a parse tree rooted in a tree item and returns a derivation tree. We also define the recursive function **dcontext**, which takes a parse tree rooted in a context item as well as the intended root category and returns a derivation context. Note that in the former case, the root node can be a leaf or a binary node, and in the latter case, the root node can be a unary or binary node. In what follows, I is an item, p_1, p_2, and p' are parse trees, and X is a category.

$$\begin{aligned}
\text{dtree}(I) &= \text{category}(I)(\text{itemyield}(I)) \\
\text{dtree}(I(p_1, p_2)) &= \text{dcontext}(p_2, \text{category}(I))\langle\text{dtree}(p_1)\rangle \\
\text{dcontext}(I(p'), X) &= \begin{cases} X(\square, \text{dtree}(p')) & \text{if direction}(I) = / \\ X(\text{dtree}(p'), \square) & \text{if direction}(I) = \backslash \end{cases} \\
\text{dcontext}(I(p_1, p_2), X) &= \text{dcontext}(p_2, X)\langle\text{dcontext}(p_1, X')\rangle \\
&\quad \text{where } X' = \text{footcat}(\text{root}(p_2), X)
\end{aligned}$$

Whereas $\square$ is a generic placeholder, the foot node is actually associated with a specific label, which we omitted for simplicity. When inserting a derivation context at the foot node of another context, the correct correspondence of the foot category and the inserted root category is ensured since the foot category of a context is calculated and then passed as the root category to the context that gets inserted later at that exact position. This root category correctly ends in the excess of that context by design of the deduction rules. Likewise, when a tree is inserted, the root node of the context that is wrapped around is set appropriately to ensure consistency.

Concerning the order of insertion, in the previous section we have seen that while splitting a derivation tree or context, the derivation part closer to the foot node is the one that corresponds to the first antecedent and the one closer to the root node corresponds to the second antecedent. As a natural consequence, the derivation part that corresponds to the first antecedent has to be inserted into the derivation part that corresponds to the second antecedent. Note also that multiple parse trees can correspond to the same derivation tree. This will be discussed in detail in Section 6.1.

Example 9
Figure 10 depicts a parse tree extracted from the parse forest for some input string $w_1 \dots w_6$. Figure 11 shows the CCG derivation tree obtained using the recursive procedure of Section 5.1. The derivation trees and contexts resulting from the first two levels of recursion are highlighted in blue. The corresponding calculation steps are as follows. For brevity, we omit the indices in the items and indicate the respective parse subtrees using Gorn tree addresses (Gorn 1965).

We start at the root and find two subtrees: p_1 rooted in tree item $[A/B/E, 0, 3]$ and p_2 rooted in context item $[/B/E, \varepsilon, 0, 0, 3, 6]$. At each subtree we invoke a recursive call:

$$\text{dtree}(p) = \text{dtree}([A](p_1, p_2)) = \text{dcontext}(p_2, A)\langle\text{dtree}(p_1)\rangle$$

The call on p_1 returns a derivation tree that is inserted at the foot node of the derivation context returned by the call on p_2. The latter receives root category A as an additional

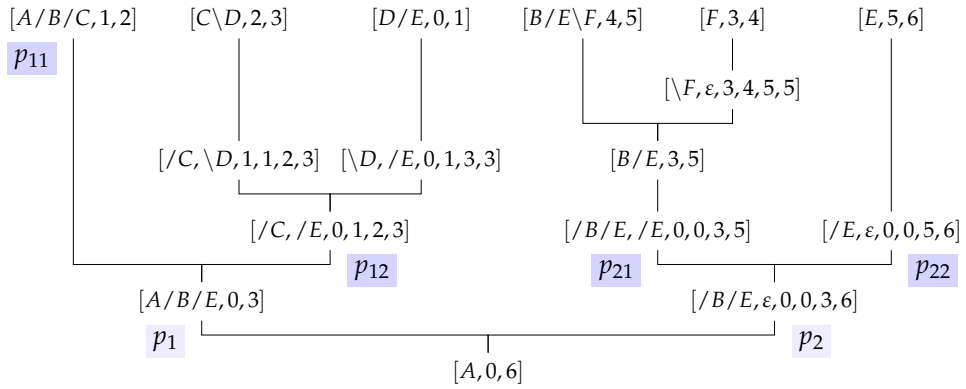


Figure 10
Parse tree with the first child of each binary node labeled by the first antecedent of the respective deduction rule. As in Figure 7, the order of leaves does not reflect the order of input categories, which can be reconstructed from the indices stored in the leaf items. The blue labels indicate subtrees referred to in Example 9 and their shade coincides with their corresponding derivation part in Figure 11.

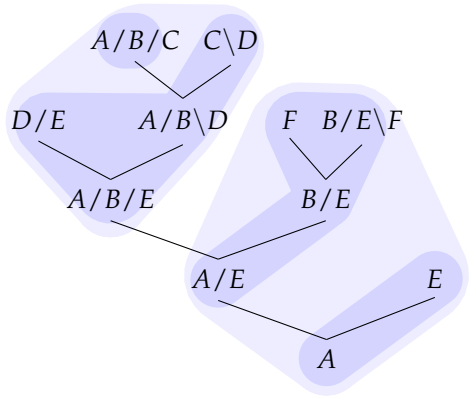


Figure 11
CCG derivation tree produced from parse tree of Figure 10. The subtrees highlighted in light blue are those resulting from the first level of recursion, and the darker shaded parts result from the second level of recursion.

parameter. In Figure 11, the derivation tree and context returned by these calls are highlighted in light blue.

