Generative and Computational Power of Combinatory Categorial Grammar

Von der Fakultät für Mathematik und Informatik
der Universität Leipzig
angenommene

DISSERTATION

zur Erlangung des akademischen Grades

DOCTOR RERUM NATURALIUM
(Dr. rer. nat.)

im Fachgebiet
Informatik

vorgelegt von

Lena Katharina Schiffer, M. Sc.
geboren am 18. Mai 1992 in Lingen an der Ems

Die Annahme der Dissertation wurde empfohlen von:

1. Prof. Dr. Andreas Maletti (Universität Leipzig)
2. Prof. Dr. Mark Steedman (University of Edinburgh)

Die Verleihung des akademischen Grades erfolgt mit Bestehen
der Verteidigung am 10. Juli 2024 mit dem Gesamtprädikat
summa cum laude.

Acknowledgments

First and foremost, I would like to thank Andreas Maletti, who has been an amazing supervisor. He gave me great advice not only directly related to my research, but also on how to navigate academia in general. He has influenced me in many ways and gave me confidence in my abilities. I would also like to thank Heiko Vogler for his support as a second supervisor.

I am grateful to Mark Steedman for reviewing this dissertation and for sending me thoughtful and attentive comments on details I have missed.

This project would not have been possible without the financial support and the research environment provided by the DFG Research Training Group QuantLA. I very much appreciate that Doreen Straß made me aware of QuantLA and introduced me to Andreas, who later became my supervisor.

I was lucky to have the opportunity to work with brilliant collaborators in addition to Andreas. Marco Kuhlmann and Giorgio Satta, it has been a privilege to work with you!

Many thanks go to my colleagues for hours of enjoyable discussions. Especially the reading group has been a great help for learning how to read, understand, and think about scientific work. It has been one of the few possibilities to stay in contact with my colleagues during the pandemic. In particular, I want to thank Sven Dziadek, Erik Paul, and Karin Quaas for their helpful advice regarding the dissertation and academia in general. Andreea-Teodora Nász, Erik Paul, and Markus Ulbricht helped me a lot with their practical suggestions on how to improve my thesis defense presentation. I also want to thank my friend Jannis Harder for the inspiring discussions we had. Special thanks go to Maria Arndt and Svø Burmeister for contributing to the celebration after my defense with their great baking skills!

Finally, I wish to express my sincere gratitude to my partner Yusuke, my friends, and my family, who have supported me enormously all these years.

Abstract

Combinatory categorial grammar (CCG) is a formalism that is well-established in computational linguistics. At the basis of the grammar are a lexicon and a rule system: The lexicon assigns syntactic categories to the symbols of a given input string, and the rule system specifies how adjacent categories can be combined, yielding a derivation tree whose nodes are labeled by categories. In this thesis, we focus on composition rules, which are present in all variants of the grammar. CCG is a mildly context-sensitive formalism, thus it has a relatively high expressivity that lies in between the context-free and context-sensitive language class and is parsable in polynomial time.

As a fundamental result regarding CCG, Vijay-Shanker and Weir show that it can generate the same class of string languages as tree-adjoining grammar, linear indexed grammar, and head grammar. Their equivalence proof relies on two particular features of the grammar. The first is the admissibility of ε-entries, which are lexicon entries that assign syntactic categories to the empty word. The second is the use of rule restrictions, which allow to impose certain restrictions on the rule set on a per-grammar basis. However, modern variants of CCG tend to avoid these features. This raises the question whether what is known about the generative and computational power of CCG still holds under different circumstances. Apart from the two abovementioned features, this also concerns the rule degree, which determines how complex a certain category involved in a rule application may be. The goal of this thesis is to shed light on the effects that these changes of the formalism have. This can help to identify properties that are desirable in a grammar formalism.

When regarding the generative power of a formalism that is used for modeling natural language, one is not only interested in the acceptability of an input sentence, but also in its underlying structure. Therefore, we study the sets of constituency trees that CCG can generate, which are obtained by relabeling sets of derivation trees. We start by investigating CCG with low rule degrees. First, we provide a new proof of an analogous result by Buszkowski, showing that when only application rules are allowed, a proper subset of regular tree languages can be generated by CCG. Then, going one step further, we show that when composition of first degree is included, CCG can generate exactly the regular tree languages. On the other hand, pure CCG, which allows all rules up to some degree and thus has no rule restrictions, is shown to not even generate all local tree languages. Our main result on the generative power of CCG is its strong equivalence to tree-adjoining grammar. This means that these formalisms can generate the same class of tree languages. This is even the case when only composition rules of second degree and no ε-entries are used, showing that a CCG with these seemingly limiting properties already has its full expressive power. Our constructions also provide an effective procedure for the removal of ε-entries.

From a computational point of view, these ε-entries and a high rule degree are in fact problematic. Kuhlmann, Satta, and Jonsson investigated the complexity of the universal recognition problem for CCG, which refers to the question whether some given string is generated by some given grammar, considering also the latter as part of the input. They prove that this problem is EXPTIME-complete if ε-entries are included, and NP-complete if not. We refine this result and, by providing a new parsing algorithm, show that this problem is exponential only in the maximum rule degree of the grammar. Hence, when the rule degree is bounded by a constant, parsing becomes polynomial in the grammar size. Moreover, this also holds when substitution rules are included in the rule system, as our algorithm is able to handle these.

Contents

Acknowledgments

Abstract

1 Introduction **1**
- 1.1 Basic Concept . . . 2
- 1.2 Research Focus . . . 3
- 1.3 Generative Power . . . 4
  - 1.3.1 Weak Generative Power . . . 4
  - 1.3.2 Strong Generative Power . . . 5
  - 1.3.3 Contributions . . . 7
- 1.4 Computational Power . . . 8
  - 1.4.1 Contributions . . . 9
- 1.5 Overview of the Dissertation . . . 11

2 Preliminaries **13**
- 2.1 Basic Definitions . . . 13
- 2.2 String Languages . . . 13
  - 2.2.1 Nondeterministic Finite Automata . . . 14
  - 2.2.2 Context-Free Grammar . . . 14
  - 2.2.3 Push-Down Automata . . . 15
- 2.3 Tree Languages . . . 16
  - 2.3.1 Tree Grammars . . . 17
  - 2.3.2 Tree-Adjoining Grammar . . . 19
- 2.4 Mild Context-Sensitivity . . . 20
  - 2.4.1 Definition . . . 20
  - 2.4.2 Tree-Adjoining Languages . . . 21
  - 2.4.3 Multiple Context-Free Languages . . . 23

3 Combinatory Categorial Grammar **25**
- 3.1 Categories . . . 25
- 3.2 Rules . . . 26
  - 3.2.1 Combinatory Rules . . . 26
  - 3.2.2 Rule Restrictions . . . 27
  - 3.2.3 Instantiation . . . 28
  - 3.2.4 Rule System . . . 29
  - 3.2.5 Type-Raising . . . 29
- 3.3 Grammars . . . 30
  - 3.3.1 Generated String Language . . . 31
  - 3.3.2 Generated Tree Language . . . 33
  - 3.3.3 Lexical Arguments . . . 33

4 Generative Power for Low Rule Degrees **35**
4.1 0-CCG . . . 36
4.2 1-CCG . . . 46
4.2.1 Pure 1-CCG . . . 50

5 Generative Power **53**
5.1 Inclusion in the Simple Monadic Context-Free Tree Languages . . . 54
5.2 Proper Inclusion for Pure CCG . . . 64
5.3 Spine Grammar . . . 65
5.4 Decomposition into Spines . . . 70
5.5 Moore Push-Down Automata . . . 74
5.6 CCG Construction . . . 80
5.6.1 Relating CCG Spines and Automaton Runs . . . 86
5.6.2 Combining Spines . . . 91
5.7 Strong Equivalence . . . 94

6 Computational Complexity for Bounded Rule Degree **97**
6.1 Parsing Algorithm . . . 98
6.1.1 Definitions and Notation . . . 99
6.1.2 Algorithm Specification . . . 103
6.2 Correctness . . . 108
6.2.1 Soundness . . . 108
6.2.2 Completeness . . . 110
6.3 Runtime Analysis . . . 115
6.3.1 Argument Contexts and Root Categories . . . 115
6.3.2 Items . . . 116
6.3.3 Deduction Rules . . . 117
6.3.4 Implementation and Runtime . . . 118
6.3.5 Hardness for CCG with ε-entries . . . 120
6.4 From Parse Tree to Derivation Tree . . . 121
6.4.1 Parse Trees and Parse Forests . . . 121
6.4.2 Construction of the Derivation Tree . . . 122
6.5 Parser Extensions and Improvements . . . 125
6.5.1 Eliminating Spurious Ambiguity . . . 126
6.5.2 Support for Rule Restrictions . . . 133
6.5.3 Support for Multi-Modal CCG . . . 133
6.5.4 Instantiated Secondary Categories . . . 134

7 Conclusion **137**
7.1 Summary . . . 137
7.1.1 Generative Power . . . 137
7.1.2 Computational Power . . . 140
7.2 Discussion . . . 141
7.3 Outlook . . . 142

Bibliography **145**

Introduction 1

1.1 Basic Concept 2
1.2 Research Focus 3
1.3 Generative Power . . . 4
1.4 Computational Power 8
1.5 Overview of the Dissertation 11

Combinatory categorial grammar (CCG) [86, 88] is an extension of categorial grammar, enriching it using ideas from combinatory logic [13]. Categorial grammar [2, 7] itself took inspiration from proof theory and was introduced alongside the phrase-structure grammars of the Chomsky-Hierarchy [12]. However, having the same expressive power as context-free grammar [8], it became evident that categorial grammar was too restricted to capture the linguistic structures arising in natural language [79]. This led to the introduction of the class of mildly context-sensitive languages [37]. The properties that characterize this language class are the containment of context-free languages, the ability to express a limited amount of cross-serial dependencies, efficient parsing (i.e., in polynomial time), and the constant growth property. The two most celebrated works on CCG are the proof of its equivalence to tree-adjoining grammar (TAG) [93] (and to several other mildly context-sensitive formalisms) and a parsing algorithm with polynomial runtime [91, 92], showing that CCG indeed belongs to the mildly context-sensitive formalisms. This combination of beneficial properties–a relatively high expressivity together with efficient parsing–are not the only reasons why CCG has since become widely applied in computational linguistics [59, 60]. Another leading cause is its notion of syntactic categories, which is very natural and allows to intuitively model the combination of constituents in natural language. This clear interface between syntax and semantics makes it possible to conveniently add semantics to the formalism through lambda calculus [86]. On one hand, the desire to fully lexicalize the formalism and to be able to easily express certain syntactic structures gave rise to the development of a wide range of variants with slightly different operating principles [5, 50, 86, 88]. On the other hand, driven by statistical natural language processing, there has also been a growing interest in probabilistic variants [56, 57, 98, 99].

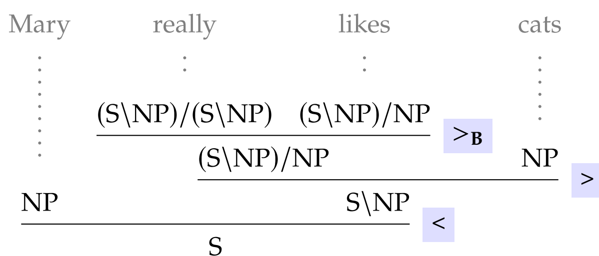


Figure 1.1: A CCG derivation tree for the sentence *Mary really likes cats*. The conventional abbreviations are provided for each rule: > forward application; < backward application; $>_\mathbf{B}$ forward harmonic composition. The term *harmonic* refers to the coinciding direction of slashes in the input categories.

1.1 Basic Concept

We will now explain the basic concept behind CCG. At the basis of the grammar is a lexicon and rule system. Given a string of input symbols, the grammar assigns a syntactic category to each of the input symbols in accordance with the lexicon. Note that this assignment is nondeterministic: There might be several categories associated with an input symbol. The rule system then repeatedly combines adjacent categories. When a (binary) derivation tree that comprises all input symbols and has its root labeled by an initial category can be obtained, the input string is accepted. Figure 1.1 depicts a derivation tree, where the symbol–category relation given by the lexicon is indicated by a dotted line.

Categories are built from atomic categories, such as NP ("noun phrase") or S ("sentence"), and slashes, which indicate directionality. The atomic category at the beginning of a category is called target and is similar to the return type of a function. After the target, a number of arguments may follow, each consisting of a slash and a category itself. The forward slash indicates that the subsequent category is expected to be provided on the right side; the backward slash indicates that it is expected on the left side. For example, a transitive verb can be modeled by the complex category (S\NP)/NP. The intended interpretation of this category is that, in order to obtain a sentence, a noun phrase on the right side (the object) and a noun phrase on the left side (the subject) have to be provided.

The rule system comprises a set of rules that specify how two adjacent categories can be combined, where the primary category is the one expecting an argument, and the secondary category is the one providing it. In the most simple type of rule, the application rule, the secondary category matches exactly with the category in the last (i.e., outermost) argument of the primary category. The combination yields an output category that follows the form of the primary category, but has its last argument removed. An example is the combination of (S\NP)/NP as a primary category and NP as a secondary category on the right side, as shown in

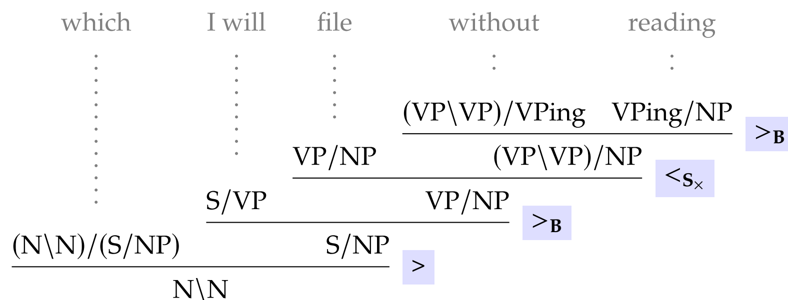


Figure 1.2: A CCG derivation tree (example from [86, page 51]) that demonstrates the combination of categories corresponding to *file* and *without reading* through backward crossed substitution ($<_{S_\times}$), where *crossed* refers to the pattern of opposing slash directions in the input categories. The derivation of S/VP from the categories associated with the words *I* and *will* is omitted.

Figure 1.1. More general are composition rules, where additional arguments may follow in the secondary category after the required category, as in the combination of (S\NP)/(S\NP) and (S\NP)/NP in Figure 1.1. The additional argument /NP gets transferred into the output category (S\NP)/NP. This is a rule of first degree, where the degree of a rule is defined as the number of additional arguments in the secondary category after the prefix that gets consumed (i.e., (S\NP) in the example). These operations are similar to function application and function composition.

Substitution is another rule type of practical relevance and related to composition. The difference is that not only the last argument of the primary category has to be provided by the secondary category, but the last two arguments, of which the first one gets removed, and the second one gets preserved in the output category. Substitution is used to model the combination of two constituents that are both followed by a gap referring to the same resource.[1] This is the case for *without reading* and *file* in Figure 1.2, which is an example taken from Steedman [86, page 51]. The occurrences of the argument /NP in the categories VP/NP and (VP\VP)/NP refer to the same resource and are collapsed through substitution. As discussed by Steedman, expressions such as *file without reading* can be used in coordinate structures and should therefore be regarded as constituents.

1: This phenomenon is called *parasitic gap* and describes a 'gap that is dependent on the existence of another gap' [17].

Another rule type that is commonly used in practical applications is type-raising [86]. However, it will not be used by the formalism we study.

1.2 Research Focus

The focus of the research presented in this thesis is twofold: We study the generative power as well as the computational power of CCG. As pointed out, there is a wide range of variants of CCG. However, for only a few of them, their generative power has been investigated. Even worse, it has become apparent that subtle changes of the formalism can have an immense effect on its expressivity. We therefore aim to bring more clarity into the relation of different variants and the relation to other mildly context-sensitive formalisms. For this, in the first part of this thesis, we take an indispensable core of the formalism and study it as a generator of formal languages. In particular, we are interested in the generated tree languages, since the underlying structure of a sentence is of major interest in natural language processing. Determining the expressive power of CCG is an important problem since a certain amount is required in order to properly model natural language. On the downside, the expressivity should also not be too high, as increasing it generally

comes at the cost of a higher parsing complexity. This leads us to the second focus of this thesis: computational power. Several features of CCG have been discussed as possible contributors to the high complexity of CCG parsing, which is known to be computationally demanding. However, it remained unanswered if omitting or restricting some of these features would actually allow for more efficient parsing. This question is studied in the second part of this thesis. The goal is to identify grammar features that lead to desirable properties in both aspects: an expressivity appropriate for modeling natural language, and a good parsing complexity to render it practically applicable. In the following, we survey the existing research in these two areas in detail and describe our contributions to them.

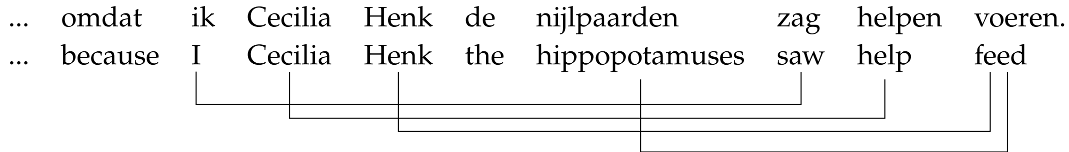
1.3 Generative Power

The generative power refers to the question what sets of structures a formalism is able to express. Other terms commonly used are expressive power, expressivity, or generative capacity. We will use them interchangeably.

1.3.1 Weak Generative Power

The weak generative power, or string-generative capacity, of a formalism is its ability to express a language, i.e., a set of strings. It is thus defined as the class of languages that it can generate. A classical result is that categorial grammar, which has only application rules, can generate exactly the context-free languages [8]. The string-generative capacity does not increase when composition of first degree is included [19, 49]. One of the best-known results on the mildly context-sensitive formalisms, which are more expressive, is the weak equivalence of CCG, TAG [38], linear indexed grammar (LIG) [35], and head grammar (HG), famously shown by Vijay-Shanker and Weir [93].[2] An automaton model with the same expressive power is the embedded push-down automaton [90]. However, the equivalence result due to Vijay-Shanker and Weir [93] crucially depends on two key features of their CCG: rule restrictions and ε-entries. The first allow the inclusion or exclusion of certain rules depending on the form of the input categories. The second are lexicon entries that assign syntactic categories to the empty word. In contrast, a CCG is called pure if it allows all possible rules up to some fixed rule degree: Then it is a fully lexicalized formalism, relying on a universal set of rules and putting all derivational control into the lexicon. The question

2: Note that Vijay-Shanker and Weir [93] and the other referenced works in general do not take into account substitution rules. If substitution is regarded, we will state so explicitly in the following.



'... because I saw Cecilia help Henk feed the hippopotamuses.'

Figure 1.3: Cross-serial dependencies in Dutch (example from [84]).

if lexicalization is possible without any sacrifices in terms of expressivity is a particular interesting one, as this model is favored by modern variants of CCG. Having said that, it has been shown that pure CCG is actually strictly less expressive than TAG [49, 50]. More precisely, it offers less control over the derivation, since the possibility of a rearrangement of derivation trees causes each generated language to contain a Parikh-equivalent[3] context-free language as a subset. For example, the (non-context-free) language consisting exactly of the strings of the form $a^n b^n c^n$ cannot be generated because other permutations of the input symbols would be generated as well. This renders the formalisms inappropriate for expressing the cross-serial word order in Dutch subordinate clauses [84] (see Figure 1.3) since also other word orders would be generated. This applies not only to pure CCG, but already to the more general prefix-closed[4] CCG without target restrictions.[5] If these conditions apply, also the expressivity of multi-modal CCG is impaired in spite of additional slash types promising further derivational control. The target restrictions play a critical role here, as prefix-closed CCG with target restrictions has the same weak generative capacity as TAG. On the other hand, the expressive power of CCG can be increased by allowing generalized composition rules of unbounded degree [97, Section 5.2]. However, it is generally accepted that substitution rules do not increase the expressive power: Steedman [86, page 210] sketches how to adapt the equivalence proof of Vijay-Shanker and Weir [93] to take substitution rules into account.

3: Two languages are called *Parikh-equivalent* if for each word in one of the languages there is a permutation of it (possibly the word itself) in the other language.

4: A CCG is called *prefix-closed* if for all valid rules also all rules of the same form, but with some (or all) arguments removed from the respective secondary category, are part of the rule set. Therefore, if the rule set allows $\frac{S/VP \quad VP/NP}{S/NP}$, then $\frac{S/VP \quad VP}{S}$ is permitted as well [50, Definition 2].

5: *Target restrictions*, a type of rule restrictions, allow to restrict a rule such that it can only be applied to primary categories with specific targets.

1.3.2 Strong Generative Power

Strong generative power describes the expressivity of a formalism considering not only the sequence of input symbols, but also some kind of additional structure that explains the relation between them. There exist several different definitions of this term. The basic distinction is between *dependency* and *constituency*, which are both fundamental and well-established conceptions of syntax. Dependency [64, page 101] describes the relation between the words of an input sentence. In CCG, when a lexical category provides an argument for another lexical category, the input symbol

Table 1.1: Overview of the weak and strong generative capacity of CCG. It is related to the following language classes: context-free languages (CFL), regular tree languages (RTL), and tree-adjoining languages (TAL). If not specified, rule restrictions are allowed and the rule system uses only composition rules. The references indicated in the first row investigate classical categorial grammar, but the results for CCG with rule restrictions can easily be concluded (see also Theorem 4.1.9). Parentheses contain the number of the respective theorem or corollary in this dissertation. If no reference is given, a result follows directly through combination of a result from the literature with a new result (see page 138).

CCG variant	rule degree	ε-entries	string languages	tree languages
(pure) with application rules only	$k = 0$	yes/no	= CFL [8]	$\subsetneq$ RTL [10] (4.1.9)
pure with composition	$k = 1$	yes/no	= CFL (4.2.8)	$\subsetneq$ RTL (5.2.2)
composition	$k = 1$	yes/no	= CFL [19, 49]	= RTL (4.2.6)
pure with composition	$k \geq 2$	yes/no	$\subsetneq$ TAL [49, 50]	$\subsetneq$ TAL (5.2.2)
prefix-closed, no target restrictions	$k \geq 2$	yes/no	$\subsetneq$ TAL [50]	$\subsetneq$ TAL
prefix-closed	$k \geq 2$	yes	= TAL [50]	
composition	$k \geq 2$	no	= TAL (5.7.3)	= TAL (5.7.2)
composition	$k \geq 2$	yes	= TAL [93]	= TAL (5.7.2)
composition and substitution	$k \geq 2$	yes	= TAL [86]	
generalized composition	unlimited	no	$\supsetneq$ TAL	$\supsetneq$ TAL
generalized composition	unlimited	yes	$\supsetneq$ TAL [97]	$\supsetneq$ TAL

associated with the first becomes a dependant of the input symbol associated with the latter. Constituency [64, page 93] describes how a sentence is subdivided into its constituents, which are certain groups of words that behave as a unit, for example noun phrases. The derivation trees of CCG are already based on the idea of constituency. Therefore, they and also relabeled derivation trees can be regarded as constituency trees.

Several works have investigated the strong generative power of CCG in terms of dependency, often with a focus on the comparison to TAG. To obtain dependencies in a straightforward way, one usually considers only lexicalized TAG (LTAG), where each elementary tree of the grammar is associated with a lexical anchor; then a dependency between these is created when an elementary tree is adjoined into another.[6] Note that LTAG has been shown to be strictly less expressive than TAG when the sets of generated trees are considered [53]. Hockenmaier and Young [31] analyze and compare the sets of dependencies expressible by LTAG and a CCG variant with type-raising. They demonstrate that there exist certain scrambling cases that cannot be expressed by the first, but can be expressed by the latter. In a similar direction, Koller and Kuhlmann [47] examine the sets of dependency trees that can be generated by CCG and LTAG, proving that these are incomparable. Stanojević and Steedman [83] study particular expressions on the basis of the “natural order of dominance” of their constituents. They show that CCG can express exactly the separable permutations of this order. A permutation is called separable if there exists a binary tree such that, when using the permutation to label the leaves of

6: For a short introduction to TAG, see Section 2.3.2.

the tree from left to right, the leaves in each subtree are labeled by a set of consecutive elements of the original order. In contrast, TAG can also express non-separable permutations. As an example, they give that from Koller and Kuhlmann [47, Figure 8b], which was used to separate the classes of dependency trees generatable by CCG and LTAG. It is noteworthy that Stanojević and Steedman [83] also draw a connection to the permutations that have been documented in natural language. While for each separable permutation of the expressions they study there exists evidence in some natural language, this is not the case for any of the non-separable permutations.

The target of our investigations on the strong generative power of CCG are tree languages in the sense of constituency trees. Analogous to string languages, a tree language is a set of trees; the term forest is commonly used as well. An overview on tree languages can be found in [22]. When comparing the strong generative power of two formalisms, we sometimes consider strong equivalence modulo relabeling. An example for two formalisms that are strongly equivalent modulo relabeling are local and regular tree grammars [22]. We define the tree languages generated by CCG as relabeled derivation tree sets. Already Weir and Joshi [97] asked what set of derivation trees CCG can generate. Since CCG derivation trees are always binary, this is the case for the generated tree languages as well. Therefore, we will consider only binary trees throughout this thesis. This conception of strong generative capacity has been studied by Tiede [89] for Lambek grammars [58], but has not yet been addressed for CCG. The only known result in this direction is the characterization of tree languages generated by categorial grammar due to Buszkowski [10, Theorem 1.1]. These tree languages form a proper subset of regular tree languages, containing exactly those where for each node the length of the shortest path to a leaf is bounded.[7]

7: Buszkowski [10] uses a grammar with a multiple-argument format, which can generate trees with node ranks higher than 2. It can be restricted to obtain the analogous result for binary trees [11, page 699].

1.3.3 Contributions

In this thesis, we study the tree-generative capacity of a CCG formalism that is based on that of Vijay-Shanker and Weir [93] and relate its generated tree languages to other well-established classes of tree languages, in particular regular tree languages [22] and context-free tree languages [72]. Our CCG has a finite set of application and composition rules; rule restrictions are permitted in general, but we also study the effect of their absence. While Vijay-Shanker and Weir [93] heavily relied on ε-entries, it remained unclear whether they can be avoided without compromising the generative power of CCG. This is particularly interesting because ε-entries violate the Principle of Adjacency [86, page 54], one of the

fundamental principles at the basis of CCG.[8] Another important aspect is that ε-entries are computationally problematic [55], as will be discussed further on.

8: The Principle of Adjaceny demands that all combining categories have phonologically realized counterparts in the input and are string-adjacent.

We provide a new proof of the abovementioned characterization of the tree languages generated by categorial grammar as a subset of regular tree languages [10, Theorem 1.1], or more general, we characterize the tree languages generated by CCG with application rules, since we also allow rule restrictions. This, however, has no effect on the tree-generative capacity. Then, we show that CCG with composition rules of first degree can generate exactly the regular tree languages.

The main result is the strong equivalence of CCG and TAG when understood as generators of tree languages. The proof is not directly using TAG, but the strongly equivalent (ε-free)[9] simple monadic context-free tree grammar (sCFTG) [45], where simple means that it is linear and nondeleting, and monadic means that the nonterminals are nullary or unary nodes. For the first inclusion, we show that the set of rule trees of a CCG can be generated by an sCFTG. In rule trees, the internal nodes are labeled by rules, and only the leaves are labeled by categories. This concept was first introduced by Weir and Joshi [97] to keep the set of node labels finite. The inclusion in the tree-adjoining languages (TAL) is proper for pure CCG, which cannot even generate all local tree languages. For the second inclusion, we show that each spine grammar [21] can be simulated by a CCG. Spine grammars are a restriction on sCFTG that is still strongly equivalent modulo relabeling. The construction uses rules of degree at most 2 and only first-order categories, thus all arguments contain atomic categories. This proves that CCG with these restrictions already has the full expressive power. A worthwhile result of the constructions is an effective procedure for removing ε-entries from a CCG. Finally, another interesting consequence of the equivalence result is the strong equivalence of CCG and linear top-down push-down tree automata [21]. The properties of the language class TAL along with further representations are discussed in Section 2.4. Table 1.1 summarizes the discussed results on the generative power of CCG, both from the literature and shown in this dissertation.

9: By ε-free, we mean that the empty word is not part of the generated language.

1.4 Computational Power

The computational power of a formalism refers to its parsing complexity. This in turn can point to two different problems: the membership problem and the universal recognition problem. Both problems consider the question whether a given string is part of the language generated by some grammar or other formalism.

In the membership problem, the grammar is fixed and one is interested only in the complexity depending on the length of the input string. In the universal recognition problem, on the other hand, one is additionally interested in the complexity in terms of grammar size. Polynomial parsing is one of the defining properties of the mildly context-sensitive language class [37]. This refers to the membership problem; it can be solved in polynomial time for CCG as shown by Vijay-Shanker and Weir [91, 92]. Kuhlmann and Satta [54] present a simpler algorithm for CCG parsing and analyze its runtime depending on the grammar size. It depends on several properties of the grammar, including the number of lexical arguments, the maximum degree of rules, and the maximum number of arguments of lexical categories. Thus, although the algorithm itself is straightforward, its runtime complexity is quite involved. Kuhlmann, Satta, and Jonsson [55] demonstrate that the runtime complexity of the universal recognition problem for CCG is in sharp contrast to the complexity of the membership problem: They show that the universal recognition problem is NP-complete when ε-entries are excluded, and even EXPTIME-complete in the presence of ε-entries. Thus, the problem cannot be solved in time polynomial in the grammar size in the worst-case scenario (assuming $P \neq NP$ in the first case). When unbounded generalized composition rules and ε-entries are included, the universal recognition problem even becomes Turing-complete [55, page 478]. The high complexity of the universal recognition problem sets CCG apart from TAG, where the problem can be solved in polynomial time [74]. Kuhlmann, Satta, and Jonsson [55] discuss a number of properties of CCG that were essential for the proof of their EXPTIME-completeness result. These properties are the availability of rule restrictions, the use of variables in secondary categories when specifying rules, the absence of a fixed bound on the maximum degree of rules,[10] and the lexical ambiguity, which allows nondeterministic assignment of categories to input symbols. However, it was not apparent if omitting or restricting any of these features would lead to a formalism with a parsing complexity polynomial in the grammar size.

10: Note that due to the CCG rule set being finite, each individual grammar has some individual maximum rule degree. (An exception are the already mentioned unbounded generalized composition rules that are usually forbidden.) The bound that Kuhlmann, Satta, and Jonsson [55] refer to, however, is a *universal* bound on the maximum degree that would hold for all possible CCGs.

1.4.1 Contributions

Our main result in this direction of research is that the universal recognition problem can be solved in polynomial time and space if the rule degree of the CCG is bounded. For this, we present a new parsing algorithm for CCG that improves and extends the algorithm of Kuhlmann and Satta [54] and leads to a considerably simpler complexity analysis. It has the notable feature that the runtime is exponential exclusively in the maximum rule degree of the grammar. Thus, bounding the rule degree leads to an algorithm

Table 1.2: Overview of the computational complexity of the universal recognition problem for several variants of CCG with rule restrictions. The first two rows are completeness-results. The third is containment in PTIME in the general case (Theorem 6.3.1), and PTIME-completeness under logspace-reduction if ε-entries are allowed (Corollary 6.3.3).

CCG variant	complexity
with ε-entries	EXPTIME [55]
without ε-entries	NP [55]
bounded rule degree	PTIME (Theorem 6.3.1, Corollary 6.3.3)

polynomial in the grammar size. This is particularly interesting because a bound as low as degree 2 is sufficient for full generative power with regard to string languages [93] and tree languages (see above). Also in linguistic work a similarly low bound has been presumed as a syntactic universal; for English, Steedman [86, page 43] suggests a maximum rule degree of 3 as sufficient. We also show that, if ε-entries are included, the universal recognition problem for CCG of bounded degree is PTIME-complete under logspace-reduction.

The second novel feature of our algorithm is that it can handle substitution rules, which are of practical relevance, but have been neglected in theoretic work on CCG so far. This is an important contribution since the effect of substitution rules on the parsing complexity was not clear, and seemingly trifle changes of the formalism can result in substantial differences in terms of complexity, as can be seen from the effect of ε-entries.

We propose several extensions of our algorithm. First, we discuss how to implement rule restrictions and how to adapt the algorithm to multi-modal CCG. Second, we describe how a small change of the algorithm makes it polynomial in the grammar size if all secondary categories in the rule set are instantiated, i.e., not containing any variables. This demonstrates that the availability of variables in secondary categories is indeed indispensable for the EXPTIME-completeness result of Kuhlmann, Satta, and Jonsson [55]. Finally, we present our ideas on how to remove spurious ambiguity[11] from the algorithm.

11: A parsing algorithm has spurious ambiguity if the same derivation tree can be produced by several different parse trees (see Section 6.5.1).

The discussed results on the computational complexity of the universal recognition problem for CCG with rule restrictions are presented in Table 1.2. Although Kuhlmann, Satta, and Jonsson [55] regarded CCG with only composition rules, the validity of their results is not affected by the inclusion of substitution rules.

1.5 Overview of the Dissertation

We will briefly outline the structure of this dissertation and indicate which articles the respective chapters are based on. Chapter 2 introduces notations and definitions of standard formalisms for string and tree languages. It also gives an overview over the class of mildly context-sensitive languages. Chapter 3 formally introduces the CCG formalism that is used throughout this dissertation. If any assumptions are made that restrict the considered CCG, this will be stated at the beginning of the respective chapters or sections. These two chapters both partly draw on contents of the publications listed below. Chapters 4 and 5 are closely related since they both discuss the generative power of CCG. They constitute the first part of contributions of this dissertation. Chapter 4 investigates CCG with low rule degrees (i.e., rule degrees 0 and 1) and characterizes the tree languages they can generate. It is based on joint work with Marco Kuhlmann and Andreas Maletti, published in the following two articles.

Marco Kuhlmann, Andreas Maletti, and Lena K. Schiffer. 'The Tree-Generative Capacity of Combinatory Categorial Grammars'. In: *Proceedings of the 39th Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*. Ed. by A. Chattopadhyay and P. Gastin. Vol. 150. LIPIcs. Schloss Dagstuhl — Leibniz-Zentrum für Informatik, 2019

Marco Kuhlmann, Andreas Maletti, and Lena K. Schiffer. 'The Tree-Generative Capacity of Combinatory Categorial Grammars'. In: *Journal of Computer and System Sciences* 124 (2022). Extended version.

Chapter 5 gives a characterization of the tree languages generatable by CCG with an arbitrary rule degree by proving strong equivalence with TAG. Sections 5.1 and 5.2 are based on the two articles listed above. They show the first inclusion of the equivalence and discuss pure CCG. The remaining sections of Chapter 5 are dedicated to the second inclusion and are based on the following joint work with Andreas Maletti.

Lena K. Schiffer and Andreas Maletti. 'Strong Equivalence of TAG and CCG'. In: *Transactions of the Association for Computational Linguistics (TACL)* 9 (2021)

Andreas Maletti and Lena K. Schiffer. 'Strong Equivalence of TAG and CCG'. Extended version. Unpublished manuscript. 2022. arXiv: `2205.07743 [cs.FL]`

Chapter 6 constitutes the second part of contributions and provides insights into the computational power of CCG. It is based on the joint work with Marco Kuhlmann and Giorgio Satta published in the following article.

Lena K. Schiffer, Marco Kuhlmann, and Giorgio Satta. 'Tractable Parsing for CCGs of Bounded Degree'. In: *Comput. Linguist.* 48.3 (2022)

Finally, Chapter 7 concludes the dissertation with a summary, a discussion of the results, and an outlook to possible future work.

Preliminaries | 2

2.1 Basic Definitions . . . 13
2.2 String Languages . . . 13
2.3 Tree Languages 16
2.4 Mild Context-Sensitivity 20

This chapter introduces the notation and several standard formalisms for string and tree languages. For string languages, we recall nondeterministic finite automata, context-free grammars, and push-down automata; for tree languages, we recall simple monadic context-free tree grammar, regular tree grammar, and tree-adjoining grammar. The chapter concludes with an introduction to mild context-sensitivity.

2.1 Basic Definitions

The nonnegative integers are $\mathbb{N}$ and the positive integers are $\mathbb{N}_+$. For every $k \in \mathbb{N}$ let $[k] = \{i \in \mathbb{N} \mid 1 \leq i \leq k\}$ as well as $\mathbb{Z}_k = \{i \in \mathbb{N} \mid i < k\}$. The power-set (i.e., set of all subsets) of a set A is $\mathcal{P}(A) = \{A' \mid A' \subseteq A\}$, and $\mathcal{P}_+(A) = \mathcal{P}(A) \setminus \{\emptyset\}$ contains all nonempty subsets. As usual, $\pi_i \colon X_1 \times \cdots \times X_n \to X_i$ projects a tuple to its i-th component and is given by $\pi_i\big(\langle x_1, \ldots, x_n\rangle\big) = x_i$, where each X_j with $j \in [n]$ is a set.

Given two sets A and A', a relation from A to A' is a subset $\rho \subseteq A \times A'$. The inverse of ρ is $\rho^{-1} = \big\{(a', a) \mid (a, a') \in \rho\big\}$, and for every $B \subseteq A$, we let $\rho(B) = \big\{a' \mid \exists b \in B \colon (b, a') \in \rho\big\}$. The relation $\rho \subseteq A \times A'$ can also be understood as a mapping $\widehat{\rho} \colon A \to \mathcal{P}(A')$ with $\widehat{\rho}(a) = \rho(\{a\})$ for all $a \in A$. We will not distinguish these two representations. Given a relation $\Rightarrow \subseteq A^2$, we let $\Rightarrow^*$ be the reflexive, transitive closure of $\Rightarrow$. Further, we let $\Rightarrow^+$ be the transitive closure of $\Rightarrow$.

2.2 String Languages

As usual, an alphabet is a finite set of symbols. The monoid $(\Sigma^*, \cdot, \varepsilon)$ consists of all strings (i.e., sequences) over a (possibly infinite) set Σ, together with concatenation $\cdot$ and the empty string ε. We let $\Sigma^+ = \Sigma^* \setminus \{\varepsilon\}$. We often write concatenation by juxtaposition. The length of a string $w \in \Sigma^*$ (i.e., the number of components in the sequence) is denoted by $|w|$. The *prefixes* $\mathrm{Pref}(w)$ of a string $w \in \Sigma^*$ are $\{u \in \Sigma^* \mid \exists v \in \Sigma^* \colon w = uv\}$. Any set $\mathcal{L} \subseteq \Sigma^*$ is a (string) language, and the languages form a monoid $\big(\mathcal{P}(\Sigma^*), \cdot, \{\varepsilon\}\big)$ with concatenation lifted to languages by $\mathcal{L} \cdot \mathcal{L}' = \{w \cdot w' \mid w \in \mathcal{L}, w' \in \mathcal{L}'\}$.

Every mapping $f\colon \Sigma \to \Delta^*$ [respectively, $f\colon \Sigma \to \mathcal{P}(\Delta^*)$] extends uniquely to a monoid homomorphism $f'\colon \Sigma^* \to \Delta^*$ [respectively, $f'\colon \Sigma^* \to \mathcal{P}(\Delta^*)$]. We will not distinguish the mapping f and its induced homomorphism f', but rather use f for both.

In the following, we briefly recall three standard formalisms for representing string languages.

2.2.1 Nondeterministic Finite Automata

We start with nondeterministic finite automata [35].

Definition 2.2.1 *A* nondeterministic finite automaton *(NFA) $\mathcal{A} = (Q, \Sigma, \delta, I, F)$ is a tuple consisting of*

(i) finite sets Q and Σ of states *and* input symbols, *respectively,*
(ii) a transition relation $\delta \subseteq Q \times \Sigma \times Q$, *and*
(iii) sets $I, F \subseteq Q$ of initial *and* final states, *respectively.*

The transition relation can be extended to a function taking a string as an input. It is denoted by $\hat{\delta}\colon Q \times \Sigma^* \to \mathcal{P}(Q)$ and is given by

$$\hat{\delta}(q, \varepsilon) = \{q\} \qquad\qquad \hat{\delta}(q, aw) = \bigcup_{(q,a,p)\in\delta} \hat{\delta}(p, w)$$

for all $q \in Q$, $a \in \Sigma$, and $w \in \Sigma^*$. The language *accepted* by a given NFA $\mathcal{A}$ is defined as

$$\mathcal{L}(\mathcal{A}) = \bigcup_{q\in I} \left\{ w \in \Sigma^* \mid \hat{\delta}(q, w) \cap F \neq \emptyset \right\} .$$

Given string language $\mathcal{L}$, if there exists an NFA $\mathcal{A}$ with $\mathcal{L}(\mathcal{A}) = \mathcal{L}$, then we call $\mathcal{L}$ *regular*.

2.2.2 Context-Free Grammar

Next, let us recall context-free grammars [9].

Definition 2.2.2 *A* context-free grammar *(CFG) $\mathcal{G} = (N, \Sigma, S, P)$ consists of*

(i) disjoint finite sets N and Σ of nonterminal *and* terminal symbols, *respectively,*
(ii) a set $S \subseteq N$ of start nonterminals, *and*
(iii) a finite set $P \subseteq N \times (N \cup \Sigma)^*$ of productions.

In the following let $\mathcal{G} = (N, \Sigma, S, P)$ be a CFG. We write productions (n, r) as $n \to r$. Productions of the form $n \to \varepsilon$ are called *ε-productions*. However, we will usually consider CFG without

ε-productions. Given $n \to r \in P$ and $u, v \in (N \cup \Sigma)^*$, we write $unv \Rightarrow_{\mathcal{G}} urv$ and say that unv *derives* urv. The language *generated* by $\mathcal{G}$ is $\mathcal{L}(\mathcal{G}) = \{w \in \Sigma^* \mid \exists s \in S \colon s \Rightarrow^*_{\mathcal{G}} w\}$. Given a string language $\mathcal{L}$, if there exists a CFG $\mathcal{G}$ with $\mathcal{L}(\mathcal{G}) = \mathcal{L}$, then we call $\mathcal{L}$ *context-free*. The *derivation trees* of $\mathcal{G}$ are inductively defined as the smallest set $\mathcal{D}(\mathcal{G})$ such that (i) $\Sigma \subseteq \mathcal{D}(\mathcal{G})$, (ii) for all $n \to \varepsilon \in P$ we have $n('\varepsilon') \in \mathcal{D}(\mathcal{G})$, and (iii) for all $n \to r_1 \dots r_m \in P$ with $r_i \in \Sigma \cup N$ and for all $t_i \in \{t \in \mathcal{D}(\mathcal{G}) \mid t(\varepsilon) = r_i\}$ for $i \in [m]$ we have $n(t_1, \dots, t_m) \in \mathcal{D}(\mathcal{G})$.[1]

1: We use the standard bracket notation and address system for trees, i.e., $n(t_1, \dots, t_m)$ denotes the tree with root label n and subtrees $t_1, \dots, t_m$, and $t(\varepsilon)$ refers to the root label of tree $t \in \mathcal{D}(\mathcal{G})$ (see Section 2.3). For disambiguation, we write $'\varepsilon'$ when the empty word is used as a symbol inside a tree or lexicon entry.

2.2.3 Push-Down Automata

Finally, let us recall push-down automata [3], which accept the (ε-free) context-free languages. For any alphabet Σ and the special symbol $\perp \notin \Sigma$, we let $\Sigma_\perp = \Sigma \cup \{\perp\}$. Moreover, we let $\Sigma^{\leq 1}$ be the set of strings of length at most 1 over the alphabet Σ; i.e., $\Sigma^{\leq 1} = \{\varepsilon\} \cup \Sigma$.

Definition 2.2.3 *A* push-down automaton *(PDA) is defined as a tuple* $\mathcal{A} = (Q, \Sigma, \Gamma, \delta, I, \perp, F)$ *consisting of*

(i) *finite sets* Q, Σ, *and* Γ *of* states, input symbols, *and* stack symbols, *respectively,*
(ii) *a set* $\delta \subseteq (Q \times \Sigma \times \Gamma_\perp^{\leq 1} \times \Gamma^{\leq 1} \times Q) \setminus (Q \times \Sigma \times \Gamma_\perp \times \Gamma \times Q)$ *of* transitions,
(iii) *sets* $I, F \subseteq Q$ *of* initial *and* final states, *respectively, and*
(iv) *an* initial stack symbol $\perp \notin \Gamma$.

Given a PDA $\mathcal{A} = (Q, \Sigma, \Gamma, \delta, I, \perp, F)$, let $\mathrm{Conf}_{\mathcal{A}} = Q \times \Sigma^* \times \Gamma_\perp^*$ be the set of *configurations*. Intuitively speaking, in configuration $\langle q, w, \alpha \rangle \in \mathrm{Conf}_{\mathcal{A}}$ the PDA $\mathcal{A}$ is in state q with stack contents α and still has to read the input string w. The *move relation* $\vdash_{\mathcal{A}} \subseteq \mathrm{Conf}_{\mathcal{A}}^2$ is defined as follows:

$$\vdash_{\mathcal{A}} = \bigcup_{\substack{(q,a,\gamma,\gamma',q') \in \delta, \\ w \in \Sigma^*,\, \alpha \in \Gamma_\perp^*}} \left\{ \big(\langle q, aw, \gamma\alpha \rangle, \langle q', w, \gamma'\alpha \rangle\big) \in \mathrm{Conf}_{\mathcal{A}}^2 \mid \gamma\alpha \neq \varepsilon \right\} .$$

The configuration $\langle q, w, \alpha \rangle$ is *initial* [respectively, *final*] if $q \in I$, $w \in \Sigma^+$, and $\alpha = \perp$ [respectively, $q \in F$, $w = \varepsilon$, and $\alpha = \varepsilon$]. An *accepting run* is a sequence $\xi_0, \dots, \xi_n \in \mathrm{Conf}_{\mathcal{A}}$ of configurations that are successively related by moves (i.e., $\xi_{i-1} \vdash_{\mathcal{A}} \xi_i$ for all $i \in [n]$), that starts with an initial configuration ξ_0, and finishes in a final configuration ξ_n. In other words, we start in an initial state with $\perp$ on the stack and finish in a final state with the empty stack, and for each intermediate step there exists a transition. An input string $w \in \Sigma^+$ is *accepted* by $\mathcal{A}$ if there exists an accepting run

starting in $\langle q, w, \bot\rangle$ with $q \in I$. The language $\mathcal{L}(\mathcal{A})$ *accepted* by the PDA $\mathcal{A}$ is the set of accepted input strings and thus given by

$$\mathcal{L}(\mathcal{A}) = \bigcup_{(q,q')\in I\times F} \left\{w \in \Sigma^+ \mid \langle q, w, \bot\rangle \vdash_{\mathcal{A}}^* \langle q', \varepsilon, \varepsilon\rangle\right\} \ .$$

Note that our PDA are ε-free (in the sense that each transition induces moves that process an input symbol) and have limited stack access: In each move we can pop a symbol, push a symbol, or ignore the stack. We explicitly exclude the case in which a symbol is popped and another symbol is pushed at the same time. However, this restriction has no influence on the expressive power (see [15, Corollary 12] for the weighted scenario; an instantiation of the result with the Boolean semiring yields the unweighted case). Note that no moves are possible anymore once the stack is empty.