Similarly, the derivation tree corresponding to p_1 is computed as follows. Category $A/B/E$ is handed down to the call yielding the context wrapping around the tree returned from the call on p_{11}. Subtree p_{11} consists of a single node $[A/B/C, 1, 2]$ and is therefore handled by the base case of the recursive function. For the other call, we directly indicate the result without showing deeper levels of recursion.

$$\begin{aligned}\text{dtree}(p_1) &= \text{dtree}([A/B/E](p_{11}, p_{12})) = \text{dcontext}(p_{12}, A/B/E)\langle\text{dtree}(p_{11})\rangle \\ &= \text{dcontext}([/C, /E](p_{121}, p_{122}), A/B/E)\langle\text{dtree}([A/B/C])\rangle \\ &= A/B/E(D/E(w_1), A/B\backslash D(\square, C\backslash D(w_3)))\langle A/B/C(w_2)\rangle\end{aligned}$$

The derivation context corresponding to p_2 is computed as follows. The subtrees p_{21}, p_{22} of $[/B/E, \varepsilon, 0, 0, 3, 6]$ correspond to derivation contexts c_{21}, c_{22} that are obtained through recursive calls of dcontext. Note that c_{22} wraps around c_{21}, which is why root category A is passed on to the call on p_{22} without modificaton. Regarding the call on p_{21}, the root category of c_{21} coincides with the foot category of c_{22} and is calculated by $\text{footcat}([/E, \varepsilon, 0, 0, 5, 6], A) = A/E$.

$$\begin{aligned}
\text{dcontext}(p_2, A) &= \text{dcontext}([/B/E, \varepsilon](p_{21}, p_{22}), A) \\
&= \text{dcontext}(p_{22}, A)\langle \text{dcontext}(p_{21}, A/E)\rangle \\
&= \text{dcontext}([/E, \varepsilon](p_{221}), A)\langle \text{dcontext}([/B/E, /E](p_{211}), A/E)\rangle \\
&= A(\square, E(w_6))\langle A/E(\square, B/E(F(w_4), B/E\backslash F(w_5)))\rangle
\end{aligned}$$

These derivation parts are then finally combined to the derivation tree of Figure 11 by performing the tree substitutions indicated in the equations above. The derivation parts corresponding to the results of the calls on p_{11}, p_{12}, p_{21}, and p_{22} are highlighted in a darker shade of blue, respectively.

6. Parser Extensions and Improvements

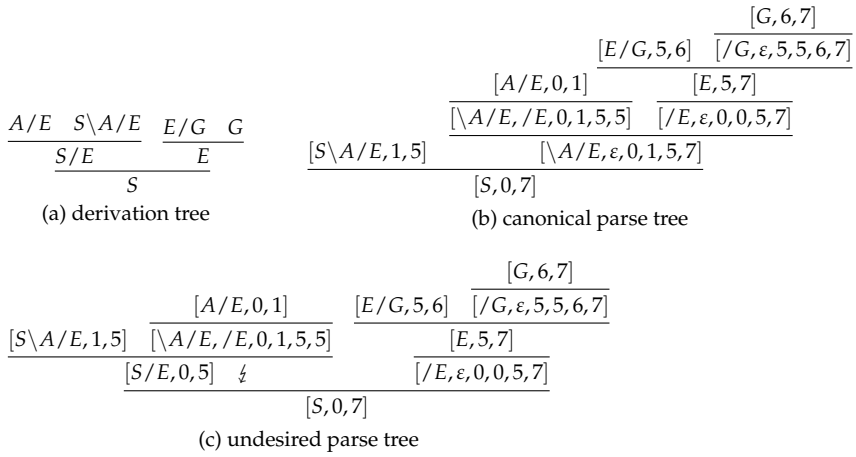
In this section we discuss possible extensions and some practical improvements to the parsing algorithm we have developed.

6.1 Eliminating Spurious Ambiguity

The term **spurious ambiguity** describes the property of a parsing algorithm to produce several parse trees for a single derivation tree. This kind of redundancy is undesirable and ought to be avoided, because it might result in flawed computations of derivation probabilities when working with generative models based on CCG (Hockenmaier and Steedman 2002). Note that there exist other notions of spurious ambiguity that aim to obtain only a single derivation tree per semantic reading by arranging forward (respectively, backward) chains (i.e., sequences of forward rule applications) in a canonical way (Eisner 1996). In the present work we will only address the former notion of spurious ambiguity.

We first observe that the algorithm as presented in Section 3 has spurious ambiguity. This can easily be seen from Figure 12, which shows two different (partial) parse trees that correspond to the same derivation tree, which is a subtree of the one in Figure 6. On the other hand, the parse tree shown in Figure 13b corresponds to the derivation tree shown in Figure 13a, which is different from the one in Figure 12a. So the parse tree is not redundant with those of Figure 12 and not a case of spurious ambuity, although it has the same leaf items. However, this derivation tree itself has several other parse trees that need to be avoided. Another example of spurious ambiguity is shown in Figure 14, depicting two parse trees corresponding to the same derivation context, where X is a category variable.

6.1.1 Sources of Spurious Ambiguity. In what follows, we first inspect our deduction rules to address the questions of whether and in what ways they might introduce spurious ambiguity. We then propose a reformulation of our deduction rules that eliminates

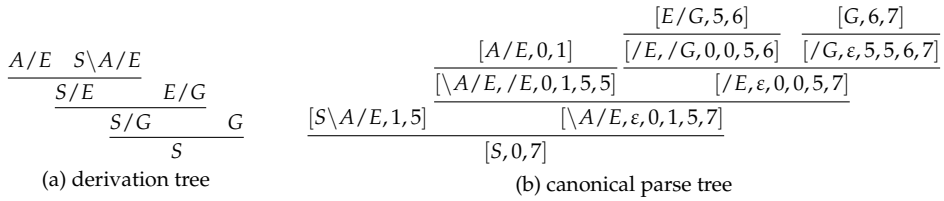


(a) derivation tree

(b) canonical parse tree

(c) undesired parse tree

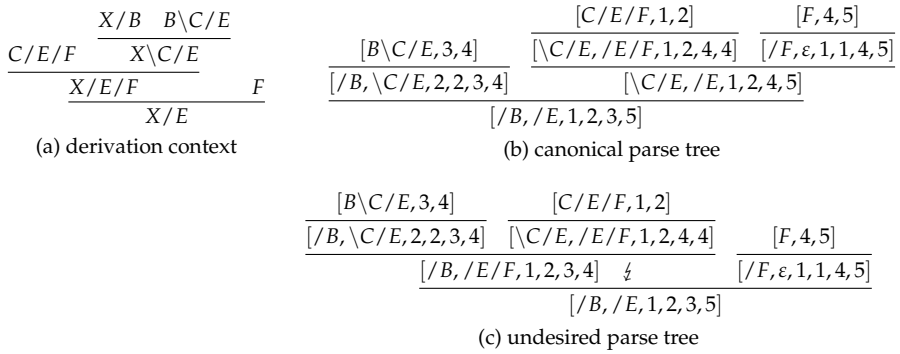
Figure 12
Spurious ambiguity caused by deduction rule 2.



(a) derivation tree

(b) canonical parse tree

Figure 13
Other derivation tree with the same lexical categories as in Figure 12.



(a) derivation context

(b) canonical parse tree

(c) undesired parse tree

Figure 14
Spurious ambiguity caused by deduction rule 3.

spurious ambiguity. Throughout the discussion, we always assume some general but fixed derivation tree, which we call the reference derivation tree.

Deduction Rule 0. This deduction rule introduces the lexical categories associated with the input symbols. It is apparent that given a reference derivation tree there is only one choice for each input symbol, providing the tree item composed of the associated lexical

category in the derivation and the index of the position in the input string. Therefore, deduction rule 0 is not responsible for any spurious ambiguity.

Deduction Rule 1. The analysis of deduction rule 1 is a bit more involved. First, we observe that deduction rule 1 is applied to exactly those tree items whose category is used as a secondary input category in our reference derivation tree. These tree items need to contain exactly the indices marking the span of the yield belonging to the subtree that is rooted in this secondary input category. Second, the leading slash of the bridging arguments and their number depends on the type of combinatory rule that is applied to that secondary input category and thus determined by the reference derivation tree as well. Third, the guessed indices of the consequent context item are the left (respectively, right) fencepost position of the reference derivation subtree that is rooted in the sibling of the secondary input category, since they mark the span that needs to be reserved for the foot node of the context. As a consequence, the derivation tree completely determines the set of items that deduction rule 1 is applied to. Any change of this set would result in a different derivation tree. This is the case for the parse trees of Figures 12b and 13b.

Deduction Rules 2 and 3. Given some spine in our reference derivation tree, its lexical anchor is introduced as a tree item by deduction rule 0, and when the root of the spine is reached, this is either the goal item or else a secondary input category, requiring an application of deduction rule 1. On the way from the lexical anchor to the root, deduction rules 2 and 3 are used to simulate the categories along the spine. Deduction rule 2 models the splitting of a derivation tree into a smaller tree and a derivation context at some spinal node, whereas deduction rule 3 models the splitting of a context into two smaller contexts, also at some spinal node. This causes spurious ambiguity because there are several points where a derivation tree or context can be split into two valid pieces of derivation that can be represented using tree or context items, respectively. Different parses of the spine therefore constitute different ways to group the combinatory rule applications along the spine into valid contexts without changing their order.

6.1.2 Approach. One solution for eliminating spurious ambiguity is to enforce that the parsing algorithm strictly follows the splitting strategy used in the completeness proof. Because the strategy of the completeness proof is pursued, the resulting modified algorithm is still complete.

When two items are combined via rule 2 or 3, we also say that the second antecedent gets added to the first antecedent. An equivalent description of the strategy of the completeness proof is that contexts are extended to contexts as large as possible before the respective context items serve as second antecedents by getting added to other items. By this, we mean that each context item used as a second antecedent, starting from its respective foot node, has to cover a segment as large as possible of the given spine and cannot be further extended in the direction of the root by adding other context items to it. Note that it might be necessary to combine other context items first to obtain an item that can be added to it. This approach leads to right-branching structures in the parse tree whenever possible.

To see that this is equivalent to the strategy of the completeness proof, note that the context whose item serves as a second antecedent is closer to the root and wrapped around the context or tree of the first antecedent. In the splitting strategy of the completeness proof, the strategy always aims to split off contexts as large as possible from

a tree or context, beginning at the root node. In one of the cases, this is made explicit by choosing the node closest to the foot node from a selection of potential split nodes (the positions with the lowest downstep arity). In the other case, it is not immediately clear by the wording of the strategy, but each larger piece of derivation contains a node with arity lower than the root node on the spine and is thus not a valid context.