2.3 Tree Languages

In this paper, we only deal with binary trees since the derivation trees of CCGs are binary. Thus, we build trees over *ranked sets* $\Sigma = \Sigma_0 \cup \Sigma_1 \cup \Sigma_2$, consisting of the set Σ_0 of leaf symbols, the set Σ_1 of unary internal symbols, and the set Σ_2 of binary internal symbols. If Σ is an alphabet, then it is a *ranked alphabet*. For every $k \in \{0, 1, 2\}$, we say that symbol $a \in \Sigma_k$ has *rank* k.[2] We write $T_{\Sigma_2,\Sigma_1}(\Sigma_0)$ for the set of all trees over Σ, which is the smallest set T such that $c(t_1, \ldots, t_k) \in T$ for all $k \in \{0, 1, 2\}$, $c \in \Sigma_k$, and $t_1, \ldots, t_k \in T$. As usual, we write just a for leaves $a()$ with $a \in \Sigma_0$. We use graphical representations of trees to increase readability. A *tree language* is a subset $\mathcal{T} \subseteq T_{\Sigma_2,\emptyset}(\Sigma_0)$. Let $T = T_{\Sigma_2,\Sigma_1}(\Sigma_0)$. The map $\mathrm{pos}\colon T \to \mathcal{P}_+\big([2]^*\big)$ assigns Gorn tree addresses [25] to a tree as follows. Let

2: Note that the sets Σ_0, Σ_1, and Σ_2 are not necessarily disjoint and a symbol can thus occur with different ranks. This accounts for the fact that the same categories can occur as leaves or internal nodes in CCG derivation trees.

$$\mathrm{pos}\big(c(t_1, \ldots, t_k)\big) = \{\varepsilon\} \cup \bigcup_{i\in[k]} \{iw \mid w \in \mathrm{pos}(t_i)\}$$

for all $k \in \{0, 1, 2\}$, $c \in \Sigma_k$, and $t_1, \ldots, t_k \in T$. The set of all leaf positions of t is defined as $\mathrm{leaves}(t) = \big\{w \in \mathrm{pos}(t) \mid w1 \notin \mathrm{pos}(t)\big\}$. Let $\mathrm{ht}(t) = \max_{w\in\mathrm{leaves}(t)} |w|$ be the height of the tree t. The *subtree* of t at position $w \in \mathrm{pos}(t)$ is denoted by $t|_w$, and the label of t at position w is denoted by $t(w)$. Moreover, $t[t']_w$ denotes the tree obtained from t by replacing the subtree at position w by the tree $t' \in T$. Let $\mathrm{yield}\colon T \to \Sigma_0^+$ be defined as $\mathrm{yield}(a) = a$ for all $a \in \Sigma_0$ and $\mathrm{yield}\big(b(t_1, \ldots, t_k)\big) = \mathrm{yield}(t_1) \cdots \mathrm{yield}(t_k)$ for all $k \in [2]$, $b \in \Sigma_k$, and $t_1, \ldots, t_k \in T$.

The special leaf symbol $\square$ is reserved and represents a hole in a tree. The set $C_{\Sigma_2,\Sigma_1}(\Sigma_0)$ of *contexts* contains all trees of $T_{\Sigma_2,\Sigma_1}\big(\Sigma_0 \cup \{\square\}\big)$ in which the special symbol $\square$ occurs exactly once. Let $C \in C_{\Sigma_2,\Sigma_1}(\Sigma_0)$.

We write $\mathrm{pos}_\square(C)$ to denote the unique position of $\square$ in C, or in other words, $w \in \mathrm{pos}(C)$ with $C(w) = \square$. Moreover, given $t \in T \cup C_{\Sigma_2,\Sigma_1}(\Sigma_0)$ we simply write $C[t]$ instead of $C[t]_{\mathrm{pos}_\square(C)}$. Let $\mathrm{yield}\colon C_{\Sigma_2,\Sigma_1}(\Sigma_0) \to \Sigma_0^* \times \Sigma_0^*$ be given by $\mathrm{yield}(\square) = (\varepsilon, \varepsilon)$ and

$$\begin{aligned}&\mathrm{yield}\big(b(t_1,\dots,t_{i-1},C,t_{i+1},\dots,t_k)\big) = \\ &\qquad\big(\mathrm{yield}(t_1)\cdots\mathrm{yield}(t_{i-1})\cdot l,\ r\cdot\mathrm{yield}(t_{i+1})\cdots\mathrm{yield}(t_k)\big)\\ &\text{with } (l,r) = \mathrm{yield}(C)\end{aligned}$$

for all $k \in [2], b \in \Sigma_k, t_1,\dots,t_{i-1},t_{i+1},\dots,t_k \in T, C \in C_{\Sigma_2,\Sigma_1}(\Sigma_0)$.

A *relabeling* is a mapping $\rho\colon \Sigma \to \mathcal{P}_+(\Delta)$ for some ranked alphabet Δ.[3] It induces a mapping $\widehat{\rho}\colon T \to \mathcal{P}_+\big(T_{\Delta_2,\Delta_1}(\Delta_0)\big)$ that is defined for every $t \in T$ by

$$\begin{aligned}\widehat{\rho}(t) = \big\{u \in T_{\Delta_2,\Delta_1}(\Delta_0) \mid {}&\mathrm{pos}(u) = \mathrm{pos}(t),\\ &\forall w \in \mathrm{pos}(u)\colon u(w) \in \rho\big(t(w)\big)\big\}\ .\end{aligned}$$

3: We require that each input symbol can be relabeled.

A relabeling is therefore in general nondeterministic since it can map one symbol of Σ to several symbols of Δ. If $|\rho(\sigma)| = 1$ for all $\sigma \in \Sigma$, a relabeling is called *deterministic* and we also write $\rho\colon \Sigma \to \Delta$ and $\widehat{\rho}\colon T \to T_{\Delta_2,\Delta_1}(\Delta_0)$. In the following, we do not distinguish between the relabeling ρ and its induced mapping $\widehat{\rho}$ on trees.

2.3.1 Tree Grammars

Now we move on to representations of tree languages. We first recall context-free tree grammars (CFTG) [72], but only the monadic simple variant [45]. The grammar is called *monadic* because there are only nullary and unary nonterminals; *simple* means that the productions are linear and nondeleting, i.e., if the left side of a production is a unary nonterminal, the special symbol $\square$ (marking the new position of the subtree of the nonterminal) appears exactly once on the right side.[4]

4: Omitting the requirement that the productions are nondeleting has no effect on the expressivity [20].

Definition 2.3.1 *A* simple monadic context-free tree grammar *(sCFTG) is a tuple $\mathcal{G} = (N, \Sigma, S, P)$ consisting of*

(i) disjoint ranked alphabets N and Σ of nonterminal *and* terminal symbols *with $N = N_1 \cup N_0$ and $\Sigma_1 = \emptyset$,*
(ii) a set $S \subseteq N_0$ of nullary start nonterminals, *and*
(iii) a finite set $P \subseteq P_0 \cup P_1$ of productions, *which are taken from $P_0 = N_0 \times T_{\Sigma_2,N_1}(N_0 \cup \Sigma_0)$ and $P_1 = N_1 \times C_{\Sigma_2,N_1}(N_0 \cup \Sigma_0)$.*

If $N_1 = \emptyset$, then $\mathcal{G}$ is a regular tree grammar *(RTG).*

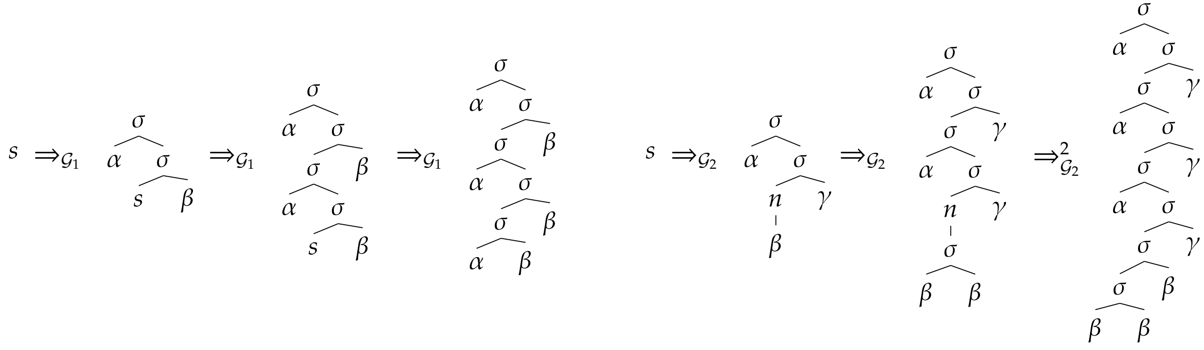


Figure 2.1: Derivations using the RTG $\mathcal{G}_1$ (left) and the sCFTG $\mathcal{G}_2$ (right) of Examples 2.3.2 and 2.3.3, respectively.

We write $(n, r) \in P$ as $n \longrightarrow r$. Next, we define the rewrite semantics [4] for the sCFTG $\mathcal{G} = (N, \Sigma, S, P)$. Given $t, u \in T_{\Sigma_2, N_1}(\Sigma_0 \cup N_0)$ and position $w \in \text{pos}(t)$, we let $t \Rightarrow_{\mathcal{G},w} u$ if there exist a production $(n \longrightarrow r) \in P$ such that

(i) $t|_w = n$ and $u = t[r]_w$ with $n \in N_0$, or
(ii) $t|_w = n(t')$ and $u = t\big[r[t']\big]_w$ with $n \in N_1$ and subtree $t' \in T_{\Sigma_2, N_1}(\Sigma_0 \cup N_0)$.

We write $t \Rightarrow_{\mathcal{G}} u$ if there exists $w \in \text{pos}(t)$ such that $t \Rightarrow_{\mathcal{G},w} u$. The *tree language generated by* $\mathcal{G}$ is

$$\mathcal{T}(\mathcal{G}) = \left\{t \in T_{\Sigma_2, \emptyset}(\Sigma_0) \mid \exists s \in S \colon s \Rightarrow^+_{\mathcal{G}} t\right\} \enspace .$$

The sCFTG $\mathcal{G}'$ is *strongly equivalent* to $\mathcal{G}$ if $\mathcal{T}(\mathcal{G}) = \mathcal{T}(\mathcal{G}')$, and it is *weakly equivalent* to $\mathcal{G}$ if $\text{yield}\big(\mathcal{T}(\mathcal{G})\big) = \text{yield}\big(\mathcal{T}(\mathcal{G}')\big)$. The *string language generated by* $\mathcal{G}$ is $\mathcal{L}(\mathcal{G}) = \big\{\text{yield}(t) \mid t \in \mathcal{T}(\mathcal{G})\big\}$. The tree languages generated by sCFTGs form a subset of the context-free tree languages, and a tree language $\mathcal{T}$ is *regular* if and only if there exists an RTG $\mathcal{G}$ such that $\mathcal{T} = \mathcal{T}(\mathcal{G})$. A detailed introduction to trees and tree languages can be found in [22].

Example 2.3.2 The RTG $\mathcal{G}_1 = (N, \Sigma, S, P)$ with nonterminals $N = N_0 = S = \{s\}$, terminals $\Sigma = \Sigma_2 \cup \Sigma_0$ with $\Sigma_2 = \{\sigma\}$ and $\Sigma_0 = \{\alpha, \beta\}$, and $P = \big\{s \to \sigma\big(\alpha, \sigma(s, \beta)\big),\ s \to \sigma(\alpha, \beta)\big\}$ generates the string language $\mathcal{L}(\mathcal{G}_1) = \{\alpha^n \beta^n \mid n \geq 1\}$. Clearly, the only nonterminal s is nullary (since $\mathcal{G}_1$ is an RTG) and thus occurs only as leaf in the right-hand sides of productions, which is similar to right-linearity for CFGs.

Two important facts on regular tree languages are that they properly include the derivation tree languages of CFGs [22, Theorem 3.2.2] and that their string languages are exactly the context-free languages [22, Theorem 3.2.7].

Example 2.3.3 The sCFTG $\mathcal{G}_2 = (N, \Sigma, S, P)$ with $N = N_1 \cup N_0$ with $N_1 = \{n\}$ and $N_0 = S = \{s\}$, $\Sigma = \Sigma_2 \cup \Sigma_0$ with $\Sigma_2 = \{\sigma\}$ and $\Sigma_0 = \{\alpha, \beta, \gamma\}$, and

$$P = \Big\{ s \to \sigma\Big(\alpha, \sigma\big(n(\beta), \gamma\big)\Big), \\ n \to \sigma\Big(\alpha, \sigma\big(n(\sigma(\square, \beta)), \gamma\big)\Big),\ \ n \to \square \Big\}$$

generates the string language $\mathcal{L}(\mathcal{G}_2) = \{\alpha^n\beta^n\gamma^n \mid n \geq 1\}$. The placeholder $\square$, which indicates the new position of the subtree under the unary nonterminal symbol n, appears exactly once on the right-hand sides of the productions with left-hand side n (as required for an sCFTG).

2.3.2 Tree-Adjoining Grammar

In the following, we will give a brief introduction to TAG [38]. We refrain from giving a formal definition since we will never use TAG itself in our proofs, but will always resort to other strongly equivalent formalisms. TAG is a mildly context-sensitive grammar formalism that operates on a set of *elementary trees* of which a subset is *initial*. To generate a tree, we start with an initial tree and successively splice elementary trees into nodes using *adjunction* operations. In an adjunction, we select a node, insert a new tree there, and reinsert the original subtree below the selected node at the distinguished and specially marked *foot node* of the inserted tree. We use the *non-strict* variant of TAG, in which the root and foot labels of the inserted tree need not coincide with the label of the replaced node to perform an adjunction. To control at which nodes adjunction is allowed, each node is equipped with two types of constraints. The *selective adjunction* constraint specifies a set of trees that can be adjoined and the Boolean *obligatory adjunction* constraint specifies whether adjunction is mandatory. Only trees without obligatory adjunction constraints are part of the generated tree language.

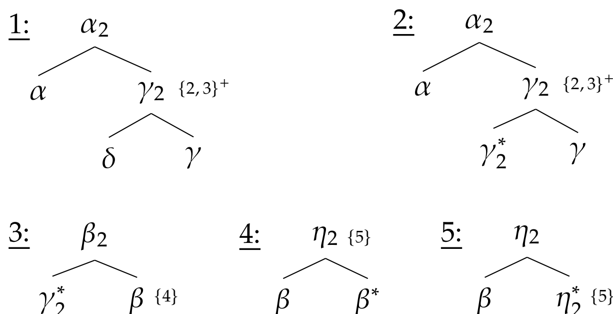


Figure 2.2: Tree-adjoining grammar $\mathcal{G}_3$ (see Example 2.3.4).

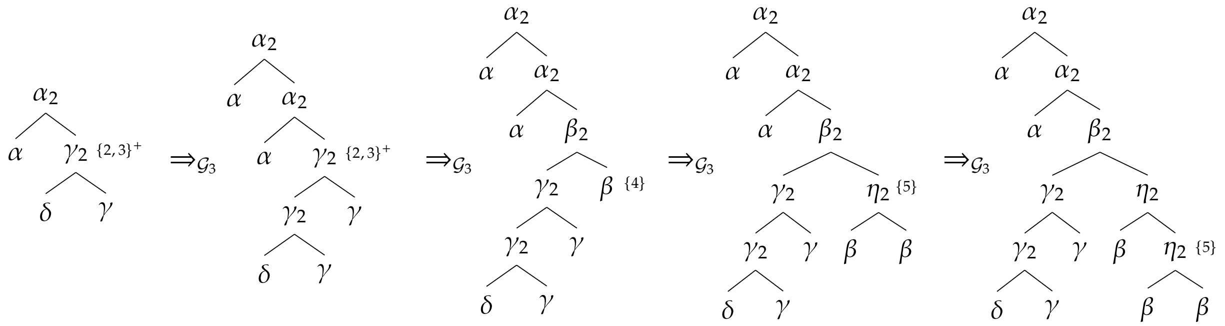


Figure 2.3: Derivation using the tree-adjoining grammar $\mathcal{G}_3$ of Figure 2.2 (see Example 2.3.4).

Example 2.3.4 Figure 2.2 shows the elementary trees of an example TAG $\mathcal{G}_3$. Only tree 1 is initial and foot nodes are marked by a superscript asterisk $\cdot^*$ on the label. Whenever adjunction is forbidden (i.e., empty set as selective adjunction constraint and non-obligatory adjunction), we omit the constraints altogether. Otherwise, the constraints are put next to the label. For example, $\{2,3\}^+$ indicates that tree 2 or 3 must (+ = obligatory) be adjoined. Figure 2.3 shows a derivation using $\mathcal{G}_3$. Note that we could stop the derivation also after two or three steps since no obligatory adjunction constraints are present in the respective trees.

2.4 Mild Context-Sensitivity

This section gives an overview of the landscape of mildly context-sensitive formalisms and their most important properties. The concepts introduced here provide additional background information for the interested reader and are intended to put our contributions into a context.

It became evident that context-free grammar is not sufficient to capture the expressivity of natural language; an example is the aforementioned cross-serial word order in Dutch subordinate clauses [84] (see Figure 1.3). This observation motivated the search for formalisms that are sufficiently expressive and have beneficial computational properties, and led to the introduction of mild context-sensitivity.

2.4.1 Definition

The concept of mild context-sensitivity was originally introduced by Joshi [37] as a property of formalisms. The following definition concerning language classes is due to Kallmeyer, who gives an overview over parsing algorithms for mildly context-sensitive formalisms in her book [39].

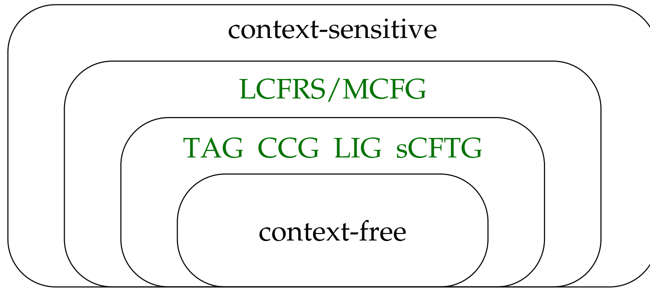


Figure 2.4: Inclusion relation of the two major mildly context-sensitive language classes. The inclusions are proper and mildly context-sensitive formalisms (highlighted) depicted in the same class have the same expressivity on strings, respectively.

Definition 2.4.1 (cf. [39, page 23]) *A class $\mathfrak{L}$ of languages is* mildly context-sensitive *iff*

1. *$\mathfrak{L}$ contains all context-free languages,*
2. *$\mathfrak{L}$ can describe cross-serial dependencies, i.e., there is an $n \geq 2$ such that $\{w^k \mid w \in \Sigma^*\} \in \mathfrak{L}$ for all $k \leq n$,*
3. *the languages in $\mathfrak{L}$ are polynomially parsable, i.e., $\mathfrak{L} \subseteq PTIME$, and*
4. *the languages in $\mathfrak{L}$ have the constant growth property.*

Similarly, a formalism is *mildly context-sensitive* iff the set of languages that it can specify (i.e., generate or accept) is mildly context-sensitive.

The *constant growth property* roughly means that, given a language, when ordering its words by length, this length grows at most linearly. Kallmeyer [39] gives the following formal definition, which is a modified version of the definition by Weir [95]: Given a language $\mathcal{L}$, there is a constant $c_0 > 0$ and a finite set of constants $C \subseteq \mathbb{N}_+$ so that for all words $w \in \mathcal{L}$ with length $|w| > c_0$ there is a word $w' \in \mathcal{L}$ such that $|w| = |w'| + c$, where $c \in C$. It is motivated by the intuition that sentences in natural language are composed of a finite set of clauses of bounded length.

There exist several natural classes of mildly context-sensitive languages, of which we will consider only the two major classes here. The placement of these classes in the Chomsky hierarchy and important formalisms generating them are shown in Figure 2.4. For a more detailed overview, see Kallmeyer [39, page 215]. Our main focus is on the tree-adjoining languages, because it is the class generated by CCG.

2.4.2 Tree-Adjoining Languages

On the lowest level of expressivity are the formalisms TAG, CCG, LIG, and sCFTG, which have the same expressive power on strings [93]. TAG and sCFTG have also been shown to have the same expressivity on trees [45]. They are usually referred to as *tree-adjoining languages* since TAG is probably the most prominent

and well-studied of these formalisms.[5] Automaton representations equivalent to TAG are the embedded push-down automaton [90] for string languages and the linear top-down push-down tree automaton [21] for tree languages. There also exist Chomsky–Schützenberger characterizations of the string languages [95] and tree languages [65] generated by TAG. Further, several logical characterizations have been proposed [1, 66, 68, 71].

5: This class is sometimes also referred to as *nearly context-free*, *just-non-context-free*, or *mildly non-context-free* [87].

The *counting language of degree* m is defined as $\{a_1^n \dots a_m^n \mid n \geq 1\}$ with alphabet $\Sigma = \{a_1, \dots, a_m\}$. Examples for languages that can be generated by TAG but not by CFG are are the counting languages of degree 3 and 4 and the *copy language* $\{ww \mid w \in \{a, b\}^*\}$ [37]. Being able to generate the cross-serial dependencies of Dutch essentially boils down to being able to generate the copy language [39, page 20].

Closure Properties The tree-adjoining languages are a substitution-closed full *abstract family of languages* (AFL) [90, Chapter 4.1]. In other words, they are closed under union, concatenation, intersection with regular languages, Kleene-star, homomorphism, inverse homomorphism, and additionally under substitution. On the other hand, they are not closed under intersection, complement, or under intersection with context-free languages.