Implementation. We want to ensure that a context item can never be added to an item if it could also have been added to the context item that was added to the respective first antecedent in the step before. For this, each tree and context item stores information on the last item that was added to it. It suffices to store an additional variable that can take one of three values, stating if the last added item had an excess of length 0, 1, or 2 and higher. This value is initialized with 0 after the introduction of a tree or context item via deduction rule 0 or 1 and set accordingly in the consequent item of deduction rule 2 or 3 depending on the second antecedent. When we want to add a second antecedent (always a context item) to some item, we first check if its excess is longer than its bridging arguments. This of course is only possible if the first antecedent is a tree item and would increase the arity of its stored root category. If this is the case, we may add it to the tree item regardless of the previously added item, since such a context can never be added to another context. Otherwise, if the excess length of the second antecedent is the same or lower than its number of bridging arguments, we check if the number of bridging arguments is higher than the excess length of the item that was previously added to the first antecedent. Only then combination is allowed.

Example 10
In Figure 12c, context item $C_1 = [\backslash A/E, /E, 0, 1, 5, 5]$ with excess length 1 is added to tree item $[S\backslash A/E, 1, 5]$. Subsequently, context item $C_2 = [/E, \varepsilon, 0, 0, 5, 7]$ with one bridging argument is added to the consequent item $T = [S/E, 0, 5]$. Because excess length 1 is stored in T, context item C_2 could also have been added to C_1, so this application of deduction rule 2 is not allowed (marked by $\lightning$). Similarly, in Figure 14c, after adding $C_1' = [\backslash C/E, /E/F, 1, 2, 4, 4]$ with excess length 2 to $C = [/B, \backslash C/E, 2, 2, 3, 4]$, another context item $C_2' = [/F, \varepsilon, 1, 1, 4, 5]$ with one bridging argument is added to the consequent item. Again, this is forbidden because the two second antecedents C_1', C_2' could have been combined first and then added to C in one step. Consequently, the canonical parse trees of Figures 12b and 14b are enforced.

Explanation. We claim that this approach suffices to remove all spurious ambiguity from the parsing algorithm. For this, we have the following argument.

We first focus on the splitting of trees and therefore on deduction rule 2. Given a spine, the applications of this deduction rule constitute its segmentation into contexts. The split nodes of the splitting strategy are positions where splitting has to take place necessarily, either because they have a low arity and there are no lower arities closer to the root, or because they have a low downstep arity with no lower downstep arity closer to the lexical anchor. No context on the given spine can contain these positions as nodes properly between the foot node and the root node. Thus, beyond these nodes, contexts cannot be further extended in either direction, so their context items can be used neither as first nor as second antecedent of deduction rule 3. The splitting strategy chooses exactly those nodes, showing that it yields maximally extended contexts.

Now assume there was a splitting into smaller contexts that respects the unavoidable split nodes, but additionally splits those maximal contexts into smaller ones that cannot be combined with each other to attain the desired maximal contexts.

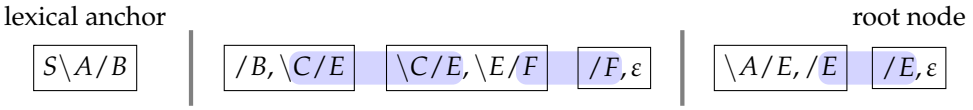


Figure 15
Grouping of combinatory rule applications along the spine of the derivation tree of Figure 6 into maximally extended contexts.

However, the extension of a context item (by adding another context item as a second antecedent) does not inhibit its ability to be added to other context or tree items. This is because the bridging arguments are unaffected and only the two topmost arguments of the excess may be exchanged or removed. In other words, extending a context further in the direction of the root node does not inhibit its ability to combine with other context or tree items in the direction of the foot node.

Additionally, when a context item is added to a tree or context item, its ability to have other context items added to it carries over to the consequent item. In other words, the consequent item can be extended further in the direction of the root node in at least the same (and possibly more) ways as the added context item can.

This shows that no combination of context items (in accordance with the given spine) prevents an extension to the maximally extended context. If the maximal context is not attained yet, its smaller parts can still be combined.

This is visualized in Figure 15. It shows the items (without indices) corresponding to the combinatory rule applications along the spine of the derivation tree of Figure 6 and how these items are grouped into maximally extended contexts. The order of these items is fixed by the reference derivation tree, so each item can only be combined with the neighboring items or their respective consequents. We can observe that after combining $[\backslash C/E, \backslash E/F]$ with $[/F, \varepsilon]$, we can still combine the consequent with $[/B, \backslash C/E]$, since the bridging arguments $\backslash C/E$ of the first antecedent are preserved. In the same manner, when combining $[\backslash C/E, \backslash E/F]$ with $[/B, \backslash C/E]$ first, the consequent can still be extended in the direction of the root by adding $[/F, \varepsilon]$ to it, since the excess $\backslash E/F$ of the second antecedent is transferred to the consequent.

With the aid of Figure 16, we will explain what happens when we add smaller contexts than the intended maximal context to some tree item. Assume that $[/B, /C/D]$ is added to $[A/B]$. Provided that the resulting category is in $\mathcal{G}$, this is allowed because the context item increases the arity of the tree item. Next, we add $[/D, /E/F]$ to the consequent. This is allowed as well since the two context items cannot be combined directly. However, it is forbidden to add $[/F, \varepsilon]$ afterward. This item reduces the arity and cannot be added to the combined item $[A/C/E/F]$, which stores the value 2 to indicate that an item with an excess of length 2 or higher was added in the previous step. So we may add several non-maximal contexts in a row that increase the arity (as

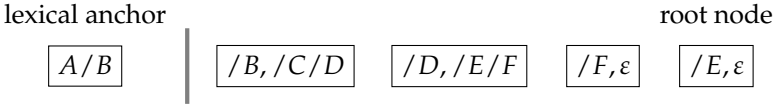


Figure 16
Group starting with two combinatory rule applications that increase the arity.

long as the result is a category in $\mathcal{G}$), but at some point we need to preserve or reduce the arity to return to the arity of the root of the maximally extended context, requiring a combination that is forbidden. Due to the context definition, this root has at most the arity of the category that is obtained by adding the first arity-increasing context.

Now we turn to the splitting of contexts and to deduction rule 3. First, we observe that the context that is split off clearly cannot be extended further in the direction of the foot node due to its low downstep arity. Then we use the same argumentation as for trees and argue that, if the context is not maximally extended yet, its parts can still be combined with each other.

Together, these properties indicate that there are no two competing splittings with non-extendable second antecedents. However, a more in-depth analysis is necessary to formally prove that spurious ambiguity is ruled out completely. This should be addressed in future work.

Separation of Splitting Strategies. A nice property of the described strategy is that by following it, the two cases of the tree splitting in the completeness proof (see 4.2.3) are strictly separated in the sense that on one side of the spine one case is used consistently. These cases not only correspond to two variants of deduction rule 2, but also correspond to guaranteed membership in the sets $\mathcal{G}_1$ or $\mathcal{G}_2$, respectively. More precisely, closer to the root, which is labeled by an instantiation of a secondary input category or the distinguished category, the categories stored in the first antecedent and the consequent item of deduction rule 2 are in $\mathcal{G}_2$ and the rule has $|\alpha| < |\beta|$ (case 1). On the other hand, closer to the lexical anchor, the first antecedent and the consequent item are in $\mathcal{G}_1$ and deduction rule 2 has $|\alpha| \geq |\beta|$ (case 2). There is a specific position on the spine where the strategies are switched, which is among the spinal nodes of lowest arity the one closest to the root. At this position, the spinal category is in $\mathcal{G}_1 \cap \mathcal{G}_2$.

To demonstrate why this is the case, examine the condition for case 2. After this condition (no lower downstep arity closer to the lexical anchor) is true for the first time, it is true at each split node that is chosen in the remaining part of the spine. Therefore, we can use this case for every split node closer to lexical anchor as well. Accordingly, deduction rule 2 is used along the spine as follows: Starting at the lexical anchor, the arity of the category is reduced until the position of lowest arity closest to the root is reached. Then, the arity is increased to construct the secondary input category labeling the root. Of course, depending on the spine, it can also be the case that only one of the two cases is required at all.

6.2 Support for Rule Restrictions

A support for non-pure CCG and for rule restrictions in particular is quite easy to implement. We use the same approach that was proposed by Kuhlmann and Satta (2014), who pointed out that it boils down to the same solution that was already used for the classical polynomial time parsing algorihm by Vijay-Shanker and Weir (1990, 1993).

There are two types of rule restrictions: target restrictions, which restrict the target of the primary input category, and secondary input restrictions. When a category is used as a secondary input category, a rule of type 1 is used to convert the corresponding tree item into a context item. This rule should be restricted such that it can only be applied if the category in the antecedent tree item matches an instantiation of a secondary input category of some combinatory rule and only in accordance with the type of rule (composition or substitution and forward or backward), which determines the length and

leading slash of the bridging arguments. This approach implements secondary input restrictions as well as the support for non-pure CCG. Additionally, if target restrictions are also used by the grammar, each context item needs to store the target along the spine of the corresponding derivation context. For this, when a rule of type 1 is used to introduce a context item, the target of the primary input category of the matching combinatory rule is stored in the consequent context item. Note that there can be several choices if that instantiation is admissible for several targets of primary input categories. For two context items to be combined via a rule of type 3, their stored targets have to match. Further, rules of type 2 only allow combination of tree items and context items if their respective targets coincide. Because a target is stored in each context item, the total number of context items increases (at most) with a multiplicative factor of $|A|$, where A is the set of atomic categories of G. As the targets of the items combined via rules of type 3 need to match, this leads to an overall increase of the runtime of the parser by a multiplicative factor $|A|$ as well.

6.3 Support for Multimodal CCG

While rule restrictions provide derivational control by allowing specific rules only for a subset of categories, there is a shift toward multimodal variants of CCG, which have a grammar-independent universal set of rules, but use lexically assigned *slash types* to provide additional control over the applicability of rules, leading to a fully lexicalized formalism (Baldridge 2002; Baldridge and Kruijff 2003; Steedman and Baldridge 2011; Stanojević and Steedman 2021). For example, $/_\diamond$ makes a category accessible to a forward harmonic rule; thus the combinatory rule $X/_\diamond Y, Y/_\diamond Z \Rightarrow X/_\diamond Z$ is allowed, but $X/_\diamond Y, Y\backslash_\diamond Z \Rightarrow X\backslash_\diamond Z$ is not. Both the primary and secondary input category need to be equipped with a slash type that permits the respective rule. Note that the formal properties of multimodal CCG depend on the precise specification of operators; the generative capacity can be lower than that of CCG with rule restrictions (Kuhlmann, Koller, and Satta 2010, 2015).