When regarding the tree languages that can be generated by TAG, the closure properties of simple monadic context-free tree languages can be assumed. They are closed under union, relabeling, α-concatenation [32, Lemma 23], α-iteration [32, Lemma 25],[6] intersection with regular tree languages [69, Theorem 2.35], and inverse linear tree homomorphism [70, Theorem 8.1]. On the other hand, they are not closed under intersection or complement. The first follows from the fact that their path languages[7] are context-free [72, page 113]. Using the same technique as for showing that context-free languages are not closed under intersection [27, Theorem 6.4.2], it is possible to construct two sCFTGs whose intersection has a path language that is non-context-free. That the class is not closed under complement is a consequence of it being closed under union but not under intersection.

6: The *α-concatenation* $\mathcal{T}_1 \cdot_\alpha \mathcal{T}_2$ of two tree languages $\mathcal{T}_1, \mathcal{T}_2$ is obtained by replacing each occurrence of the leaf symbol α in a tree in $\mathcal{T}_1$ (nondeterministically) by a tree in $\mathcal{T}_2$. The *α-iteration* $\mathcal{T}_\alpha^*$ is obtained by iterating this process with tree language $\mathcal{T}$. These operations are also referred to as *α-substitution* and *α-substitution closure* [22] or as *α-replacement* and *α-replacement iteration* [32].

7: The *path language* of a tree language is the set of all paths that lead from the root to some leaf in any of the trees.

Pumping Lemma The following pumping lemma for TAG has been shown by Vijay-Shanker. It can be used to show that a language cannot be generated by TAG.

Theorem 2.4.2 *[90, Theorem 4.7, page 101] Let $\mathcal{L} = \mathcal{L}(\mathcal{G})$ be a language generated by a TAG $\mathcal{G}$. Then there exists a constant n such that if $w \in \mathcal{L}$ with $|w| > n$, we can write $w = u_1 v_1 w_1 v_2 u_2 v_3 w_2 v_4 u_3$ with $|v_1 w_1 v_2 v_3 w_2 v_4| \leq n$ and $|v_1 v_2 v_3 v_4| \geq 1$ such that for all $i \geq 0$, the word $u_1 v_1^i w_1 v_2^i u_2 v_3^i w_2 v_4^i u_3 \in \mathcal{L}$ as well.*

2.4.3 Multiple Context-Free Languages

On a higher level of expressivity we find two very similar formalisms that use the same idea, but have been developed independently. They are called multiple context-free grammar (MCFG) [78] and linear context-free rewriting systems (LCFRS) [94, 95]. They extend CFG in such a way that during the derivation, each nonterminal may store a tuple of strings that can be combined and extended using the production rules. Due to this feature, a nonterminal can generate a discontinuous part of the output string. Increasing the *fan-out* of the grammar, i.e., the maximum number of strings associated with a nonterminal, yields a hierarchy of increasing expressivity [78, Theorem 3.4].

Notable languages generated by MCFG are the *double copy language* $\{www \mid w \in \{a, b\}^*\}$ [39, page 110] and the counting languages of higher degrees (depending on the fan-out) [78, Example 2.1]. The abovementioned languages cannot be generated by a TAG [37]. Let MIX $= \{w \in \{a, b, c\}^* \mid |w|_a = |w|_b = |w|_c\}$, where $|w|_s$ is the number of occurrences of $s \in \{a, b, c\}$ in w. MIX cannot be generated by a TAG [43], but by an MCFG with fan-out 2 [73].

An important subclass is that of *well-nested* MCFG [40, 42], whose expressivity lies between that of TAG and MCFG. It can be generated by a number of different formalisms and can therefore be considered as a natural class as well. Well-nested MCFG with fan-out 2 has the same expressive power as TAG [41, Theorem 5.2].

Combinatory Categorial Grammar 3

3.1 Categories 25
3.2 Rules 26
3.3 Grammars 30

In this chapter, we will formally introduce the CCG formalism that is used throughout this thesis. An informal introduction was given in Section 1.1. The formalism is based on the definition given by Vijay-Shanker and Weir [93], but extends it by substitution rules.

We will start by introducing categories, which constitute the basic building blocks of CCG derivations. Then, we will describe how categories can be combined using rules, and also how these rules can be restricted, instantiated, and how they interact inside a rule system. Then, CCG is defined by combining the rule system and a lexicon. While the lexicon stores syntactic information for the input symbols in the form of categories, the rule system controls how to project this syntactic information onto longer input sequences.

3.1 Categories

Categories are syntactic types that can either be primitive, like S ("sentence") or NP ("noun phrase"), or complex, like (S\NP)/NP or S\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* or *atoms* and the set of *slashes* $D = \{/, \backslash\}$, we view *categories* as trees $\mathcal{C}(A) = T_{D,\emptyset}(A)$, which are binary trees whose internal nodes are slashes and whose leaves are atomic categories. The slashes are left-associative by convention, so by omitting unnecessary parentheses and using infix notation, 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 $i \in [k]$. The atomic category a is called the *target* of c and denoted by $\mathrm{target}(c)$. The slash–category pairs $|_i c_i$ are called *arguments* and their number k is called the *arity* of c and written as $\mathrm{arity}(c)$. Let $\mathrm{args}(c) = \{|_i c_i \mid i \in [k]\}$ be the set of all arguments of category c. In addition, we write $\mathrm{arg}(c, i)$ to access its i-th argument $|_i c_i$.

We distinguish two types of categories. In *first-order categories*, all arguments are atomic, so $c_i \in A$ for all $i \in [k]$, whereas in *higher-order categories*, the arguments can have arguments themselves. The set of all first-order categories over A is denoted by $\mathcal{C}_0(A)$.

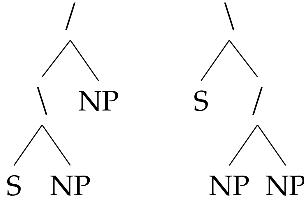


Figure 3.1: Categories (S\NP)/NP (left) and S\(NP/NP) (right) depicted as trees.

Example 3.1.1 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 it is a first-order category with arity 2. Note that this category is different from S\(NP/NP), which is a higher-order category with arity 1.

It is noted that, from the tree perspective, a sequence of arguments is a context $\alpha = \square|_1 c_1 \cdots |_k c_k$, which is why we will also call it *argument context*. We will sometimes omit the leading $\square$ and write an argument context simply as $|_1 c_1 \cdots |_k c_k$. The number k is the *length* of α and we write it as $|\alpha|$. We further let $\mathcal{A}(A) \subseteq C_{D,\emptyset}(A)$ be the set of all argument contexts over A. Finally, for every $k \in \mathbb{N}$, we define the sets $\mathcal{C}(A, k) = \{c \in \mathcal{C}(A) \mid \mathrm{arity}(c) \leq k\}$ and $\mathcal{A}(A, k) = \{\alpha \in \mathcal{A}(A) \mid k \geq |\alpha|\}$.

In Section 2.3, we introduced the notation $C[t]$ for the substitution $C[t]_{\mathrm{pos}_\square(C)}$, where $t \in T_{\Sigma_2,\Sigma_1}(\Sigma_0) \cup C_{\Sigma_2,\Sigma_1}(\Sigma_0)$ and $C \in C_{\Sigma_2,\Sigma_1}(\Sigma_0)$ for some ranked set Σ. When substituting a category or argument context $t \in \mathcal{C}(A) \cup \mathcal{A}(A)$ into an argument context $C \in \mathcal{A}(A)$, we may simply write tC.

For all categories and argument contexts $\alpha, \alpha' \in \mathcal{C}(A) \cup \mathcal{A}(A)$ we write $\alpha \sqsubseteq \alpha'$ if there exists a position $w \in \mathrm{pos}(\alpha') \cap \{1\}^*$ such that $\alpha = \alpha'|_w$ (i.e., α is a subtree of α' that is located on the path from the root to the left-most leaf). In other words, if $\alpha \sqsubseteq \alpha'$ and $\alpha' = a|_1 c_1 \cdots |_\ell c_\ell$, then $\alpha = a|_1 c_1 \cdots |_i c_i$ for some $i \leq \ell$. We say that α is a *prefix* of α'.

3.2 Rules

The rule system specifies how categories can be combined to derive new categories. We use the binary rules of composition, their special case of application, and substitution. We unify these binary rule types using a compact schema. Each binary rule describes how a *primary category* can be combined with a *secondary category* to produce an *output category*, which is written below the input categories. There is a *forward* and *backward* variant, differing in the direction of a specific slash in the primary category. We will write the respective forward rule on the left side and the backward rule on the right side in following, with the respective primary category being highlighted.

3.2.1 Combinatory Rules

We introduce *combinatory rules* as a compact schema that covers both composition and substitution rules in a uniform manner. Each

combinatory rule has one of the forms

$$\frac{b/c\alpha \quad c\alpha\beta}{b\alpha\beta} \qquad \frac{c\alpha\beta \quad b\backslash c\alpha}{b\alpha\beta}$$

where α and β are sequences of slash–variable pairs such that $|\alpha| \leq 1$ and $|\beta| \geq 0$. The variables in α and β as well as b and c are category variables that range over $\mathcal{C}(A)$. We shall refer to $|c\alpha$ as *bridging arguments* and to the sequence $\alpha\beta$ as the *excess*. The primary category expects the bridging arguments to be provided by the secondary category. The leading slash of the argument $|c$, which is the argument that gets removed when the rule is applied, specifies the direction where the secondary category is found. The output category follows the form of the primary category with the bridging arguments being replaced by the excess. It is noted that we can write down combinatory rules more explicitly as

$$\frac{b/c\alpha \quad c\alpha|_1c_1\cdots|_kc_k}{b\alpha|_1c_1\cdots|_kc_k} \qquad \frac{c\alpha|_1c_1\cdots|_kc_k \quad b\backslash c\alpha}{b\alpha|_1c_1\cdots|_kc_k}$$

where $\alpha \in \{\square, |d\}$ with $| \in D$, and for $i \in [k]$, we have $|_i \in D$, and b, c, d and c_i are category variables that range over $\mathcal{C}(A)$. Rules with $|\alpha| = 0$ are *composition rules*, and rules with $|\alpha| = 1$ are *substitution rules*; *application rules* have $|\alpha| = |\beta| = 0$. The length $|\alpha| + |\beta|$ is called the *degree* of the rule.

The form of combinatory rules presented above is generic in the sense that it only specifies the type of rule, the length of the excess, and the slash directions. As long as the input categories match these conditions and their bridging arguments coincide, such a rule can be applied. We call these rules *unrestricted*. For every $k \in \mathbb{N}$ let $\mathcal{R}(A, k)$ be the finite set of all unrestricted composition rules over A with degree at most k and let $\mathcal{R}_s(A, k)$ be the finite set of all unrestricted composition and substitution rules over A with degree at most k.

3.2.2 Rule Restrictions

We can optionally restrict combinatory rules using two types of *rule restrictions*. A *target restriction* can restrict the target of the variable b to a specific atomic category. More specifically, it can restrict b to range over a set of the form $\{c \in \mathcal{C}(A) \mid \mathrm{target}(c) = a\}$ with $a \in A$. A *secondary restriction* can restrict the variables c, d, and any of the variables c_i with $i \in \{0, \ldots, k\}$ to some specific category in $\mathcal{C}(A)$, respectively. A combinatory rule with rule restrictions has thus

one of the forms

$$\frac{ax/c\alpha \quad c\alpha|_1 c_1 \cdots |_k c_k}{ax\alpha|_1 c_1 \cdots |_k c_k} \qquad \frac{c\alpha|_1 c_1 \cdots |_k c_k \quad ax\backslash c\alpha}{ax\alpha|_1 c_1 \cdots |_k c_k}$$

where $a \in A, c \in \mathcal{C}(A) \cup \{y\}, \alpha \in \{\square, |d\}$ with $| \in D, d \in \mathcal{C}(A) \cup \{z\}$, and $|_i \in D$ and $c_i \in \mathcal{C}(A) \cup \{y_i\}$ for every $i \in [k]$. The category variables $y, z, y_1, \ldots, y_k$ can match any category of $\mathcal{C}(A)$, while the argument context variable x can match any argument context of $\mathcal{A}(A)$. Thus, $c, d, c_1, \ldots, c_k$ can each be either a concrete category of $\mathcal{C}(A)$ or a category variable if no restriction is intended. Note that in the primary category, we can only restrict the target and the last argument in the case of composition rules; in substitution rules, we can additionally restrict the second-to-last argument.

Let $\mathcal{R}(A)$ be the set of all composition and substitution rules over A, including both restricted and unrestricted rules.

3.2.3 Instantiation

Before a combinatory rule can be applied, it first has to be *instantiated* by replacing the variables by concrete categories or argument contexts, respectively, yielding a *ground instance* of the combinatory rule. More specifically, in the unrestricted variant, variables b, c, d, and c_i for $i \in [k]$ are replaced by concrete categories from $\mathcal{C}(A)$; in the restricted variant, variables y, z, and y_i for $i \in [k]$ are replaced by concrete categories from $\mathcal{C}(A)$ and the variable x by a concrete argument context from $\mathcal{A}(A)$.

We would like to point out that for a fixed CCG there is only a finite set of categories that the category variables c, d, and c_i for $i \in [k]$ [respectively, y, z, and y_i for $i \in [k]$] can be instantiated with to yield useful rules. 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 [93, Lemma 3.1] (see Section 3.3.3). Thus, the category variables listed above only offer a more succinct rule description in the notion of CCG. However, even if the secondary category contains no variables, a combinatory rule still can have infinitely many useful ground instances due to b ranging over infinitely many categories [respectively, x ranging over infinitely many argument contexts].

Example 3.2.1 Consider the restricted forward composition rule $r = \frac{Cx/C \quad C/E\backslash B}{Cx/E\backslash B}$, where $\{B, C, E\}$ are atoms. It has bridging argument $/C$ and excess $/E\backslash B$. A possible ground instance of this rule is $\frac{C/B/E/C \quad C/E\backslash B}{C/B/E/E\backslash B}$, where x was replaced by argument context $\square/B/E$. The primary category $c_1 = C/B/E/C$

has $\text{target}(c_1) = C$ and $\text{args}(c_1) = \{/B, /E, /C\}$. As c_1 takes three atomic categories as arguments, it is a first-order category and $\text{arity}(c_1) = 3$. The rule degree of composition rules is determined by the number of arguments replacing the last argument of the primary category, so r has degree $k = 2$. Note that the category $C/B/E/C$ can also be written as $((C/B)/E)/C$, which is different from $(C/B)/(E/C)$. The latter is a higher-order category. The rules $\frac{Cx/(E/C) \quad E/C\backslash B}{Cx\backslash B}$ and $\frac{Cx/(E/C) \quad E/C\backslash(B/B)}{Cx\backslash(B/B)}$ are both composition rules of degree 1.

Example 3.2.2 Let us now consider the restricted backward substitution rule $\frac{B/E/y_1 \quad Cx\backslash B/E}{Cx/E/y_1}$ with bridging arguments $\backslash B/E$ and excess $/E/y_1$, where y_1 is a category variable. Because the excess consists of two arguments, the rule has degree 2. Further, since the bridging arguments start with a backward slash, it is a backward rule and the secondary category appears on the left side. A possible instantiation of this rule is $\frac{B/E/(B\backslash C) \quad C\backslash B/E}{C/E/(B\backslash C)}$, where x was replaced by the empty context $\square$ and y_1 by the category $B\backslash C$.

3.2.4 Rule System

A *rule system* is a pair $\Pi = (A, R)$ consisting of an alphabet A and a finite set $R \subseteq \mathcal{R}(A)$ of rules over A. The set of all ground instances of Π induces a relation $\frac{\cdot \quad \cdot}{\cdot}\Pi \subseteq \mathcal{C}(A)^2 \times \mathcal{C}(A)$, which extends to a relation $\Rightarrow_\Pi \subseteq \mathcal{C}(A)^* \times \mathcal{C}(A)^*$ by

$$\Rightarrow_\Pi = \bigcup_{\varphi,\psi \in \mathcal{C}(A)^*} \left\{(\varphi\, c\, c'\, \psi,\ \varphi\, c''\, \psi) \;\middle|\; \frac{c \quad c'}{c''}\Pi\right\} .$$

It describes how to derive from a sequence of categories a new sequence of categories by repeatedly combining neighboring categories using the rules in R.

3.2.5 Type-Raising

Although it is not used by our CCG, *type-raising* [86, page 43] is presented here because it plays a role in practical implementations of CCG. It is used to 'turn arguments into functions over functions-over-such-arguments' [86, page 43]. Thereafter, a category that would normally serve as an argument (i.e., be used as a secondary category) can be composed with other function type categories and participate in certain coordinating structures. Type-raising is a unary rule, but as for binary rules, there exists a

forward and a backward variant. The rule has one of the forms

$$\frac{c}{b/(b\backslash c)} \qquad \frac{c}{b\backslash(b/c)}$$

where b and c are category variables. However, they are limited to a finite set of categories, thus infinite recursion is forbidden.

3.3 Grammars

We are now ready to give a formal definition of CCG based on that of Vijay-Shanker and Weir [93], but additionally allowing substitution rules.

Definition 3.3.1 ([93]) *A* combinatory categorial grammar *(CCG) is a tuple $\mathcal{G} = (\Sigma, A, R, I, L)$ consisting of*

- *an alphabet Σ of* input symbols*,*
- *a rule system (A, R),*
- *a set $I \subseteq A$ of* initial categories*,*
- *and a finite relation $L \subseteq \Sigma^{\leq 1} \times \mathcal{C}(A)$ called* lexicon*.*

It is a k-CCG, for $k \in \mathbb{N}$, if each $r \in R$ has degree at most k.

The lexicon can assign categories not only to the possible input symbols from Σ, but also to the empty word ε. These lexicon entries are called *ε-entries*. A category c occurring in some lexicon entry $c \in L(\alpha)$ with $\alpha \in \Sigma^{\leq 1}$ is called *lexical category*. Because $L(\Sigma^{\leq 1})$ is finite, there exists $k \in \mathbb{N}$ such that $L(\Sigma^{\leq 1}) \subseteq \mathcal{C}(A, k)$. The least such integer k is called the *arity of L* and denoted by $\text{arity}(L)$; i.e., $\text{arity}(L) = \max\{\text{arity}(c) \mid c \in L(\Sigma^{\leq 1})\}$. If $L = \emptyset$, then we let $\text{arity}(L) = 0$.

In a pure k-CCG, the rule system contains all unrestricted rules up to degree k, either including or excluding substitution rules. Thus, as long as the degree limit is respected, all instances of forward and backward rules of the respective rule types can be applied. Note how "non-pure" is different from "no rule restrictions": For example, a CCG that uses only unrestricted forward rules up to some degree is not pure although it has no rule restrictions.

Definition 3.3.2 *A k-CCG $\mathcal{G} = (\Sigma, A, R, I, L)$ is called* pure *if $R = \mathcal{R}(A, k)$ or $R = \mathcal{R}_s(A, k)$.*

The classical categorial grammars of Ajdukiewicz and Bar-Hillel [8] (AB-grammars) are 0-CCGs as they allow only application rules. However, 0-CCGs are more general since they may use a subset of

application rules or have rule restrictions, whereas AB-grammars are pure.

Example 3.3.3 As an example, let $\mathcal{G}_4 = \big(\Sigma, A, \mathcal{R}(A,0), I, L\big)$ be the CCG given by the input alphabet $\Sigma = \{b, c\}$, atomic categories $A = \{B, C\}$, initial categories $I = \{B\}$, and the lexicon L with $L(b) = \{B/C,\ B/C/B\}$ and $L(c) = \{C\}$. Clearly, $\mathcal{G}_4$ is a 0-CCG and an AB-grammar. For a slightly more involved example containing rule restrictions, we refer to Example 3.3.7.

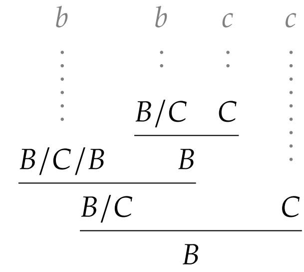


Figure 3.2: Derivation using the AB-grammar $\mathcal{G}_4$ of Example 3.3.3.

3.3.1 Generated String Language

The string language generated by a given CCG consists of those strings that via the lexicon are associated with some sequence of categories that the rule system can derive an initial category from. Note that we may also use ε-entries. Their categories can appear in the sequence of lexical categories without contributing any input symbols to the string as they correspond to the empty word ε.

Definition 3.3.4 *A CCG $\mathcal{G} = (\Sigma, A, R, I, L)$* generates *the category sequences $\mathcal{C}_{\mathcal{G}} \subseteq \mathcal{C}(A)^*$ and the string language $\mathcal{L}(\mathcal{G}) \subseteq \Sigma^*$, where $\Pi = (A, R)$,*

$$\mathcal{C}_{\mathcal{G}} = \left\{\varphi \in \mathcal{C}(A)^* \;\middle|\; \exists a_0 \in I \colon \varphi \Rightarrow_{\Pi}^{*} a_0\right\} \qquad \textit{and}$$
$$\mathcal{L}(\mathcal{G}) = L^{-1}(\mathcal{C}_{\mathcal{G}}) \ .$$

So the string language $\mathcal{L}(\mathcal{G})$ contains all strings that can be relabeled via the lexicon to a category sequence in $\mathcal{C}_{\mathcal{G}}$.

Definition 3.3.5 *A tree $t \in T_{\mathcal{C}(A),\emptyset}\big(L(\Sigma^{\leq 1})\big)$ is a* derivation tree of $\mathcal{G}$ *if $\frac{t(w1) \quad t(w2)}{t(w)}\Pi$ for every $w \in \mathrm{pos}(t) \setminus \mathrm{leaves}(t)$. The set of all such trees is denoted by $\mathcal{D}(\mathcal{G})$. Note that it is not required that $t(\varepsilon) \in I$.*

In other words, a derivation tree is a tree whose leaves are labeled by lexical categories and whose internal nodes are labeled by output categories of admissible rule applications that take as an input the categories labeling the respective child nodes.

Following standard conventions for CCG, we draw derivation trees with the root at the bottom. If the input symbol–category mapping specified by the lexicon is indicated, we visualize it as dotted lines (see Figure 1.1). However, when drawing derivation trees we will usually omit lexical entries.

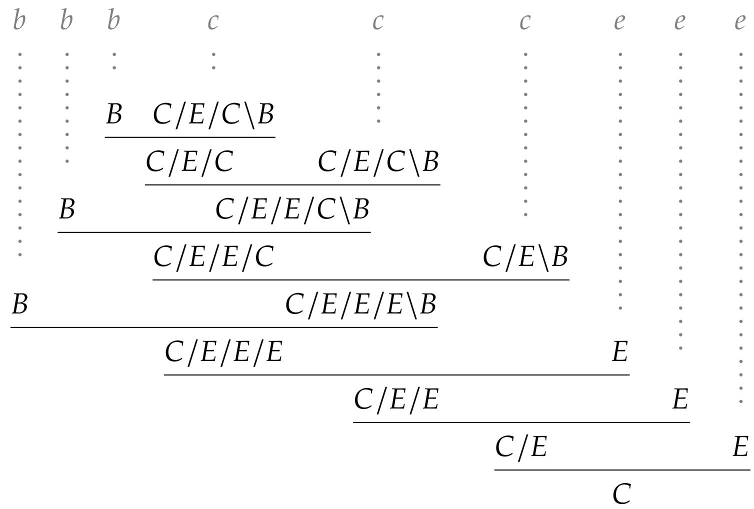


Figure 3.3: Derivation using the CCG $\mathcal{G}_5$ of Example 3.3.7.

Example 3.3.6 The grammar of Example 3.3.3 generates the string language $\mathcal{L}(\mathcal{G}_4) = \{b^i c^i \mid i \geq 1\}$, which is context-free but not regular. A derivation tree for the string $bbcc$ is shown in Figure 3.2. Note that in this and the following derivations, we do no longer explicitly label the rule instances with their names ("forward application", "backward composition", etc.) as we did in Figure 1.1.

Note that there can occur potentially infinitely many categories in the derivation trees $\mathcal{D}(\mathcal{G})$ of a given CCG $\mathcal{G}$, since their arity can grow linearly in the length of the input string. This is illustrated by the following example.

Example 3.3.7 Let $\mathcal{G}_5 = (\Sigma, A, R, \{C\}, L)$ be the 3-CCG given by the alphabet $\Sigma = \{b, c, e\}$, the atoms $A = \{B, C, E\}$, the lexicon L with $L(b) = \{B\}$, $L(c) = \{C/E\backslash B,\ C/E/C\backslash B\}$, $L(e) = \{E\}$, and the rule set R consisting of the rules

$$\frac{Cx/C \quad C/E/C\backslash B}{Cx/E/C\backslash B} \qquad \frac{Cx/E \quad E}{Cx}$$

$$\frac{Cx/C \quad C/E\backslash B}{Cx/E\backslash B} \qquad \frac{B \quad Cx\backslash B}{Cx}$$

where $x \in \mathcal{A}(A)$. From a few sample derivations (see Figure 3.3) we can convince ourselves that $\mathcal{G}_5$ generates the string language $\mathcal{L}(\mathcal{G}_5) = \{b^i c^i e^i \mid i \geq 1\}$, which demonstrates that 3-CCGs can generate non-context-free string languages. In addition, the derivation trees $\mathcal{D}(\mathcal{G}_5)$ contain infinitely many categories as labels.

3.3.2 Generated Tree Language

The string language generated by a CCG can be obtained by relabeling the leaf categories of the derivation trees rooted in an initial category using the lexicon. For the generated tree language we similarly allow a relabeling to avoid the restriction to the particular symbols of $\mathcal{C}(A)$. However, since the set $\mathcal{C}(A)$ is infinite, we restrict the possible relabelings such that they only depend on the target and the last argument of a given category.

Definition 3.3.8 *A* category relabeling $\rho\colon \mathcal{C}(A) \to \Delta$ *is a deterministic relabeling such that for all categories* $c, c' \in \mathcal{C}(A)$ *with* $\mathrm{target}(c) = \mathrm{target}(c')$ *and* $\mathrm{arg}\big(c, \mathrm{arity}(c)\big) = \mathrm{arg}\big(c', \mathrm{arity}(c')\big)$ *we have* $\rho(c) = \rho(c')$.

Definition 3.3.9 *Assume we are given a CCG* $\mathcal{G} = (\Sigma, A, R, I, L)$ *and a category relabeling* $\rho\colon \mathcal{C}(A) \to \Delta$*. Together,* $\mathcal{G}$ *and* ρ generate *the tree language* $\mathcal{T}_\rho(\mathcal{G}) \subseteq T_{\Delta_2, \emptyset}(\Delta_0)$ *given by*

$$\mathcal{T}_\rho(\mathcal{G}) = \left\{ \rho(t) \,\middle|\, t \in \mathcal{D}(\mathcal{G}),\ t(\varepsilon) \in I \right\} \ .$$

A tree language $\mathcal{T} \subseteq T_{\Delta_2,\emptyset}(\Delta_0)$ *is* generatable $\mathcal{G}$ *if there exists a category relabeling* $\rho'\colon \mathcal{C}(A) \to \Delta$ *such that* $\mathcal{T} = \mathcal{T}_{\rho'}(\mathcal{G})$.

Note that for the sake of simplicity the above definition uses a deterministic relabeling, although the lexicon constitutes a nondeterministic relabeling. This has no effect on the generative capacity.

3.3.3 Lexical Arguments

As mentioned above, all useful instantiations of secondary categories can only have *lexical arguments* (i.e., arguments of some lexical category), limiting these instantiations to a finite number. Similarly, primary categories can only have lexical arguments as well, although their number of instantiations may be infinite (see Example 3.3.7). The reason for this constraint on arguments is that the application of combinatory rules can never create any new arguments that were not present in the input categories already. Thus, all arguments appearing in valid derivation trees have their origin in the lexicon. We restate the following proposition [93, Lemma 3.1], which expresses this property.

Proposition 3.3.10 (see [93, Lemma 3.1]) *Let* $\Pi = (A, R)$ *be a rule system, and let*

$$c_1 \cdots c_k \Rightarrow^*_\Pi c'_1 \cdots c'_{k'}$$

for some categories $c_1, \ldots, c_k, c'_1, \ldots, c'_{k'} \in \mathcal{C}(A)$*. Then* $k' \leq k$ *and for each* $i \in [k']$ *the category* c'_i *is of the form* $c'_i = a_i |_1 c''_1 \cdots |_\ell c''_\ell$*, where* a_i *is the target of* c_j *for some* $j \in [k]$ *and for each* $m \in [\ell]$ *the argument* $|_m c''_m$ *is an argument of the category* c_{j_m} *for some* $j_m \in [k]$*.*

Let $\text{args}(L) = \bigcup_{c \in L(\Sigma^{\leq 1})} \text{args}(c)$ be the set of lexical arguments. We will often restrict ourselves to this finite set of arguments or to categories using only these arguments. For this, we define the set $\mathcal{C}_L(A, k) = \{c \in \mathcal{C}(A, k) \mid \text{args}(c) \subseteq \text{args}(L)\}$. The sets $\mathcal{C}_L(A)$, $\mathcal{A}_L(A, k)$, and $\mathcal{A}_L(A)$ are defined analogously.

Generative Power for Low Rule Degrees 4

4.1 0-CCG 36
4.2 1-CCG 46

In this chapter, we investigate the tree-generative capacity of CCG with rules of first and second degree. It is based on joint work with Marco Kuhlmann and Andreas Maletti (see Section 1.5). As in most of the formal work on CCG, we restrict our attention to the rules of application and composition. Although the formalism we use is based on the definition given by Vijay-Shanker and Weir [93], different from them, we will refrain from using ε-entries. However, our findings also show that their inclusion does not increase the generative power for these CCGs. In the following, we will briefly sketch the ideas behind our proofs.

CCG without composition operations can generate exactly those regular tree languages where for each node there exists a short path of bounded length to a leaf. Intuitively, application rules shorten a category on the way from the leaf to the root. This is why the maximal category length in the lexicon, which contains the categories labeling leaves, puts a bound on this path length. In our construction, we consider decompositions of trees into these short paths and compile a lexicon containing categories modeling these paths, which can then be assembled appropriately through the CCG operations. For the construction, pure CCG is sufficient.

CCG allowing composition rules of first degree can generate exactly the regular tree languages. To show this, we construct a CCG that uses a certain finite set of categories, such that given a set of states of a tree automaton, for each transition that could be present in a tree automaton using these, there are corresponding categories that could be combined via rules. The rule restrictions of the CCG then control which of these rules are permitted, with the aim to simulate only valid transitions of the given tree automaton.

If the 1-CCG is pure, as we will see in Section 5.2, it is less expressive in terms of tree languages. Regarding string languages, it can still generate all (ε-free) context-free languages. This is not immediately clear, as pure CCG allows certain transformations of derivation trees without affecting their acceptability. We show that, when using the classical construction for showing weak equivalence of CFG and categorial grammar [8], a pure 1-CCG with this lexicon still generates the same string language as a pure 0-CCG.

4.1 0-CCG

In this section we characterize the tree languages generatable by 0-CCG. This has already been investigated by Buszkowski with a focus on classical categorial grammar [10, Theorem 1.1]. We present an alternative proof for this result. Let $\mathcal{G} = (\Sigma, A, R, I, L)$ be a 0-CCG. An important property of 0-CCG is that each category that occurs in a derivation tree has arity at most $\operatorname{arity}(L)$. Thus, the derivation trees are built over a finite set of symbols.

Theorem 4.1.1 (see [8] and [89, Proposition 3.25]) *The string languages generated by* 0-CCG *are exactly the ε-free context-free languages. Moreover, for each* 0-CCG $\mathcal{G}$ *the derivation tree language* $\mathcal{D}(\mathcal{G})$ *and the tree languages generatable by* $\mathcal{G}$ *are regular.*

Proof. It is rather easy to show that every 0-CCG generates a context-free language.This also follows from the following fact about its derivation tree language. It is considerably more complicated to show that every ε-free context-free language can be generated by some 0-CCG, which has been proven by Bar-Hillel, Gaifman, and Shamir [8].[1] By [89, Proposition 3.25], the tree language $\mathcal{D}(\mathcal{G})$ is regular. Moreover, the regular tree languages are closed under relabelings [22, Theorem 2.4.16] and intersection [22, Theorem 2.4.2]. The intersection can be used to restrict $\mathcal{D}(\mathcal{G})$ to those trees whose root symbol belongs to I. As a result, also $\mathcal{T}_\rho(\mathcal{G})$ is regular for every category relabeling ρ. □

1: The result is considered to be equivalent to the *Greibach normal-form theorem* [26]. A CFG $\mathcal{G} = (N, \Sigma, S, P)$ in Greibach normal form has only productions of the form $n \to \alpha n_1 \dots n_m$, where $m \geq 0$, $n, n_1, \dots, n_m \in N$, and $\alpha \in \Sigma$. An equivalent categorial grammar can be constructed by using atoms N, input symbols Σ, initial categories S, and for each production a lexicon entry of the form $n/n_m/\dots/n_1 \in L(\alpha)$ [67, Proposition 1.9].

To characterize the tree languages generatable by 0-CCG, we need to introduce an additional structural property of the derivation tree language $\mathcal{D}(\mathcal{G})$ and the generatable tree languages. Roughly speaking, the *min-height* $\operatorname{mht}(t)$ of a tree t is the minimal length of a path from the root to a leaf. Recall that the height coincides with the maximal length of those paths. For all alphabets Σ_2 and Σ_0, let $\operatorname{mht}\colon T_{\Sigma_2,\emptyset}(\Sigma_0) \to \mathbb{N}$ be such that $\operatorname{mht}(a) = 0$ and

$$\operatorname{mht}\bigl(c(t_1, t_2)\bigr) = 1 + \min\bigl(\operatorname{mht}(t_1), \operatorname{mht}(t_2)\bigr)$$

for all $a \in \Sigma_0$, $c \in \Sigma_2$, and $t_1, t_2 \in T_{\Sigma_2,\emptyset}(\Sigma_0)$. A tree $t \in T_{\Sigma_2,\emptyset}(\Sigma_0)$ is *universally* mht-*bounded by* $h \in \mathbb{N}$ if $\operatorname{mht}(t|_w) \leq h$ for every $w \in \operatorname{pos}(t)$. Finally, a tree language $\mathcal{T} \subseteq T_{\Sigma_2,\emptyset}(\Sigma_0)$ is *universally* mht-*bounded by* h if every $t \in \mathcal{T}$ is universally mht-bounded by h, and it is *universally* mht-*bounded* if there exists $h \in \mathbb{N}$ such that it is universally mht-bounded by h. Note that "universally mht-bounded" is a purely structural property of a tree as it only depends on the shape of the tree, and is completely agnostic about the node labels. The property is thus preserved by the application

of a relabeling. Consequently, $\rho(\mathcal{T})$ is universally mht-bounded by h if and only if $\mathcal{T}$ is universally mht-bounded by h for every tree language $\mathcal{T} \subseteq T_{\Sigma_2,\emptyset}(\Sigma_0)$ and relabeling $\rho\colon (\Sigma_2 \cup \Sigma_0) \to \Delta$.

Example 4.1.2 Let us reconsider the 0-CCG $\mathcal{G}_4$ of Example 3.3.3. The derivation tree language $\mathcal{D}(\mathcal{G}_4)$ is universally mht-bounded by 1 (see Figure 3.3). The tree $\gamma\big(\alpha, \gamma\big(\gamma(\alpha,\alpha), \gamma(\alpha,\alpha)\big)\big)$ shown in Figure 4.1 also has min-height 1, but is only universally mht-bounded by 2, since the subtree $\gamma\big(\gamma(\alpha,\alpha), \gamma(\alpha,\alpha)\big)$ has min-height 2.

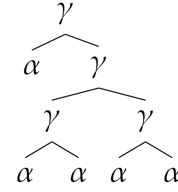


Figure 4.1: Tree with universal mht-bound 2.

It turns out that exactly the universally mht-bounded regular tree languages are generatable by 0-CCG. We already observed that the tree languages generatable by 0-CCG are regular, but for the converse we have to exploit the universal mht-bound. We first show that indeed the tree languages generatable by 0-CCG are universally mht-bounded.

Lemma 4.1.3 *Let $\mathcal{G}$ be a 0-CCG. Then the tree language $\mathcal{D}(\mathcal{G})$ is universally* mht*-bounded by* $\mathrm{arity}(L)$.

Proof. We first prove that $\mathrm{mht}(t) \le \mathrm{arity}(L) - \mathrm{arity}(t(\varepsilon))$ for every $t \in \mathcal{D}(\mathcal{G})$. For this, we use induction on t. In the induction base, for $t = c \in L(\Sigma)$, we have $\mathrm{arity}(c) \le \mathrm{arity}(L)$ and thus $\mathrm{mht}(c) = 0 \le \mathrm{arity}(L) - \mathrm{arity}(c)$. In the induction step, let $t = c(t_1, t_2)$ with $c \in \mathcal{C}(A)$ and $t_1, t_2 \in \mathcal{D}(\mathcal{G})$. Since $\mathcal{G}$ is a 0-CCG, we can only use application rules. As a result, we have $\mathrm{arity}(c) \le \max\big(\mathrm{arity}(t(1)), \mathrm{arity}(t(2))\big) - 1$, which we call (†). By the induction hypothesis (IH), we have $\mathrm{mht}(t_1) \le \mathrm{arity}(L) - \mathrm{arity}(t(1))$ and $\mathrm{mht}(t_2) \le \mathrm{arity}(L) - \mathrm{arity}(t(2))$. Thus, we obtain

$$
\begin{aligned}
&\mathrm{mht}\big(c(t_1, t_2)\big) \\
&= 1 + \min\Big(\mathrm{mht}(t_1), \mathrm{mht}(t_2)\Big) \\
&\overset{\text{(IH)}}{\le} 1 + \min\Big(\mathrm{arity}(L) - \mathrm{arity}\big(t(1)\big), \mathrm{arity}(L) - \mathrm{arity}\big(t(2)\big)\Big) \\
&= 1 + \mathrm{arity}(L) - \max\Big(\mathrm{arity}\big(t(1)\big), \mathrm{arity}\big(t(2)\big)\Big) \\
&\overset{(\dagger)}{\le} 1 + \mathrm{arity}(L) - \big(\mathrm{arity}(c) + 1\big) \\
&= \mathrm{arity}(L) - \mathrm{arity}\big(t(\varepsilon)\big)
\end{aligned}
$$

as required. This completes the induction. Now let $t \in \mathcal{D}(\mathcal{G})$ and $w \in \mathrm{pos}(t)$. Then $t|_w \in \mathcal{D}(\mathcal{G})$ is a derivation tree, and thus $\mathrm{mht}(t|_w) \le \mathrm{arity}(L)$ by the auxiliary statement, which proves that t is universally mht-bounded by $\mathrm{arity}(L)$. □

Since the universal mht-bound is a structural property, we can transfer it from the derivation trees to the relabeled trees. The following corollary is a direct consequence of Lemma 4.1.3.

Corollary 4.1.4 *There exist regular tree languages $\mathcal{T} \subseteq T_{\Sigma_2,\emptyset}(\Sigma_0)$ that are not generatable by any* 0*-CCG.*

Proof. The tree language $\mathcal{T} = T_{\Sigma_2,\emptyset}(\Sigma_0)$ for non-empty alphabets Σ_2 and Σ_0 is not generatable by any 0-CCG since it is not universally mht-bounded. □

We have thus established that the tree languages generatable by 0-CCG are regular and universally mht-bounded. We note that this result does not concern weak generative capacity. In particular, every (binary) regular tree language can be converted into a universally mht-bounded one that yields the same strings; this implies that a formalism that is able to generate all universally mht-bounded regular tree languages will still be weakly equivalent to the full class of regular tree languages, and therefore, to context-free (string) grammars.

The remainder of this section will be devoted to proving the inverse direction of Theorem 4.1.1. More precisely, we will show that every universally mht-bounded regular tree language can be generated by a 0-CCG, which will actually be pure. The construction uses the universal mht-bound, which yields short paths to a leaf. We utilize those paths to decompose the tree into *spines*, which are short paths in the tree that lead from a node to a leaf and are never longer than the universal min-height. When starting from a lexical category, the primary categories for the applications are placed along those spines and each spine terminates in an atomic category that can be combined with a category from another spine.

We will start by introducing the necessary notions and presenting the construction before proving its correctness. Therefore, let $\mathcal{T} \subseteq T_{\Sigma_2,\emptyset}(\Sigma_0)$ be a tree language that is regular and universally mht-bounded by some $h \in \mathbb{N}$. Since $\mathcal{T}$ is regular, there exists a CFG $\mathcal{G} = (N, \Gamma, S, P)$ and a deterministic relabeling ρ [22, Theorem 2.9.5] such that $\mathcal{T} = \{\rho(t) \mid t \in \mathcal{D}(\mathcal{G}),\ t(\varepsilon) \in S\}$. Without loss of generality, we can assume that N does not contain useless nonterminals.[2] Since the composition of two (deterministic) relabelings (seen as relations) is again a (deterministic) relabeling, it is sufficient to prove that there exists a pure 0-CCG $\mathcal{G}'$ and a category relabeling ρ' such that $\mathcal{T}_{\rho'}(\mathcal{G}') = \{t \in \mathcal{D}(\mathcal{G}) \mid t(\varepsilon) \in S\}$. We observe that $\mathcal{D}(\mathcal{G})$ is also universally mht-bounded by h.

2: We require that for every nonterminal $n \in N$ there exist strings $w, w_1, w_2 \in \Gamma^*$ and a start nonterminal $n_0 \in S$ such that $n_0 \Rightarrow^*_{\mathcal{G}} w_1 n w_2 \Rightarrow^*_{\mathcal{G}} w$.

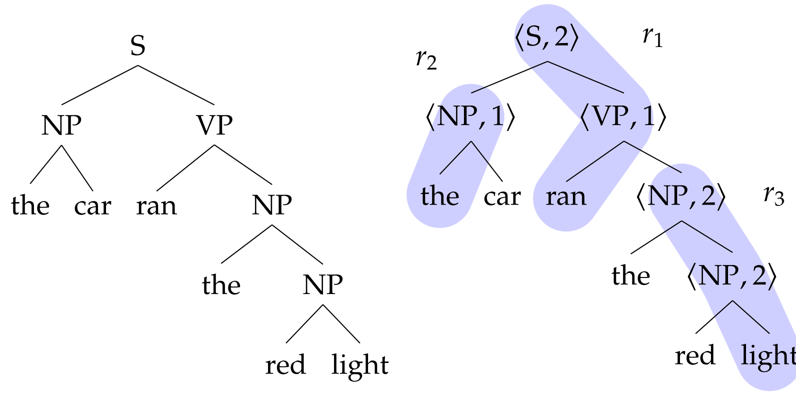


Figure 4.2: CFG derivation tree and a decomposition into spinal runs. (Trivial spines consisting of a single node are omitted.)

First, we will annotate each nonterminal of the CFG derivation trees with a direction $\delta \in [2]$ that indicates which successor continues the spine, where 1 indicates left and 2 indicates right (see Figure 4.2). Based on these annotations, the derivation trees can be decomposed into *spinal runs*, which are trees consisting of a path from some node to a leaf following the spine, together with the (unannotated) children located next to the spine (see Figure 4.3). For each spinal run r, we define direct spine access via $r[i] \in \Gamma \cup N$, which is the (unannotated) i-th node label on the spine. The *base* $\mathrm{base}(r) \in \Gamma$ of the spine is the label of the leaf at its bottom.

Definition 4.1.5 *For every $r \in T_{N\times[2],\emptyset}(\Gamma \cup N)$ we let $r[0] = r$ if $r \in \Gamma \cup N$ and $r[0] = n$ if $r = \langle n, \delta\rangle(r_1, r_2)$ for some $n \in N$, $\delta \in [2]$, and $r_1, r_2 \in T_{N\times[2],\emptyset}(\Gamma \cup N)$.*

The set $\mathrm{SR}(\mathcal{G})$ *of* spinal runs *of $\mathcal{G}$ is defined as the smallest set $M \subseteq T_{N\times[2],\emptyset}(\Gamma \cup N)$ such that*

(i) $\Gamma \subseteq M$, and
(ii) $\langle n, \delta\rangle(r_1, r_2) \in M$ for every $n \in N$, $\delta \in [2]$, $r_{3-\delta} \in \Gamma \cup N$, and $r_\delta \in M$ with $\big(n \to r_1[0] \cdot r_2[0]\big) \in P$.

For $r \in \mathrm{SR}(\mathcal{G})$ and $i \in [\mathrm{ht}(r)]$, direct spine access $r[i] \in \Gamma \cup N$ is inductively defined through $r[0]$ and, if $r = \langle n, \delta\rangle(r_1, r_2)$ for some $n \in N$, $\delta \in [2]$, $r_{3-\delta} \in \Gamma \cup N$, and $r_\delta \in \mathrm{SR}$, by $r[i] = r_\delta[i-1]$.

Additionally, let $\mathrm{base}(r) = r[\mathrm{ht}(r)]$.

Since $\mathcal{D}(\mathcal{G})$ is universally mht-bounded by h, we are interested only in those spines of length at most h. Thus, let us define the set $\mathrm{SR} = \big\{r \in \mathrm{SR}(\mathcal{G}) \mid \mathrm{ht}(r) \le h\big\}$. It is clearly finite, so we can use it to build the atomic categories $A = (\Gamma \cup N) \times \mathrm{SR} \times (\Gamma \cup N)$. Each atomic category thus stores two symbols, which are required for relabeling, and a spinal run to enforce consistency.