Our parsing algorithm can easily be adapted to multimodal variants of CCG by enriching the slashes occurring in items with the respective slash types and filtering the deduction rules accordingly. The implementation details clearly depend on the specific variant of multimodal CCG, so we will sketch the idea exemplarily. Rules of type 1 need to ensure that tree items are only transformed into context items representing combinatory rules that are permitted with the category stored in the tree item as a secondary input category. For instance, from items of the form $[Y/_\diamond Z, j, k]$, we may infer $[/_\diamond Y, /_\diamond Z, i, i, j, k]$, but not $[\backslash_\times Y, /_\diamond Z, j, k, l, l]$, where $\backslash_\times$ indicates a backward crossed rule. Here, we already set the leading slash type of the bridging arguments in accordance with the applied combinatory rule. Rules of type 2 and 3 may only be applied if the slash types at the end of the category or excess stored in the first antecedent and of the bridging arguments stored in the second antecedent are consistent.

Alternatively, some versions of multimodal CCG can be converted into an equivalent CCG with rule restrictions using the construction by Baldridge and Kruijff (2003), before using the extension described in the previous section.

6.4 Instantiated Secondary Input Categories

We have seen that the runtime complexity of the algorithm is exponential in the maximum degree of the grammar. This holds true for a CCG whose combinatory rules may contain variables in their secondary input categories, which thus require proper

instantiation with all possible lexical arguments. However, when considering a grammar where the secondary input categories in the combinatory rules do not contain any variables, we can modify the deduction system such that runtime becomes polynomial in the size of the grammar. To see this, we have to examine the items of the deduction system.

First, there exist tree items for all categories in $\mathcal{G}$ (in combination with all spans of the input, but we are only concerned with the grammar size here). $\mathcal{G}_1$ has size polynomial in $|G|$ already when secondary input categories may contain variables. The size of $\mathcal{G}_2$ becomes also polynomial if all instantiated secondary input categories are part of G, since in the same manner as for $\mathcal{G}_1$, the relevant prefix of each category in the set is already present in the grammar.

The crucial point are the context items. Instead of allowing an arbitrary argument sequence of length d or smaller in the excess, they have to be restricted such that the bridging arguments together with the excess follow the same pattern as $\mathcal{G}_2$ when the leading slash is omitted. If there is only one bridging argument, that argument concatenated with the excess has to follow the pattern of $\mathcal{G}_2$, whereas if there are two bridging arguments, only the first argument is concatenated with the excess. To understand that this suffices, consider how context items arise and develop throughout the deduction. A context item is introduced via a tree item, and this conversion means that the category in the tree item gets applied as a secondary input category. Thus, at that point, if a composition rule is simulated, the bridging arguments without the leading slash concatenated with the excess have the form of this secondary input category. If a substitution rule is simulated, one of the arguments of the secondary input category occurs twice—as the last bridging argument and as the first argument of the excess. Afterward, the context item can only be modified by a deduction rule of type 3. Each use of such a deduction rule leads to a modification of at most the two topmost arguments of the excess, either by exchanging or by removing them. Consequently, we start with a (segmented) category in $\mathcal{G}_2$ when the context item is introduced, and if the first antecedent of deduction rule 3 contained a category in $\mathcal{G}_2$ before the application, the consequent context item does as well. The argument that we omitted before concatenating bridging arguments and excess is a lexical argument, and their number is bounded by $|G|$. We can conclude that the number of context items is polynomial in $|G|$. As the number of items is polynomial in the grammar size, the same holds for the number of instantiations of deduction rules.

7. Final Remarks and Conclusion

In this article we have contributed two technical results to the literature on CCG. First, we have offered a treatment of the substitution operator, extending existing parsers which only focus on composition/application rules. Substitution rules are used in several syntactic accounts of natural language, but have been ignored so far in the development of parsing algorithms. Second, we have shown that, when the grammar G is taken into account as an input to the problem, CCG parsing can be carried out in time exponential only in the maximum degree of G's rules. Previously, the best known complexity analysis of CCG parsing accounting for grammar size reported an exponential function of a combination of three parameters: the maximum rule degree, the maximum arity of categories in G's lexicon, and the maximum arity of arguments appearing in these categories. We now know that CCGs of bounded degree can be parsed in polynomial time in the size of both the string *and* the grammar. This result

Table 2
Computational complexity of the universal recognition problem for several variants of CCG with rule restrictions.

CCG variant	Complexity
without ε-entries	NP (Kuhlmann, Satta, and Jonsson 2018)
with ε-entries	EXPTIME (Kuhlmann, Satta, and Jonsson 2018)
bounded rule degree	PTIME (this article)

is especially relevant in view of the fact that it has been observed that CCGs of bounded degree are linguistically motivated.

Table 2 summarizes known results on the computational complexity of the universal recognition problem for several variants of CCG, all with rule restrictions. The first two entries represent completeness results already discussed in the Introduction. Kuhlmann, Satta, and Jonsson (2018) studied CCG with composition rules, but all of their results hold regardless of whether substitution rules are included or not. The last entry in the table refers to the result in this article, attesting membership in PTIME for the universal recognition problem for CCG of bounded rule degree. For completeness, we remark here that PTIME-hardness for this problem easily follows from the fact that the universal recognition problem for context-free grammar is PTIME-hard. This problem can be reduced in logarithmic space to the universal recognition problem for CCG of bounded rule degree. We also remark that, on a par with CCG of bounded rule degree, the universal recognition problem for TAG can be solved in PTIME (Schabes 1990).