Each short spinal run can then be transformed into an argument context that will be used to simulate the respective spine in the CCG derivation tree. For this, the node that is placed at the bottom of the

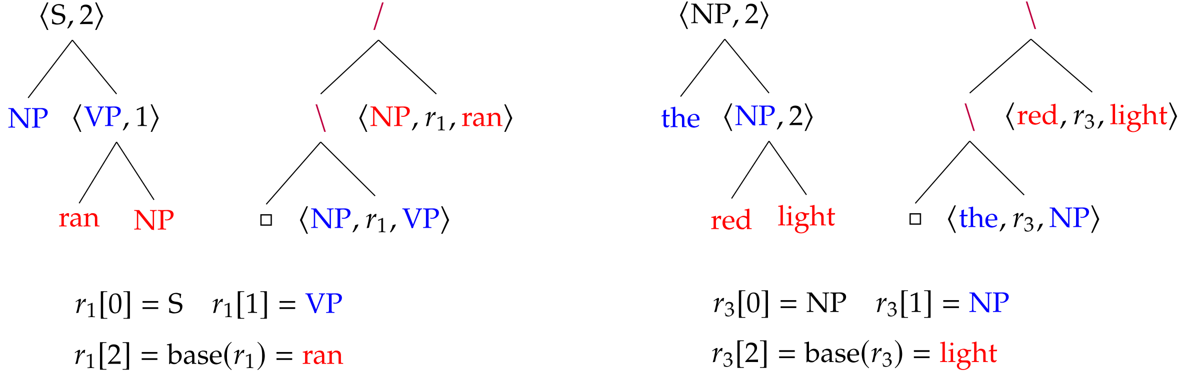


Figure 4.3: Two spinal runs r_1 (left) and r_3 (right) indicated in Figure 4.2 together with their argument contexts $\mathrm{argc}(r_1, r_1)$ and $\mathrm{argc}(r_3, r_3)$ and their bases $\mathrm{base}(r_1)$ and $\mathrm{base}(r_3)$.

spinal run needs to be transformed into the outermost argument, since it will be removed first in the CCG derivation. The labels of a spinal node and its non-spinal neighbor are stored in a single argument, which will be used to relabel the primary category and the combining secondary category that correspond to these nodes, respectively. The notion of spinal runs and the construction of the argument context are illustrated in Figure 4.3.

We formally specify this construction using the function $\mathrm{argc}(r, r')$, where r is the spinal run that the argument context to be constructed is based on, and r' will be stored in each atom, indicating the spinal run that the atom belongs to and started the construction. When the construction is started, r and r' coincide.

Definition 4.1.6 *We construct the argument context* $\mathrm{argc}(r, r')$, *which is an element of* $\mathcal{A}(A, h)$, *for all spinal runs* $r, r' \in \mathrm{SR}$ *as follows:*

- *if* $r \in \Gamma$, *then* $\mathrm{argc}(r, r') = \square$,
- *if* $r = \langle n, \delta\rangle(r_1, r_2)$ *for some* $n \in N$, $\delta \in [2]$, $r_{3-\delta} \in \Gamma \cup N$, *and* $r_\delta \in \mathrm{SR}$, *then*[3]

$$\mathrm{argc}(r, r') = \mathrm{argc}(r_\delta, r')\left[\square \,|\, \langle r_{3-\delta}, r', r_\delta[0]\rangle\right]$$

where $| = /$ *if* $\delta = 1$ *and* $| = \backslash$ *otherwise.*

3: For better readability, we write $|(\square, c)$ using the infix notation $\square \,|\, c$.

Now we are ready to define the pure 0-CCG that simulates $\mathcal{G}$ and the associated category relabeling ρ'. Each of the constructed argument contexts can be combined with any target storing the correct root symbol of the spinal run that the argument context belongs to, and is associated with the base of the spinal run via the lexicon.

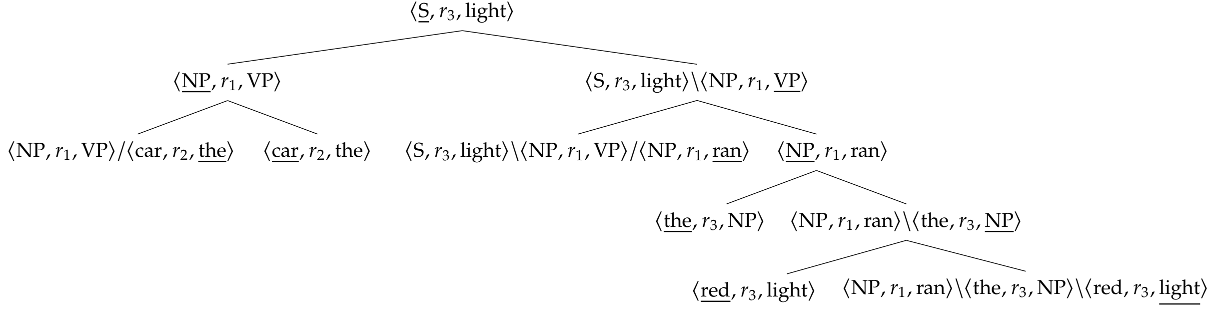


Figure 4.4: Derivation tree of the 0-CCG with the symbol resulting after relabeling underlined. It is based on the decomposition into spinal runs indicated in Figure 4.2. Note that, despite our conventions for CCG derivation trees, the root is drawn at the top to make the correspondence to Figure 4.2 more evident.

Definition 4.1.7 *Given CFG $\mathcal{G} = (N, \Gamma, S, P)$ without useless nonterminals and with $\mathcal{D}(\mathcal{G})$ being universally* mht*-bounded by h, the* 0*-CCG $\mathcal{G}' = \big(\Gamma, A, \mathcal{R}(A, 0), S \times \mathrm{SR} \times (\Gamma \cup N), L\big)$ is defined on the basis of $\mathrm{SR} = \big\{r \in \mathrm{SR}(\mathcal{G}) \mid \mathrm{ht}(r) \leq h\big\}$, atomic categories $A = (\Gamma \cup N) \times \mathrm{SR} \times (\Gamma \cup N)$, and the lexicon given by*

$$L(\gamma) = \Big\{ \mathrm{argc}(r, r)\Big[\big\langle r[0], r', g\big\rangle\Big] \;\Big|\; r, r' \in \mathrm{SR},\ \mathrm{base}(r) = \gamma,\ g \in \Gamma \cup N \Big\}$$

for every $\gamma \in \Gamma$.

Category relabeling $\rho' \colon \mathcal{C}(A) \to \Gamma \cup N$ is given by $\rho'\big(\langle g, r, g'\rangle\big) = g$ and $\rho'\big(c \,|\, \langle g, r, g'\rangle\big) = g'$ for all $\langle g, r, g'\rangle \in A$, $c \in \mathcal{C}(A)$, and $| \in D$ and is irrelevant for all other categories.

Example 4.1.8 The construction is illustrated in Figure 4.4. It shows the CCG derivation tree that is based on the decomposition into spinal runs indicated in Figure 4.2. Consider for example the rightmost run, which is r_3. In the CCG derivation tree, the leaf at the bottom of this run is a primary category that has two arguments, which store besides r_3 the labels of nodes on the spine and their siblings. Following the spine towards the root, these arguments are removed through application rules, until only the target remains. It stores r_1, which is the run that r_3 gets combined with.

Theorem 4.1.9 *Let $\mathcal{T} \subseteq T_{\Sigma_2,\emptyset}(\Sigma_0)$ be a tree language. Then the following are equivalent:*

- *$\mathcal{T}$ is generatable by some* 0*-CCG.*
- *$\mathcal{T}$ is generatable by some pure* 0*-CCG.*
- *$\mathcal{T}$ is regular and universally* mht*-bounded.*

Proof. The inclusion of the tree languages generatable by 0-CCG in the regular and universally mht-bounded tree languages is trivially true by Theorem 4.1.1 and Lemma 4.1.3. The step from the second to the first statement is also immediately apparent since each pure 0-CCG is a 0-CCG. In the following, we will show the remaining step: that each regular, universally mht-bounded tree language $\mathcal{T}$ is generated by some pure 0-CCG.

For this, we employ the construction specified in Definition 4.1.7. As already elaborated above, let $h \in \mathbb{N}$ be such that $\mathcal{T}$ is universally mht-bounded by h. Moreover, let CFG $\mathcal{G} = (N, \Gamma, S, P)$ and a deterministic relabeling ρ be chosen such that $\mathcal{G}$ does not contain useless nonterminals and such that $\mathcal{T} = \{\rho(t) \mid t \in \mathcal{D}(\mathcal{G}),\ t(\varepsilon) \in S\}$ [22, Theorem 2.9.5]. Then the pure 0-CCG $\mathcal{G}'$ and the category relabeling ρ' are constructed in accordance with Definition 4.1.7. Note that $\rho \circ \rho'$ is still a category relabeling. It remains to prove that $\mathcal{T}_{\rho'}(\mathcal{G}') = \{t \in \mathcal{D}(\mathcal{G}) \mid t(\varepsilon) \in S\}$.

Let us start with the observation that all categories that can occur in $\mathcal{D}(\mathcal{G}')$ are of a certain form. It is clear that all categories of $L(\Gamma)$ are left-spinal (i.e., all right children are leaves; see Figure 4.3). Thus, all arguments contain only atomic categories. Together with the fact that we can only use application rules, we obtain that all categories that can occur in derivations of $\mathcal{D}(\mathcal{G}')$ must be subtrees of the categories in $L(\Gamma)$. Consequently, let $\mathcal{C} \subseteq \mathcal{C}(A)$ be that subset of all categories.[4] Moreover, if an argument of a category $c \in \mathcal{C}$ is decorated with the spinal run r, then the category c is a subtree of a category built from $\operatorname{argc}(r, r)$ in the definition of L.

4: Although we defined the category relabeling $\rho' \colon \mathcal{C}(A) \to \Gamma \cup N$ for a wider domain, it could in fact be restricted to the subset $\mathcal{C}$ as the remaining categories cannot occur in derivations of $\mathcal{D}(\mathcal{G}')$.

We first prove the auxiliary statement $\rho'\big(\mathcal{D}(\mathcal{G}')\big) = \mathcal{D}(\mathcal{G})$.

We start with the direction $\rho'\big(\mathcal{D}(\mathcal{G}')\big) \subseteq \mathcal{D}(\mathcal{G})$ using induction. In the induction base we regard $c \in L(\Gamma)$. Now we distinguish two cases: If $c = \langle g, r', g' \rangle \in A$ is atomic, then $\rho'(c) = g = \operatorname{base}(r) \in \Gamma$ for some $r \in \mathrm{SR}$ by construction of the lexicon. Otherwise we have $c = c' | \langle g, r', g' \rangle$ for some $c' \in \mathcal{C}$, $| \in D$, and $\langle g, r', g' \rangle \in A$. Moreover, c was obtained from the argument context $\operatorname{argc}(r, r)$ for some $r \in \mathrm{SR}$ by substitution. In this case $\rho'(c) = g' = \operatorname{base}(r) \in \Gamma$ by the definition of "argc". Consequently, we have $\rho'(c) \in \Gamma$ and $\Gamma \subseteq \mathcal{D}(\mathcal{G})$, which completes the induction base.

In the induction step, let $d = g(d_1, d_2) \in \rho'(\mathcal{D}(\mathcal{G}'))$ with $g \in \Gamma \cup N$ and $d_1, d_2 \in \rho'\big(\mathcal{D}(\mathcal{G}')\big)$. Thus, there exist $c \in \mathcal{C}$ and $t_1, t_2 \in \mathcal{D}(\mathcal{G}')$ such that $c(t_1, t_2) \in \mathcal{D}(\mathcal{G}')$ and $\rho'\big(c(t_1, t_2)\big) = g(d_1, d_2)$. Moreover, $d_1, d_2 \in \mathcal{D}(\mathcal{G})$ by the induction hypothesis. It remains to prove that $g \to d_1(\varepsilon) \cdot d_2(\varepsilon) \in P$, and thus $g(d_1, d_2) \in \mathcal{D}(\mathcal{G})$. We already remarked that only left-spinal categories can occur in $\mathcal{D}(\mathcal{G}')$, hence $\{t_1(\varepsilon), t_2(\varepsilon)\} = \{c|a,\ a\}$ for some $| \in D$ and $a \in A$. Moreover, let $a = \langle g_1, r, g_2 \rangle$ for some $g_1, g_2 \in \Gamma \cup N$ and $r \in \mathrm{SR}$. We assume that $t_1(\varepsilon) = a$ and $t_2(\varepsilon) = c \backslash a$. The remaining case,

in which $t_1(\varepsilon) = c/a$ and $t_2(\varepsilon) = a$, is analogous. By the definition of ρ', we obtain that $d_1(\varepsilon) = g_1$ and $d_2(\varepsilon) = g_2$. Moreover, we already remarked that $c\backslash a$ must be a subtree of a category in $L(\Gamma)$. More precisely, it must be a subtree of the category $\mathrm{argc}(r, r)[\langle r[0], r', \overline{g}\rangle]$ for some $r' \in \mathrm{SR}$ and $\overline{g} \in \Gamma \cup N$ because we have the spinal run r annotated to an argument. Clearly, the left-spinal property makes it easy for us to locate the required subtree.

We distinguish two cases according to the definition of ρ'. If c is atomic, then $c = \langle r[0], r', \overline{g}\rangle$ and $r[0] = g$ by the definition of ρ'. By the construction of the argument context 'argc(r, r)' we have

$$\begin{aligned} r(\varepsilon) &= \langle r[0], 2\rangle = \langle g, 2\rangle \\ r(1) &= g_1 = d_1(\varepsilon) \\ r(2) &= \langle r[1], \delta\rangle = \langle g_2, \delta\rangle = \langle d_2(\varepsilon), \delta\rangle \end{aligned}$$

for some $\delta \in [2]$. Since $r \in \mathrm{SR}$ we have $r[0] \to r(1) \cdot r[1] \in P$, which yields $g \to d_1(\varepsilon) \cdot d_2(\varepsilon) \in P$ as desired with the help of the equations above. In the remaining case c is not atomic. Let $c = c'|\langle g', r, g''\rangle$ for some $c' \in \mathcal{C}$, $| \in D$, and $g', g'' \in \Gamma \cup N$. The definition of ρ' yields that $g'' = g$. Since 'argc' reverses the order (see Figure 4.3), our subtree $c\backslash a$ corresponds to an initial fragment of r. Thus, let $r = C[r'']$ with $C \in C_{N\times[2],\emptyset}(\Gamma\cup N)$ and $r'' \in \mathrm{SR}$ such that $c\backslash a = \mathrm{argc}\big(C[r''(\varepsilon)], r\big)[\langle r[0], r', \overline{g}\rangle]$. Let $w = \mathrm{pos}_\square(C)$. Since we have at least two arguments in $c\backslash a$, the definition of 'argc' yields $|w| \geq 2$, so let $w = w'\delta_1\delta_2$ with $w' \in [2]^*$ and $\delta_1, \delta_2 \in [2]$. Then the last two arguments are constructed by $\square|\langle g', r, g''\rangle\backslash\langle g_1, r, g_2\rangle = \mathrm{argc}\big(C|_{w'}[r''(\varepsilon)], r\big)$ and thus

$$\begin{aligned} r(w'\delta_1) &= \langle g'', \delta_2\rangle = \langle g, 2\rangle \\ r(w'\delta_1 1) &= g_1 = d_1(\varepsilon) \\ r(w'\delta_1 2) &= \langle g_2, \delta'\rangle = \langle d_2(\varepsilon), \delta'\rangle \end{aligned}$$

for some $\delta' \in [2]$. Since $r \in \mathrm{SR}$, we can infer that there is a production $r|_{w'\delta_1}[0] \to r|_{w'\delta_1}(1) \cdot r|_{w'\delta_1}[1] \in P$, which together with the equalities above yields the existence of the production $g \to d_1(\varepsilon) \cdot d_2(\varepsilon) \in P$ as required. Hence, $\rho'(\mathcal{D}(\mathcal{G}')) \subseteq \mathcal{D}(\mathcal{G})$.

For the converse inclusion $\mathcal{D}(\mathcal{G}) \subseteq \rho'\big(\mathcal{D}(\mathcal{G}')\big)$ we first prove an auxiliary statement. Let $t \in \mathcal{D}(\mathcal{G}')$ be a derivation with $\mathrm{arity}\big(t(\varepsilon)\big) = 0$ (i.e., it terminates in an atomic category). Further, let $t(\varepsilon) = \langle g, r, \overline{g}\rangle$ for some $g, \overline{g} \in \Gamma \cup N$ and $r \in \mathrm{SR}$. Then for every $r' \in \mathrm{SR}$ and $g' \in \Gamma \cup N$ there exists a derivation $t_{r',g'} \in \mathcal{D}(\mathcal{G}')$ such that $t_{r',g'}(\varepsilon) = \langle g, r', g'\rangle$ and $\rho'(t_{r',g'}) = \rho'(t)$. In other words, in any derivation with an atomic category at the root we can adjust the derivation such that the root label contains any desired spinal run $r' \in \mathrm{SR}$ and third component $g' \in \Gamma \cup N$. The resulting tree is

still a derivation and relabels to the same tree as t. This statement is very easy to prove using Proposition 3.3.10, which shows that $t(\varepsilon)$ is the target of a category of $L(\Gamma)$. However, by the construction of $L(\Gamma)$ those targets always allow each spinal run r' as second component and each g' as third component.

We return to the main proof that $\mathcal{D}(\mathcal{G}) \subseteq \rho'\big(\mathcal{D}(\mathcal{G}')\big)$. We rather prove the stronger statement that

$$\mathcal{D}\big(\mathcal{G}\big) \subseteq \rho'\big(\mathcal{D}_0(\mathcal{G}')\big) \qquad (\dagger)$$
$$\text{with} \quad \mathcal{D}_0(\mathcal{G}') = \big\{t \in \mathcal{D}(\mathcal{G}') \mid \operatorname{arity}\big(t(\varepsilon)\big) = 0\big\}$$

by induction. This means that we restrict ourselves on the right-hand side to those derivations $\mathcal{D}_0(\mathcal{G}')$ that finish in an atomic category.

In the induction base, let $u = \gamma \in \Gamma$. Then there is an atomic category $\langle \gamma, r, g\rangle \in L(\gamma) \cap \mathcal{D}_0(\mathcal{G}')$ for every $r \in \mathrm{SR}$ and $g \in \Gamma \cup N$, and thus $u \in \rho'\big(\mathcal{D}_0(\mathcal{G}')\big)$ by the definition of ρ'. In the induction step, we have $u \notin \Gamma$ and for all proper subtrees v of u the desired property $v \in \rho'\big(\mathcal{D}_0(\mathcal{G}')\big)$ is true.

For every $u' \in \mathcal{D}(\mathcal{G})$ and $w \in \operatorname{pos}(u')$ such that $u'(w) \in \Gamma$ we construct a spinal run $r_{w,u'}$ as follows:

- If $w = \varepsilon$, then $r_{w,u'} = u'$.
- If $w = \delta w'$ and $u' = n(u'_1, u'_2)$ for some $\delta \in [2]$, $w' \in \operatorname{pos}(u'_\delta)$, $n \in N$, and $u'_1, u'_2 \in \mathcal{D}(\mathcal{G})$, then the spinal run $r_{w,u'}$ is given by $r_{w,u'}(\varepsilon) = \langle n, \delta\rangle$, $r_{w,u'}|_\delta = r_{w',u'_\delta}$, and $r_{w,u'}|_{3-\delta} = u'_{3-\delta}(\varepsilon)$.

Roughly speaking, the spinal run $r_{w,u'}$ leads from the root of u' to the leaf located at w. It is easily checked that $r_{w,u'}$ is a spinal run of $\mathcal{G}$ (i.e., $r_{w,u'} \in \mathrm{SR}(\mathcal{G})$) and $\operatorname{ht}(r_{w,u'}) = |w|$.

Now we return to the tree $u \in \mathcal{D}(\mathcal{G})$. Since u is universally mht-bounded by h, there exists a position $w \in \operatorname{pos}(u)$ such that $|w| \leq h$ and $u(w) \in \Gamma$. Thus, we select such a leaf w with a short path from the root arbitrarily and let $w = \delta_1 \cdots \delta_\ell$ with $\delta_1, \ldots, \delta_\ell \in [2]$. Consequently, the spinal run $r = r_{w,u} \in \mathrm{SR}(\mathcal{G})$ has height $\ell \leq h$, which yields that $r \in \mathrm{SR}$. We first deal with the positions outside the spine. For every $i \in [\ell]$, let $\overline{\delta_i} = 3 - \delta_i$, so we have $\overline{\delta_i} = 1$ if $\delta_i = 2$, and $\overline{\delta_i} = 2$ if $\delta_i = 1$. Moreover, we define the positions $\overline{w_i} = \delta_1 \cdots \delta_{i-1}\overline{\delta_i}$, which refer to the positions outside the spine of r. Similarly, for every $0 \leq i \leq \ell$, let $w_i = \delta_1 \cdots \delta_i$ be the i-th position on the spine of r. Trivially, $r(\overline{w_i}) = u(\overline{w_i})$ for all $i \in [\ell]$ by the construction of $r = r_{w,u}$. By the induction hypothesis, for every $i \in [\ell]$ we know that $u|_{\overline{w_i}} \in \rho'(\mathcal{D}_0(\mathcal{G}'))$ and together with the auxiliary statement we obtain that there exists a derivation $t_i \in \mathcal{D}_0(\mathcal{G}')$ such that $u|_{\overline{w_i}} \in \rho'(t_i)$ and $t_i(\varepsilon) = \langle u(\overline{w_i}), r, u(w_i)\rangle$. By construction, we have $r[i] = u(w_i)$ for all $0 \leq i \leq \ell$. In particular, $\operatorname{base}(r) = u(w)$. Let $r' \in \mathrm{SR}$ be an arbitrary spinal run

and $g' \in \Gamma \cup N$. Consider category $c = \operatorname{argc}(r, r)[\langle r[0], r', g' \rangle]$, which is in $L(u(w))$ by construction of L, since $\operatorname{base}(r) = u(w)$. More precisely, let $c = \langle r[0], r', g' \rangle |_1 a_1 |_2 \cdots |_\ell a_\ell$ for some $|_1, \ldots, |_\ell \in D$ and $a_1, \ldots, a_\ell \in A$. Note that the categories $a_1, \ldots, a_\ell$ are atomic because all relevant categories are left-spinal. By the construction of c we know for every $i \in [\ell]$ that (i) $|_i = /$ if and only if $\delta_i = 1$, and (ii) $a_i = t_i(\varepsilon)$. Now we can construct the required derivation of $\mathcal{D}_0(\mathcal{G}')$ by combining this category c with the subderivations $t_i \in \mathcal{D}_0(\mathcal{G}')$. Let $t'_\ell = c$ and for every $i \in \mathbb{Z}_\ell$ let

$$t'_i(\varepsilon) = \langle r[0], r', g' \rangle |_1 a_1 |_2 \cdots |_i a_i$$
$$t'_i|_{\delta_{i+1}} = t'_{i+1} \qquad t'_i|_{\overline{\delta_{i+1}}} = t_{i+1} \ .$$

Finally, we set $t' = t'_0$. A simple check shows that $t' \in \mathcal{D}_0(\mathcal{G}')$. It remains to show that $u = \rho'(t')$. Obviously, $\operatorname{pos}(u) = \operatorname{pos}(t')$, so we need to show that $u(w) = \rho'\big(t'(w)\big)$ for every $w \in \operatorname{pos}(u)$. If $w = \overline{w_i} w'$ for some $i \in [\ell]$ and $w' \in \operatorname{pos}(u|_{\overline{w_i}})$, then this is trivially true because $t'|_{\overline{w_i}} = t_i$ and we already observed that $u|_{\overline{w_i}} \in \rho'(t_i)$. Consequently, we only need to prove the property for all the prefixes w_i (with $i \in \mathbb{Z}_{\ell+1}$) of w. Let $i \in \mathbb{Z}_{\ell+1}$. By the construction of t' we have $t'(w_i) = \langle r[0], r', g' \rangle |_1 a_1 |_2 \cdots |_i a_i$. For $i = 0$, we thus obtain $\rho'\big(\langle r[0], r', g' \rangle\big) = r[0] = u(\varepsilon)$. For all $i \in [\ell]$ we have $\rho'\big(t'(w_i)\big) = u(w_i)$ since $a_i = t_i(\varepsilon) = \langle u(\overline{w_i}), r, u(w_i) \rangle$. Consequently, we established the stronger statement (†) and thus also $\mathcal{D}(\mathcal{G}) \subseteq \rho'\big(\mathcal{D}(\mathcal{G}')\big)$.