Our parsing algorithm achieves a runtime of $\mathcal{O}(\rho \cdot |G|^{d+4} \cdot |w|^6)$, where w is the input string, G is the input grammar with rules of degree bounded by d, and $\mathcal{O}(\rho)$ is the space required for the representation of each item. When target restrictions are allowed, the number $|A|$ of atomic categories needs to be included as a multiplicative factor. For comparison, TAG is known to be parsable in $\mathcal{O}(|G|^2 \cdot |w|^6)$ when the grammar size is taken into account (Schabes 1990).

Additionally, we have proposed a modification of our parsing algorithm for a variant of CCG that contains all admissible instantiations of secondary input categories as part of the rule system, leading to an algorithm that is polynomial in the size of the input grammar even if there is no bound on the maximum degree of combinatorial rules. This shows that the possibility to use variables in secondary input categories is crucial for the complexity results reported by Kuhlmann, Satta, and Jonsson (2018) and already discussed in the introduction.

Another worthwhile problem that the present work might facilitate is the removal of ε-entries. The currently available construction leads to an exponential blowup of the grammar size (Schiffer and Maletti 2021). As Kuhlmann, Satta, and Jonsson (2018) point out, due to the different computational complexity of the universal recognition problem for CCG with and without ε-entries, avoiding exponential runtime for ε-removal is not possible unless EXPTIME = NP, which is highly unlikely. However, our findings suggest that a variant of CCG where either the rule degree is bounded or where all rules have fully instantiated secondary input categories allows for ε-removal with only polynomial increase of the grammar size. More precisely, the techniques presented in this article can be used to improve the transformation from CCG to simple monadic context-free tree grammar (Kuhlmann, Maletti, and Schiffer 2022, Definition 23) for these variants.

This construction is the step of the ε-removal involving the exponential blowup and is structurally closely related to our parsing algorithm.

Finally, we would like to point out that a formal treatment of CCG is not only of theoretical interest, but can have practical benefits in parsing applications. This is because both directions of research rely on the same or very similar techniques, like tree rotation or compact encoding of derivation trees. These play an important role in work on the generative power of CCG (Kuhlmann, Koller, and Satta 2010, 2015; Kuhlmann, Maletti, and Schiffer 2019, 2022) and also in classical (Eisner 1996; Vijay-Shanker and Weir 1990) as well as recent practical CCG parsers (Stanojević and Steedman 2019; Kato and Matsubara 2021).

We conclude with an open problem. The already mentioned complexity results for CCG parsing reported by Kuhlmann, Satta, and Jonsson (2018) make crucial use of a combination of unbounded maximum rule degree and rule restrictions. Our novel result shows that, if we drop unbounded maximum rule degree, we can achieve polynomial time parsing both in the input string length and in the grammar size. It still remains to be assessed whether parsing can be carried out in polynomial time for pure CCGs—that is, CCGs that have all possible (unrestricted) rules up to some fixed but unbounded degree.

Acknowledgments

We would like to thank Jannis Harder, Peter Jonsson, Andreas Maletti, and Andrea Pietracaprina for discussion and valuable advice on a draft version of this article, or parts of it. We are also grateful to the three anonymous reviewers for their insightful comments and suggestions, which helped improve this article. Schiffer's work was supported by the German Research Foundation (DFG) Research Training Group GRK 1763 'Quantitative Logics and Automata'. Kuhlmann's work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation.

References

Baldridge, Jason. 2002. *Lexically Specified Derivational Control in Combinatory Categorial Grammar*. Ph.D. thesis, University of Edinburgh, Edinburgh, UK.

Baldridge, Jason and Geert-Jan M. Kruijff. 2003. Multi-modal combinatory categorial grammar. In *Tenth Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, pages 211–218. https://doi.org/10.3115/1067807.1067836

Bar-Hillel, Yehoshua, Haim Gaifman, and Eli Shamir. 1960. On categorial and phrase-structure grammars. *Bulletin of the Research Council of Israel*, 9F(1):1–16.

Buszkowski, Wojciech. 1988. Generative power of categorial grammars. In Richard T. Oehrle, E. Bach, and Deirdre Wheeler, editors, *Categorial Grammars and Natural Language Structures*, volume 32 of *Studies in Linguistics and Philosophy*. Springer, chapter 4, pages 69–94. https://doi.org/10.1007/978-94-015-6878-4_4

Eisner, Jason. 1996. Efficient normal-form parsing for Combinatory Categorial Grammar. In *34th Annual Meeting of the Association for Computational Linguistics*, pages 79–86. https://doi.org/10.3115/981863.981874

Fowler, Timothy A. D. and Gerald Penn. 2010. Accurate context-free parsing with combinatory categorial grammar. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 335–344.

Fujiyoshi, Akio and Takumi Kasai. 2000. Spinal-formed context-free tree grammars. *Theory of Computing Systems*, 33(1):59–83. https://doi.org/10.1007/s002249910004

Gorn, Saul. 1965. Explicit definitions and linguistic dominoes. In *Systems and Computer Science, Proceedings of the Conference held at Univ. of Western Ontario*, pages 77–115. https://doi.org/10.3138/9781487592769-008

Hockenmaier, Julia and Mark Steedman. 2002. Generative models for statistical parsing with Combinatory Categorial Grammar. In *Proceedings of the 40th Annual Meeting of the Association for Computational*

Linguistics, pages 335–342. https://doi.org/10.3115/1073083.1073139

Hockenmaier, Julia and Peter Young. 2008. Non-local scrambling: The equivalence of TAG and CCG revisited. In *Proceedings of the 9th International Workshop Tree Adjoining Grammar and Related Formalisms*, pages 41–48.