We have thus shown the two main statements $\rho'\big(\mathcal{D}(\mathcal{G}')\big) = \mathcal{D}(\mathcal{G})$ and $\rho'\big(\mathcal{D}_0(\mathcal{G}')\big) = \mathcal{D}(\mathcal{G})$. With the help of the latter statement, we can now reason as follows:

$$\begin{aligned} \mathcal{T}_{\rho'}(\mathcal{G}') &= \big\{ \rho'(t) \mid t \in \mathcal{D}_0(\mathcal{G}'),\ t(\varepsilon) \in S \times \mathrm{SR} \times (\Gamma \cup N) \big\} \\ &= \big\{ t \in \mathcal{D}(\mathcal{G}) \mid t(\varepsilon) \in S \big\} \ , \end{aligned}$$

which concludes the proof. □

The good closure properties of regular tree languages allow us to derive a number of closure results for the tree languages generatable by 0-CCG (see Table 7.1). We have seen that, while classical categorial grammars and context-free grammars are weakly equivalent, they are not strongly equivalent when considered as tree-generating devices. More specifically, the class of derivation tree languages of classical categorial grammars are a proper subclass of the class of local tree languages (i.e., derivation tree languages of context-free grammars). This result is similar to a result by Schabes, Abeillé, and Joshi [75] showing that context-free grammars are not closed under strong lexicalization, meaning that there are context-free grammars such that no lexicalized grammar[5] generates the same derivation tree language.

5: A CFG is called lexicalized if every production contains a terminal symbol.

4.2 1-CCG

In this section we will consider 1-CCG, which allows rules of degree at most 1. Thus, the secondary categories appearing in derivation trees have at most one additional argument after the category that is consumed by the composition. We will prove that 1-CCG generates exactly the regular tree languages by showing inclusion in both directions. Regarding the first direction of the equivalence proof, it has already been reasoned in the literature [19, 49] that the derivation trees of 1-CCG can be simulated by CFG.

Lemma 4.2.1 (see [19, Proposition 4] and [49, Section 3.1]) *For each* 1-CCG $\mathcal{G}$ *the derivations* $\mathcal{D}(\mathcal{G})$ *and the generated tree language are regular.*

Proof. The derivation tree language $\mathcal{D}(\mathcal{G})$ contains only a finite number of arguments. Furthermore, there exist only finitely many secondary categories, since the degree of the rules is limited [49]. A rule of degree 1 only replaces the last argument of the primary category by another argument. As a consequence, the arity of the primary category cannot increase through composition. So we have only a finite number of categories and can use the same construction that was used in [89, Proposition 3.25] for 0-CCG to show that $\mathcal{D}(\mathcal{G})$ is regular. Analogous to the proof for 0-CCG, $\mathcal{T}_\rho(\mathcal{G})$ is regular for every relabeling ρ as well since regular tree languages are closed under relabelings [22, Theorem 2.4.16] and intersection [22, Theorem 2.4.2]. □

The following lemma establishes a normal form for regular tree grammar that is easily achieved using standard techniques. The construction is illustrated in Example 4.2.3. An RTG (N, Σ, S, P) is a *tree automaton* (TA) if for each production $(n \to r) \in P$ there exist $\sigma \in \Sigma$ and $n', n'' \in N$ such that $r = \sigma$ or $r = \sigma(n', n'')$.

Lemma 4.2.2 *For each RTG there exist a TA* $\mathcal{G}' = (\mathbb{Z}_m, \Sigma, S', P')$ *that accepts the same tree language and a mapping* $\pi \colon \mathbb{Z}_m \to \Sigma$ *such that every nonterminal* $n \in \mathbb{Z}_m$ *generates a uniquely defined terminal symbol* $\pi(n)$*; i.e., for all* $n \in \mathbb{Z}_m$ *and* $t \in T_\Sigma$ *with* $n \Rightarrow^+_{\mathcal{G}'} t$ *we have* $t(\varepsilon) = \pi(n)$*.*

Proof. For each RTG there exists an equivalent TA $\mathcal{G} = (N, \Sigma, S, P)$ by [22, Theorem 2.3.6]. Given a TA $\mathcal{G}$ in which a nonterminal n can produce terminals σ_1 and σ_2 with $\sigma_1 \neq \sigma_2$, we can construct an equivalent TA $\mathcal{G}'$ by creating copies n_{σ_1} and n_{σ_2} of n. Productions with n on the left-hand side like $n \to \sigma(n', n'')$ and $n \to \sigma$

with $\sigma \in \{\sigma_1, \sigma_2\}$ are replaced by $n_\sigma \longrightarrow \sigma(n', n'')$ and $n_\sigma \longrightarrow \sigma$, respectively. On the other hand, productions with n on the right-hand side [e.g., $n' \longrightarrow \sigma(n, n'')$] are replaced by one copy of the production for each copy of n [e.g., $n' \longrightarrow \sigma(n_{\sigma_1}, n'')$ and $n' \longrightarrow \sigma(n_{\sigma_2}, n'')$]. Each copy n_σ of a start nonterminal $n \in S$ becomes part of the new set of start nonterminals. The nonterminal set $\mathbb{Z}_m$ is obtained by applying a bijection $\pi\colon N' \longrightarrow \mathbb{Z}_{|N'|}$, where N' contains all copied and unmodified nonterminals of N. It is easy to see that $\mathcal{G}'$ generates the same tree language as $\mathcal{G}$. $\square$

Example 4.2.3 Let $\mathcal{G} = (N, \Sigma, S, P)$ be the TA that is given by $N = \{s, a, b, c\}, \Sigma = \{\sigma, \tau\}, S = \{s\}$, and

$$P = \Big\{s \longrightarrow \sigma(a, b),\ a \longrightarrow \sigma(b, c),\ a \longrightarrow \tau,\ b \longrightarrow \sigma,\ c \longrightarrow \sigma\Big\}\ .$$

Nonterminal a can produce the terminal symbol σ or τ, so our intermediate TA $\mathcal{G}' = (N', \Sigma, S, P')$ has the nonterminal alphabet $N' = \{s, a_\sigma, a_\tau, b, c\}$, the production $a \longrightarrow \sigma(b, c)$ has been replaced by $a_\sigma \longrightarrow \sigma(b, c)$, and $a \longrightarrow \tau$ has been replaced by $a_\tau \longrightarrow \tau$. Instead of $s \longrightarrow \sigma(a, b)$, the two copies $s \longrightarrow \sigma(a_\sigma, b)$ and $s \longrightarrow \sigma(a_\tau, b)$ are contained in P'. After mapping N' to $\mathbb{Z}_5$, we obtain the productions

$$P'' = \Big\{0 \longrightarrow \sigma(1,3),\ 0 \longrightarrow \sigma(2,3),\ 1 \longrightarrow \sigma(3,4),\ 2 \longrightarrow \tau,\ 3 \longrightarrow \sigma,\ 4 \longrightarrow \sigma\Big\}\ .$$

Given a TA $\mathcal{G} = (\mathbb{Z}_m, \Sigma, S, P)$ in the normal form of Lemma 4.2.2 with mapping $\pi\colon \mathbb{Z}_m \longrightarrow \Sigma$, we are allowed to regard only the nonterminals of $\mathcal{G}$ when constructing an equivalent 1-CCG. Our goal is to find a 1-CCG $\mathcal{G}' = (\Sigma', A, R, I, L)$ and a category relabeling $\rho\colon \mathcal{C}(A) \longrightarrow \mathbb{Z}_m$ such that $\mathcal{T}(\mathcal{G}) = \mathcal{T}_{\pi\circ\rho}(\mathcal{G}')$. Because ρ maps from categories to nonterminals, but $\mathcal{T}(\mathcal{G})$ is labeled by terminal symbols, we use $\pi\colon \mathbb{Z}_m \longrightarrow \Sigma$ to map from nonterminals to terminals. Given a production $n \longrightarrow \sigma(n_1, n_2) \in P$ and a category relabeling $\rho\colon \mathcal{C}(A) \longrightarrow \mathbb{Z}_m$, there have to exist categories $c_1 \in \rho^{-1}(n_1)$ and $c_2 \in \rho^{-1}(n_2)$ for each category $c \in \rho^{-1}(n)$ such that $\frac{c_1 \quad c_2}{c}$ is a valid ground instance of a rule in R. This ensures that each category can be derived by the composition of two categories mapped to matching nonterminals. We only regard first-order categories with at most one argument. Starting from any nonterminal, the productions in P allow the direct derivation of at most all ordered pairs of nonterminals as children. The number of ordered pairs $\mathbb{Z}_m^2$ increases quadratically in m, whereas the number of different composition input pairs resulting in a fixed category increases only linearly in $|A|$: The *category matrix* depicted in Figure 4.5 illustrates that a first-order category with one argument is the result of the forward compositions of $|A|$ different

	a_0	a_1	a_2	a_3
a_0	a_0/a_0	a_0/a_1	a_0/a_2	a_0/a_3
a_1	a_1/a_0	a_1/a_1	a_1/a_2	a_1/a_3
a_2	a_2/a_0	a_2/a_1	a_2/a_2	a_2/a_3
a_3	a_3/a_0	a_3/a_1	a_3/a_2	a_3/a_3

Figure 4.5: The category matrix contains all first-order categories of arity 1 with only forward slashes in a CCG with four atoms. Each category is the result of the forward composition of a category taken from the same row and one from the same column, respectively. The i-th entry of each row can be combined with the i-th entry of each column. Thus, each category a/a' is the result of four different forward compositions combining a/a'' and a''/a' (with four choices for a'').

	$\langle 0,0\rangle$	$\langle 0,1\rangle$	$\langle 0,2\rangle$	$\langle 1,0\rangle$	$\langle 1,1\rangle$	$\langle 1,2\rangle$	$\langle 2,0\rangle$	$\langle 2,1\rangle$	$\langle 2,2\rangle$
$\langle 0,0\rangle$	0	1	2	0	1	2	0	1	2
$\langle 0,1\rangle$	0	1	2	0	1	2	0	1	2
$\langle 0,2\rangle$	0	1	2	0	1	2	0	1	2
$\langle 1,0\rangle$	1	2	0	1	2	0	1	2	0
$\langle 1,1\rangle$	1	2	0	1	2	0	1	2	0
$\langle 1,2\rangle$	1	2	0	1	2	0	1	2	0
$\langle 2,0\rangle$	2	0	1	2	0	1	2	0	1
$\langle 2,1\rangle$	2	0	1	2	0	1	2	0	1
$\langle 2,2\rangle$	2	0	1	2	0	1	2	0	1

Figure 4.6: The relabeling matrix illustrates how a 1-CCG with nine atoms is relabeled via $\rho_{\mathcal{G}} : \mathcal{C}(\mathbb{Z}_3^2) \to \mathbb{Z}_3$, obtained from a TA $\mathcal{G}$ with three nonterminals by applying Definition 4.2.4. Suppose we want to find two categories projected to nonterminals $(g, h) = (0, 2)$ whose composition yields $\langle i, j\rangle/\langle i', j'\rangle = \langle 0, 1\rangle/\langle 0, 1\rangle$. These are the categories $\langle 0, 1\rangle/\langle 1, 0\rangle$ and $\langle 1, 0\rangle/\langle 0, 1\rangle$ because

$$\begin{aligned}\langle k, \ell\rangle &= \langle h - j' \bmod 3, g - i \bmod 3\rangle \\ &= \langle 2 - 1 \bmod 3, 0 - 0 \bmod 3\rangle \\ &= \langle 1, 0\rangle \ .\end{aligned}$$

category pairs. In addition to composition rules, application rules are necessary to obtain an atomic initial category. Based on these observations, we construct the 1-CCG $C_{\mathcal{G}}$ with m^2 atoms in the following way.

Definition 4.2.4 *Given a TA $\mathcal{G} = (\mathbb{Z}_m, \Sigma, S, P)$ in the normal form of Lemma 4.2.2, we construct the* 1-CCG

$$C_{\mathcal{G}} = \left(\Sigma_0,\ \mathbb{Z}_m^2,\ R,\ \rho_{\mathcal{G}}^{-1}(S) \cap \mathbb{Z}_m^2,\ L\right)$$

and the category relabeling $\rho_{\mathcal{G}} : \mathcal{C}(\mathbb{Z}_m^2) \to \mathbb{Z}_m$ such that

$$\begin{aligned}R = \bigcup_{\substack{\sigma\in\Sigma_2\\ a,b,c\in\mathbb{Z}_m^2}} \Bigg(&\left\{ \frac{ax/b \quad b}{ax} \;\middle|\; \rho_{\mathcal{G}}(a) \to \sigma\big(\rho_{\mathcal{G}}(a/b), \rho_{\mathcal{G}}(b)\big) \in P \right\} \\ &\cup \left\{ \frac{ax/b \quad b/c}{ax/c} \;\middle|\; \rho_{\mathcal{G}}(a/c) \to \sigma\big(\rho_{\mathcal{G}}(a/b), \rho_{\mathcal{G}}(b/c)\big) \in P \right\} \Bigg) \ ,\end{aligned}$$

$$\begin{aligned}L(\alpha) = \bigcup_{a,b\in\mathbb{Z}_m^2} &\Big(\big\{ a \mid \rho_{\mathcal{G}}(a) \to \alpha \in P \big\} \cup \big\{ a/b \mid \rho_{\mathcal{G}}(a/b) \to \alpha \in P \big\} \Big) \\ &\text{for all } \alpha \in \Sigma_0 \ ,\end{aligned}$$

and $\rho_{\mathcal{G}}\big(\langle i, j\rangle\big) = i$ as well as $\rho_{\mathcal{G}}\big(\langle i, j\rangle/\langle i', j'\rangle\big) = i + j' \bmod m$ for all $i, i', j, j' \in \mathbb{Z}_m$. The relabeling on all other categories is irrelevant.

Lemma 4.2.5 *Every regular tree language $\mathcal{T}$ is generatable by a 1-CCG.*

Proof. By definition there exists an RTG $\mathcal{G}$ such that $\mathcal{T}(\mathcal{G}) = \mathcal{T}$. By Lemma 4.2.2 there exists an equivalent TA $\mathcal{G}' = (\mathbb{Z}_m, \Sigma, S, P)$ and a mapping $\pi : \mathbb{Z}_m \to \Sigma$ with the properties specified in Lemma 4.2.2 (i.e., each nonterminal symbol $n \in \mathbb{Z}_m$ generates a uniquely de-

fined terminal symbol $\pi(n)$). In the following, we show that the 1-CCG $C_{\mathcal{G}'} = \big(\Sigma_0, \mathbb{Z}_m^2, R, \rho_{\mathcal{G}'}^{-1}(S) \cap \mathbb{Z}_m^2, L\big)$ given in Definition 4.2.4 generates the tree language $\mathcal{T} = \mathcal{T}(\mathcal{G}')$ using the category relabeling $\pi \circ \rho_{\mathcal{G}'}$. We achieve this by arguing that $\mathcal{D}(\mathcal{G}') = \rho_{\mathcal{G}'}\big(\mathcal{D}(C_{\mathcal{G}'})\big)$, which by the choice $\rho_{\mathcal{G}'}^{-1}(S) \cap \mathbb{Z}_m^2$ of initial categories and the definition of $\rho_{\mathcal{G}'}$ already proves the main statement.

The category $\langle i, j\rangle/\langle i', j'\rangle$ results from composing $\langle i, j\rangle/\langle k, \ell\rangle$ and $\langle k, \ell\rangle/\langle i', j'\rangle$, where $i, i', j, j', k, \ell \in \mathbb{Z}_m$. Figure 4.6 illustrates the category relabeling $\rho_{\mathcal{G}'}$ by means of a *relabeling matrix*, which is a matrix indexed by atoms a and a' with entries indicating the relabeling $\rho_{\mathcal{G}'}(a/a')$. The row and column labels of this matrix follow lexicographic order. When we slice the matrix evenly into blocks of size $m \times m$, we can observe that the entries in the rows cycle through the nonterminals, whereas in a single column, each block has only a single nonterminal in all m entries. This is because when looking up category $\langle i, j\rangle/\langle i', j'\rangle$, the value of j' changes in every entry, whereas the value of i changes only every m entries. Nonetheless, a complete column of the whole relabeling matrix contains all m nonterminals. Relabeling in this manner ensures that all pairs (g, h) of nonterminals are covered for each output category a/a': We can find an atom a'' such that a/a'' relabels to g and a''/a' relabels to h and their composition yields a/a' as required.

Formally, when given a category $\langle i, j\rangle/\langle i', j'\rangle$ and an ordered pair $(g, h) \in \mathbb{Z}_m^2$ of nonterminals, we need to verify that there exist $k, \ell \in \mathbb{Z}_m$ with $\rho_{\mathcal{G}'}\big(\langle i, j\rangle/\langle k, \ell\rangle\big) = g$ and $\rho_{\mathcal{G}'}\big(\langle k, \ell\rangle/\langle i', j'\rangle\big) = h$. Since $g = i + \ell \bmod m$ and $h = k + j' \bmod m$, we can conclude that $\ell = g - i \bmod m$ and $k = h - j' \bmod m$. Furthermore, assume we are given an arbitrary atom $\langle i, j\rangle$ and nonterminals $g, h \in \mathbb{Z}_m$, and want to find a category $\langle i, j\rangle/\langle k, \ell\rangle$ and an atom $\langle k, \ell\rangle$ such that $\rho_{\mathcal{G}'}\big(\langle i, j\rangle/\langle k, \ell\rangle\big) = g$ and $\rho_{\mathcal{G}'}\big(\langle k, \ell\rangle\big) = h$. From the definition of the relabeling we have $\rho_{\mathcal{G}'}\big(\langle k, \ell\rangle\big) = k$, so $k = h$ and $\ell = g - i \bmod m$.

It is straightforward to show that each derivation of $C_{\mathcal{G}'}$ relabels (via $\rho_{\mathcal{G}'}$) to a derivation of $\mathcal{G}'$ due to the definition of R and L. For the converse, suppose we would like to simulate a production $n \to \sigma(g, h) \in P$ and have already settled on category a/a' for n. We already argued that we can always find suitable preimages a/a'' and a''/a' that relabel to g and h, respectively. So for every derivation $d \in \mathcal{D}(\mathcal{G}')$ we can find a derivation of $C_{\mathcal{G}'}$ that relabels to d. Due to the fact that the categories occurring in derivation trees of $C_{\mathcal{G}'}$ cannot have higher order or arity greater than 1, they never leave the relevant domain of $\rho_{\mathcal{G}'}$. □

$$\dfrac{\dfrac{ax/(by)\quad by\alpha/c}{ax\alpha/c}\quad c}{ax\alpha} \xrightarrow{\text{R1}} \dfrac{ax/(by)\quad \dfrac{by\alpha/c\quad c}{by\alpha}}{ax\alpha} \qquad \dfrac{c\quad \dfrac{ax/(by)\quad by\alpha\backslash c}{ax\alpha\backslash c}}{ax\alpha} \xrightarrow{\text{R3}} \dfrac{ax/(by)\quad \dfrac{c\quad by\alpha\backslash c}{by\alpha}}{ax\alpha}$$

$$\dfrac{c\quad \dfrac{by\alpha\backslash c\quad ax\backslash(by)}{ax\alpha\backslash c}}{ax\alpha} \xrightarrow{\text{R2}} \dfrac{\dfrac{c\quad by\alpha\backslash c}{by\alpha}\quad ax\backslash(by)}{ax\alpha} \qquad \dfrac{\dfrac{by\alpha/c\quad ax\backslash(by)}{ax\alpha/c}\quad c}{ax\alpha} \xrightarrow{\text{R4}} \dfrac{\dfrac{by\alpha/c\quad c}{by\alpha}\quad ax\backslash(by)}{ax\alpha}$$

Figure 4.7: Rule schemes of [50] with $a, b \in A$, $x, y, \alpha \in \mathcal{A}(A)$, and $c \in \mathcal{C}(A)$.

Theorem 4.2.6 *The tree languages generatable by* 1-CCG *are exactly the regular tree languages.*

Because the yield languages of the regular tree languages are exactly the context-free languages, we obtain the following generalization[6] of Theorem 4.1.1.

6: Note that 0-CCG is included in 1-CCG.

Corollary 4.2.7 *The string languages generated by* 1-CCG *are exactly the ε-free context-free languages. Moreover, for each* 1-CCG $\mathcal{G}$ *the derivation tree language* $\mathcal{D}(\mathcal{G})$ *and the tree languages generatable by* $\mathcal{G}$ *are regular.*

4.2.1 Pure 1-CCG

In Section 5.2, we will see that pure k-CCG cannot even generate all local tree languages. Thus, pure 1-CCG has a reduced tree-generative capacity compared to 1-CCG with rule restrictions. Although it follows from Corollary 4.2.7 that pure 1-CCG cannot generate non-context-free languages, it is not immediately clear that it can generate all context-free languages. This is due to the existence of several transformation schemes that allow to change the order of consecutive application and non-application operations [50]. This reordering of the subtrees of derivation trees is also referred to as *tree rotation*. The transformation schemes are depicted in Figure 4.7. In pure CCG, their applicability cannot be prevented, such that for each derivation tree, also all derivation trees arising from the application of these transformations are valid as well. However, the following theorem demonstrates that pure 1-CCG can indeed still generate all (ε-free) context-free languages.

Theorem 4.2.8 *The string languages generated by pure* 1-CCG *are exactly the ε-free context-free languages.*

Proof. We use the classical construction for pure 0-CCG [8]. Given a CFG, a pure 0-CCG generating the same string language is con-

structed, such that lexicon entries contain only atomic arguments with leading backward slashes.[7] As a consequence, each derivation tree generated by a CCG with this lexicon can use only backward rules, and all secondary categories of application rules are atomic. Now consider some derivation tree of the pure 1-CCG with the same lexicon, such that the root category is an initial atomic category. If a composition rule appears in the derivation tree, we can find a position where a composition is followed by an application. This is because the root category, as it is atomic, is always obtained through application. We transform the derivation tree by repeated use of rule scheme R2 (see Figure 4.7) into a derivation tree that uses only application rules. Note that each use of the transformation rule eliminates a composition from the derivation tree, since y and α are empty due to the properties of the grammar. The transformation does not change the string that labels the leaves, since this particular rotation does not change the order of the three involved subtrees. This shows that the corresponding pure 0-CCG can generate the same string. Thus, the pure 1-CCG generates the same string language as the pure 0-CCG, and therefore the desired context-free language. □

7: Alternatively, the simpler construction outlined on page 36 can be employed, resulting in a grammar with all lexical categories containing only forward slashes.