Joshi, Aravind K. 1985. Tree Adjoining Grammars: How much context-sensitivity is required to provide reasonable structural descriptions? In David R. Dowty, Lauri Karttunen, and Arnold M. Zwicky, editors, *Natural Language Parsing*. Cambridge University Press, pages 206–250. https://doi.org/10.1017/CBO9780511597855.007

Kallmeyer, Laura. 2010. *Parsing Beyond Context-Free Grammars*. Cognitive Technologies, Springer. https://doi.org/10.1007/978-3-642-14846-0

Kato, Yoshihide and Shigeki Matsubara. 2021. A new representation for span-based CCG parsing. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 10579–10584. https://doi.org/10.18653/v1/2021.emnlp-main.826

Kepser, Stephan and Jim Rogers. 2011. The equivalence of tree adjoining grammars and monadic linear context-free tree grammars. *Journal of Logic, Language and Information*, 20(3):361–384. https://doi.org/10.1007/s10849-011-9134-0

Klein, Dan and Christopher D. Manning. 2001. Parsing and hypergraphs. In *Proceedings of the Seventh International Workshop on Parsing Technologies (IWPT-2001)*, pages 123–134.

Koller, Alexander and Marco Kuhlmann. 2009. Dependency trees and the strong generative capacity of CCG. In *Proceedings of the 12th EACL*, pages 460–468. https://doi.org/10.3115/1609067.1609118

Kuhlmann, Marco, Alexander Koller, and Giorgio Satta. 2010. The importance of rule restrictions in CCG. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 534–543. https://www.aclweb.org/anthology/P10-1055

Kuhlmann, Marco, Alexander Koller, and Giorgio Satta. 2015. Lexicalization and generative power in CCG. *Computational Linguistics*, 41(2):187–219. https://doi.org/10.1162/COLI_a_00219

Kuhlmann, Marco, Andreas Maletti, and Lena K. Schiffer. 2019. The tree-generative capacity of combinatory categorial grammars. In *Proceedings of the 39th FSTTCS*, volume 150 of *LIPIcs*, pages 44:1–44:14.

Kuhlmann, Marco, Andreas Maletti, and Lena K. Schiffer. 2022. The tree-generative capacity of combinatory categorial grammars. *Journal of Computer and System Sciences*, 124:214–233. https://doi.org/10.1016/j.jcss.2021.10.005

Kuhlmann, Marco and Giorgio Satta. 2014. A new parsing algorithm for Combinatory Categorial Grammar. *Transactions of the Association for Computational Linguistics*, 2(Oct):405–418. https://doi.org/10.1162/tacl_a_00192

Kuhlmann, Marco, Giorgio Satta, and Peter Jonsson. 2018. On the complexity of CCG parsing. *Computational Linguistics*, 44(3):447–482. https://doi.org/10.1162/coli_a_00324

Schabes, Yves. 1990. *Mathematical and Computational Aspects of Lexicalized Grammars*. Ph.D. thesis, University of Pennsylvania.

Schiffer, Lena K. and Andreas Maletti. 2021. Strong equivalence of TAG and CCG. *Transactions of the Association for Computational Linguistics*, 9:707–720. https://doi.org/10.1162/tacl_a_00393

Shieber, Stuart M. 1985. Evidence against the context-freeness of natural language. *Linguistics and Philosophy*, 8(3):333–343. https://doi.org/10.1007/BF00630917

Shieber, Stuart M., Yves Schabes, and Fernando Pereira. 1995. Principles and implementation of deductive parsing. *Journal of Logic Programming*, 24(1–2):3–36. https://doi.org/10.1016/0743-1066(95)00035-I

Stanojević, Miloš and Mark Steedman. 2019. CCG parsing algorithm with incremental tree rotation. 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 228–239. https://aclanthology.org/N19-1020

Stanojević, Miloš and Mark Steedman. 2021. Formal basis of a language universal. *Computational Linguistics*, 47(1):9–42. https://doi.org/10.1162/coli_a_00394

Steedman, Mark. 2000. *The Syntactic Process*. MIT Press. https://doi.org/10.7551/mitpress/6591.001.0001

Steedman, Mark. 2011. *Taking Scope*. MIT Press. https://doi.org/10.7551/mitpress/9780262017077.001.0001

Steedman, Mark and Jason Baldridge. 2011. Combinatory Categorial Grammar. In Robert D. Borsley and Kersti Börjars, editors, *Non-Transformational Syntax: Formal and Explicit Models of Grammar*, Blackwell, chapter 5, pages 181–224. https://doi.org/10.1002/9781444395037.ch5

Vijay-Shanker, Krishnamurti and David J. Weir. 1990. Polynomial time parsing of combinatory categorial grammars. In *28th Annual Meeting of the Association for Computational Linguistics*, pages 1–8. https://doi.org/10.3115/981823.981824

Vijay-Shanker, Krishnamurti and David J. Weir. 1993. Parsing some constrained grammar formalisms. *Computational Linguistics*, 19(4):591–636.

Vijay-Shanker, Krishnamurti and David J. Weir. 1994. The equivalence of four extensions of context-free grammars. *Mathematical Systems Theory*, 27(6):511–546. https://doi.org/10.1007/BF01191624

Weir, David J. and Aravind K. Joshi. 1988. Combinatory categorial grammars: Generative power and relationship to linear context-free rewriting systems. In *Proceedings of the 26th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 278–285. https://doi.org/10.3115/982023.982057