5 Generative Power

5.1 Inclusion in the Simple Monadic Context-Free Tree Languages 54
5.2 Proper Inclusion for Pure CCG 64
5.3 Spine Grammar 65
5.4 Decomposition into Spines 70
5.5 Moore Push-Down Automata 74
5.6 CCG Construction . . 80
5.7 Strong Equivalence . . 94

In this chapter, we study the generative power of CCG with composition rules of arbitrary degree and rule restrictions, but consider also pure CCG. The first two sections are based on joint work with Marco Kuhlmann and Andreas Maletti, and the five remaining sections on joint work with Andreas Maletti (see Section 1.5). To keep the presentation simple, we assume without loss of generality that all secondary categories of rules in R are concrete categories of $\mathcal{C}(A)$ and that the targets of primary categories are specified; i.e., we disallow rules with category variables. Thus, each combinatory rule of degree k has one of the forms

$$\frac{ax/c \quad c|_1 c_1 \cdots |_k c_k}{ax|_1 c_1 \cdots |_k c_k} \qquad \frac{c|_1 c_1 \cdots |_k c_k \quad ax\backslash c}{ax|_1 c_1 \cdots |_k c_k}$$

with $a \in A$, $c \in \mathcal{C}(A)$, and $|_i \in D$ and $c_i \in \mathcal{C}(A)$ for every $i \in [k]$. The only remaining variable is the argument context variable x in the primary category.

We start by showing that the tree languages generatable by CCG are included in those generated by sCFTG (Section 5.1). Next, we show that this inclusion is proper for pure CCG (Section 5.2). The next four sections are devoted to showing the inverse direction of Section 5.1; namely, that each sCFTG can be simulated by a CCG. Our construction proceeds roughly as follows. We begin with a spine grammar, which is a variant of sCFTG that is strongly equivalent to TAG up to relabeling (Section 5.3). Then, we encode its spines using a context-free grammar (Section 5.4). These in turn can be represented by a special variant of push-down automata (Section 5.5). Finally, the runs of the push-down automaton are simulated by a CCG such that the stack operations of the automaton are realized by adding and removing arguments of categories (Section 5.6). In conclusion, each spine of primary categories of a CCG derivation tree corresponds to a run of the automaton, which in turn corresponds to a spine of the spine grammar. An overview of the construction is shown in Figure 5.4. Finally, in Section 5.7, we combine the results proven up to that point and discuss their ramifications. The main results are the strong equivalence of sCFTG and CCG, and as a consequence the strong equivalence of TAG and CCG. Another important finding that we would like to emphasize is the fact that CCG without ε-entries is as expressive as CCG with them. In addition, rule degree 2 and first-order categories are sufficient to give CCG its full expressive power.

5.1 Inclusion in the Simple Monadic Context-Free Tree Languages

In this section, we will show that the tree languages generatable by CCG are included in the simple monadic context-free tree languages. However, this is complicated by the presence of potentially infinitely many categories in the derivation trees $\mathcal{D}(\mathcal{G})$ for a CCG $\mathcal{G}$, while classical tree language theory only handles finitely many labels. The CCG $\mathcal{G}_5$ of Example 3.3.7 illustrates this problem. Therefore, instead of directly generating derivation trees, we construct an sCFTG that generates the *rule trees* of the CCG. Rule trees can represent derivation trees while using only a finite set of symbols. Using a category relabeling, they can be relabeled in the same manner as derivation trees.

We construct an sCFTG that uses nonterminals representing either categories or CCG rules. The derivation takes place by gradually extending a *spine* and by using branching productions to start new spines that are attached to superordinate spines. Nonterminals can only be replaced by terminal symbols when they represent either categories present in the lexicon or rules permitted by the CCG rule system.

Rule Trees

The idea behind rule trees is to label the internal nodes of these trees not by categories, but by the rules that are applied at them to obtain the respective categories, whereas the leaves are still labeled by lexical categories. We introduce the following abbreviations. We let $\mathrm{T} = T_{R,\emptyset}\big(L(\Sigma)\big)$ be the set of all *rule trees*. Moreover, for all alphabets N_1 and N_0 we define the set $\mathrm{SF}(N_1, N_0) = T_{R,N_1}\big(L(\Sigma) \cup N_0\big)$ of sentential forms of a sCFTG with unary nonterminals N_1 and nullary nonterminals N_0.

Definition 5.1.1 *Let $\mathcal{G} = (\Sigma, A, R, I, L)$ be a CCG. A tree $t \in \mathrm{T}$ is a* rule tree of $\mathcal{G}$ *if $\mathrm{cat}_{\mathcal{G}}(t) \in I$, where $\mathrm{cat}_{\mathcal{G}} \colon \mathrm{T} \to \mathcal{C}(A)$ is the partial mapping that is inductively defined by*

- $\mathrm{cat}_{\mathcal{G}}(c) = c$ *for all* $c \in L(\Sigma)$,
- $\mathrm{cat}_{\mathcal{G}}\big(\frac{ax/c \quad c\gamma}{ax\gamma}(t_1, t_2)\big) = a\alpha\gamma$ *for all trees* $t_1, t_2 \in \mathrm{T}$ *such that* $\mathrm{cat}_{\mathcal{G}}(t_1) = a\alpha/c$ *and* $\mathrm{cat}_{\mathcal{G}}(t_2) = c\gamma$, *and*
- $\mathrm{cat}_{\mathcal{G}}\big(\frac{c\gamma \quad ax\backslash c}{ax\gamma}(t_1, t_2)\big) = a\alpha\gamma$ *for all trees* $t_1, t_2 \in \mathrm{T}$ *such that* $\mathrm{cat}_{\mathcal{G}}(t_1) = c\gamma$ *and* $\mathrm{cat}_{\mathcal{G}}(t_2) = a\alpha\backslash c$

and is undefined for all other cases. The set of all rule trees of $\mathcal{G}$ is denoted by $\mathcal{R}(\mathcal{G})$.

Through rule trees, the derivation trees of a CCG can be encoded in a natural way while using only finitely many labels. More precisely, there is an (obvious) bijection between the derivation trees $\mathcal{D}(\mathcal{G})$ and the domain of the function $\mathrm{cat}_{\mathcal{G}}$. An example rule tree alongside the corresponding CCG derivation tree is depicted in Figure 5.1. Note, however, that all $t \in \mathcal{R}(\mathcal{G})$ have $\mathrm{cat}_{\mathcal{G}}(t) \in I$, which does not apply to the depicted rule tree.

sCFTG Construction

In the following, let $\mathcal{G} = (\Sigma, A, R, I, L)$ be a CCG. Our goal is to construct an sCFTG that generates exactly the rule tree language $\mathcal{R}(\mathcal{G})$. To this end, we first need to limit the number of categories. Let $k \in \mathbb{N}$ be the maximal arity of a category in

$$I \cup L(\Sigma) \cup \left\{ c\gamma \;\middle|\; \frac{ax/c \quad c\gamma}{ax\gamma} \in R \right\} \cup \left\{ c\gamma \;\middle|\; \frac{c\gamma \quad ax\backslash c}{ax\gamma} \in R \right\} ,$$

i.e., the maximal arity of the categories that occur as initial category, in the lexicon, or as the secondary category of a rule of R. Roughly speaking, the constructed sCFTG will use the categories $\mathcal{C}_L(A,k)$ as nullary nonterminals and tuples $\langle a, |c, \gamma \rangle$ consisting of an atomic category $a \in A$, a single argument $|c \in \mathrm{args}(L)$, and an argument context $\gamma \in \mathcal{A}_L(A,k)$ as unary nonterminals.[1] The unary nonterminals represent rules, where a is the target of the primary category, $|c$ is the bridging argument, and γ is the excess. Recall that we write substitutions $\alpha[t]$ as $t\alpha$ for $\alpha \in \mathcal{A}(A)$ and $t \in \mathcal{C}(A) \cup \mathcal{A}(A)$.

1: Recall that $\mathcal{C}_L(A,k)$ and $\mathcal{A}_L(A,k)$ may only use arguments in $\mathrm{args}(L)$, which are those present in the lexicon (see Section 3.3.3).

Definition 5.1.2 *The sCFTG* $\mathcal{G}' = \big(N_1 \cup N_0, R \cup L(\Sigma), S, P\big)$ *is given by*

- $N_1 = \big\{\langle a, |c, \gamma\rangle \mid a \in A, |c \in \mathit{args}(L), \gamma \in \mathcal{A}_L(A,k)\big\}$ *and* $N_0 = \big\{\langle c \rangle \mid c \in \mathcal{C}_L(A,k)\big\}$,
- $S = \big\{\langle a_0 \rangle \mid a_0 \in I\big\}$, *and*
- *the set* $P = P_1 \cup P_2 \cup P_3 \cup P_4 \cup P_5$ *of productions with*

$$P_1 = \big\{\langle c \rangle \to c \mid c \in L(\Sigma)\big\} \tag{5.1}$$

$$P_2 = \left\{\langle a, /c, \gamma\rangle \to r\big(\square, \langle c\gamma\rangle\big) \;\middle|\; r = \frac{ax/c \quad c\gamma}{ax\gamma} \in R\right\} \tag{5.2}$$

$$P_3 = \left\{\langle a, \backslash c, \gamma\rangle \to r\big(\langle c\gamma\rangle, \square\big) \;\middle|\; r = \frac{c\gamma \quad ax\backslash c}{ax\gamma} \in R\right\} \tag{5.3}$$

$$P_4 = \big\{\langle a\alpha\gamma\rangle \to \langle a, |c, \gamma\rangle\big(\langle a\alpha|c\rangle\big) \tag{5.4}$$
$$\mid a \in A,\ \alpha, \gamma \in \mathcal{A}_L(A),\ |c \in \mathit{args}(L),\ |\alpha| < k,\ |\alpha\gamma| \le k\big\}$$

$$P_5 = \big\{\langle a, |c, \gamma\rangle \to \langle a, |'c', \square\rangle\big(\langle a, |c, \gamma|'c'\rangle(\square)\big) \tag{5.5}$$
$$\mid a \in A,\ |c, |'c' \in \mathit{args}(L),\ \gamma \in \mathcal{A}_L(A, k-1)\big\}$$

We still have to establish that $\mathcal{G}'$ indeed generates exactly $\mathcal{R}(\mathcal{G})$. This will be achieved by showing both inclusions in the next chain of lemmas.

Only Rule Trees are Generated

We will start by showing that the sCFTG $\mathcal{G}'$ can only generate valid rule trees of $\mathcal{G}$.

Lemma 5.1.3 $\mathcal{T}(\mathcal{G}') \subseteq \mathcal{R}(\mathcal{G})$

Proof. We will start with an auxiliary statement. For all sentential forms $\xi \in \mathrm{SF}(N_1, N_0)$ and $t \in \mathrm{T}$ such that $\xi \Rightarrow^+_{\mathcal{G}'} t$, we prove that

(i) if $\xi(\varepsilon) = \langle c \rangle \in N_0$, then $\mathrm{cat}_{\mathcal{G}}(t) = c$, and
(ii) if $\xi(\varepsilon) = \langle a, |c, \gamma \rangle \in N_1$, $\xi|_1 \in \mathrm{T}$, and $\mathrm{cat}_{\mathcal{G}}(\xi|_1) = a\alpha|c$ with $\alpha \in \mathcal{A}_L(A)$, then $\mathrm{cat}_{\mathcal{G}}(t) = a\alpha\gamma$.

Note that the statement does not concern trees ξ with $\xi(\varepsilon) \notin N_0 \cup N_1$. For the remaining trees, we prove this statement by induction on the length of the derivation.

We have $\xi \Rightarrow_{\mathcal{G}',\varepsilon} \zeta \Rightarrow^{\ell}_{\mathcal{G}'} t$ for some $\zeta \in \mathrm{SF}(N_1, N_0)$ and $\ell \in \mathbb{N}$, where we applied the first derivation step at the root and $\Rightarrow^{\ell}_{\mathcal{G}'}$ is the ℓ-fold composition of $\Rightarrow_{\mathcal{G}'}$ with itself. We distinguish five cases based on the production $p \in P$ used at the root in the first derivation step:

(1) If $p = \langle c \rangle \to c \in P_1$ is a production of type (5.1), then $c \in L(\Sigma)$, $\xi = \langle c \rangle$, and $\zeta = t = c$. Since $c \in L(\Sigma)$, we trivially have $\mathrm{cat}_{\mathcal{G}}(t) = c$, which proves statement (i).[2]

2: This also proves statement (ii) in this case since its precondition is not fulfilled. We omit such obvious observations in the next cases.

(2) If $p = \langle a, /c, \gamma \rangle \to r(\square, \langle c\gamma \rangle) \in P_2$ is a production of type (5.2), we have $\xi(\varepsilon) = \langle a, /c, \gamma \rangle$ and we only need to prove statement (ii). Thus, let $\xi|_1 \in \mathrm{T}$ and $\mathrm{cat}_{\mathcal{G}}(\xi|_1) = a\alpha/c$ for some $\alpha \in \mathcal{A}_L(A)$. From derivation $\zeta = r(\xi|_1, \langle c\gamma \rangle) \Rightarrow^{\ell}_{\mathcal{G}'} t$, we can conclude that $\langle c\gamma \rangle \Rightarrow^{\ell'}_{\mathcal{G}'} t|_2$ for some $1 \le \ell' < \ell$. The latter yields $\mathrm{cat}_{\mathcal{G}}(t|_2) = c\gamma$ by the induction hypothesis. From the facts $r = \frac{ax/c \quad c\gamma}{ax\gamma} \in R$, $\mathrm{cat}_{\mathcal{G}}(t|_1) = a\alpha/c$, and $\mathrm{cat}_{\mathcal{G}}(t|_2) = c\gamma$, we conclude that $\mathrm{cat}_{\mathcal{G}}(t) = a\alpha\gamma$.
(3) If $p = \langle a, \backslash c, \gamma \rangle \to r(\langle c\gamma \rangle, \square) \in P_3$ is a production of type (5.3), we need to prove statement (ii), which can be done in the same way as in the previous case (2).
(4) If $p = \langle a\alpha\gamma \rangle \to \langle a, |c, \gamma \rangle(\langle a\alpha|c \rangle) \in P_4$ is a production of type (5.4), we need to prove statement (i). By context-freeness, we can rearrange the derivation $\zeta \Rightarrow^{\ell}_{\mathcal{G}'} t$ such that

$$\zeta = \langle a, |c, \gamma \rangle(\langle a\alpha|c \rangle) \Rightarrow^{\ell'}_{\mathcal{G}'} \langle a, |c, \gamma \rangle(t') \Rightarrow^{\ell''}_{\mathcal{G}'} t$$

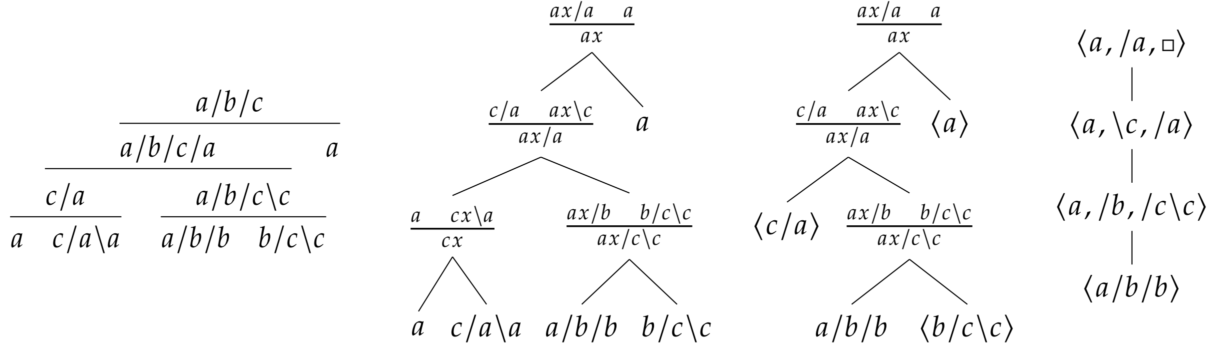


Figure 5.1: CCG derivation tree (without lexical entries), corresponding rule tree t, spinal(t), and its encoding enc(t). The derivation tree is depicted with the root at the top to make the correspondence to the other trees more apparent.

for some $t' \in \mathrm{T}$ and $\ell', \ell'' \geq 1$ such that $\ell = \ell' + \ell''$. Consequently, we have a subderivation $\langle a\alpha|c\rangle \Rightarrow^{\ell'}_{\mathcal{G}'} t'$, from which we conclude that $\mathrm{cat}_{\mathcal{G}}(t') = a\alpha|c$ by the induction hypothesis. Now we established the preconditions of statement (ii) for the subderivation $\langle a, |c, \gamma\rangle(t') \Rightarrow^{\ell''}_{\mathcal{G}'} t$, so $\mathrm{cat}_{\mathcal{G}}(t) = a\alpha\gamma$ by the induction hypothesis.

(5) If $p = \langle a, |c, \gamma\rangle \to \langle a, |'c', \Box\rangle(\langle a, |c, \gamma|'c'\rangle(\Box)) \in P_5$ is a production of type (5.5), then we need to prove statement (ii). Let $\xi|_1 \in \mathrm{T}$ and $\mathrm{cat}_{\mathcal{G}}(\xi|_1) = a\alpha|c$ for some $\alpha \in \mathcal{A}_L(A)$. We can again reorder the derivation $\zeta \Rightarrow^{\ell}_{\mathcal{G}'} t$ such that

$$\zeta = \langle a, |'c', \Box\rangle(\langle a, |c, \gamma|'c'\rangle(\xi|_1)) \Rightarrow^{\ell'}_{\mathcal{G}'} \langle a, |'c', \Box\rangle(t') \Rightarrow^{\ell''}_{\mathcal{G}'} t$$

for some $t' \in \mathrm{T}$ and $\ell', \ell'' \geq 1$ such that $\ell = \ell' + \ell''$. In the first subderivation we find $\langle a, |c, \gamma|'c'\rangle(\xi|_1) \Rightarrow^{\ell'}_{\mathcal{G}'} t'$. Since $\xi|_1 \in \mathrm{T}$ and $\mathrm{cat}_{\mathcal{G}}(\xi|_1) = a\alpha|c$, we meet the requirements of statement (ii), obtaining $\mathrm{cat}_{\mathcal{G}}(t') = a\alpha\gamma|'c'$ by the induction hypothesis. Once more we now have established that $t' \in \mathrm{T}$ and $\mathrm{cat}_{\mathcal{G}}(t') = a\alpha\gamma|'c'$, so we satisfy the requirements of statement (ii) of the induction hypothesis applied to the second subderivation $\langle a, |'c', \Box\rangle(t') \Rightarrow^{\ell''}_{\mathcal{G}'} t$. Consequently, $\mathrm{cat}_{\mathcal{G}}(t) = a\alpha\gamma$.

This completes the proof of the auxiliary statement. We can apply the auxiliary statement to derive that $\mathrm{cat}_{\mathcal{G}}(t) = a_0 \in I$ for all $t \in \mathrm{T}$ and $\langle a_0\rangle \in S$ such that $\langle a_0\rangle \Rightarrow^{+}_{\mathcal{G}'} t$. Consequently, $\mathcal{T}(\mathcal{G}') \subseteq \mathcal{R}(\mathcal{G})$. □

Decomposition of Rule Trees

For the converse, we decompose and encode rule trees in a more compact manner. These encodings will help us to structure the derivation of rule trees, as we can use them as components and intermediate steps of the complete derivation. First, we translate a

rule tree into its *primary spine form*. It uses terminal symbols when following the spine in the direction of primary categories, but the subtrees located next to the spine are abbreviated to nullary nonterminals.

Definition 5.1.4 *The* primary spine form *of a rule tree is defined by the function* spinal: $\mathrm{T} \longrightarrow T_{R,\emptyset}\big(L(\Sigma) \cup N_0\big)$ *that is given by*

$$\begin{aligned} \mathrm{spinal}\big(b\big) &= b \\ \mathrm{spinal}\Big(\frac{ax/c \quad c\gamma}{ax\gamma}(t_1,t_2)\Big) &= \frac{ax/c \quad c\gamma}{ax\gamma}\big(\mathrm{spinal}(t_1), \langle c\gamma\rangle\big) \\ \mathrm{spinal}\Big(\frac{c\gamma \quad ax\backslash c}{ax\gamma}(t_1,t_2)\Big) &= \frac{c\gamma \quad ax\backslash c}{ax\gamma}\big(\langle c\gamma\rangle, \mathrm{spinal}(t_2)\big) \end{aligned}$$

where $a \in A$, $b \in L(\Sigma)$, $|c \in \mathit{args}(L)$, $\gamma \in \mathcal{A}_L(A,k)$, *and* $t_1, t_2 \in \mathrm{T}$.

Additionally, we encode rule trees using only the nonterminals of $\mathcal{G}'$. These represent exactly the nodes of the rule tree that are located on the spine following the direction of primary categories.

Definition 5.1.5 *The* encoding *of a rule tree is defined by the function* enc: $\mathrm{T} \longrightarrow T_{\emptyset,N_1}(N_0)$ *that is given by*

$$\begin{aligned} \mathrm{enc}(b) &= \langle b\rangle \\ \mathrm{enc}\Big(\frac{ax/c \quad c\gamma}{ax\gamma}(t_1,t_2)\Big) &= \langle a, /c, \gamma\rangle\big(\mathrm{enc}(t_1)\big) \\ \mathrm{enc}\Big(\frac{c\gamma \quad ax\backslash c}{ax\gamma}(t_1,t_2)\Big) &= \langle a, \backslash c, \gamma\rangle\big(\mathrm{enc}(t_2)\big) \end{aligned}$$

where $a \in A$, $b \in L(\Sigma)$, $|c \in \mathit{args}(L)$, $\gamma \in \mathcal{A}_L(A,k)$, *and* $t_1, t_2 \in \mathrm{T}$.

The primary spine form and the encoding are demonstrated in Figure 5.1. We will show that $\langle \mathrm{cat}_{\mathcal{G}}(t)\rangle \Rightarrow^*_{\mathcal{G}'} \mathrm{enc}(t) \Rightarrow^*_{\mathcal{G}'} \mathrm{spinal}(t)$ for every $t \in \mathrm{T}$ with $\mathrm{cat}_{\mathcal{G}}(t) \in \mathcal{C}_L(A,k)$ in the following sequence of lemmas. We begin by proving the second, easier part.

Deriving the Primary Spine Form

Lemma 5.1.6 *For every* $t \in \mathrm{T}$ *with* $\mathrm{cat}_{\mathcal{G}}(t) \in \mathcal{C}_L(A,k)$ *there exists a derivation* $\mathrm{enc}(t) \Rightarrow^*_{\mathcal{G}'} \mathrm{spinal}(t)$.

Proof. The proof is by induction on t. In the induction base, we have $t \in L(\Sigma)$. Therefore, we have $\mathrm{enc}(t) = \langle t\rangle$ and $\mathrm{spinal}(t) = t$. Since $t \in L(\Sigma)$, we can apply a production of type (5.1) to obtain $\mathrm{enc}(t) = \langle t\rangle \Rightarrow_{\mathcal{G}'} t = \mathrm{spinal}(t)$ as desired.

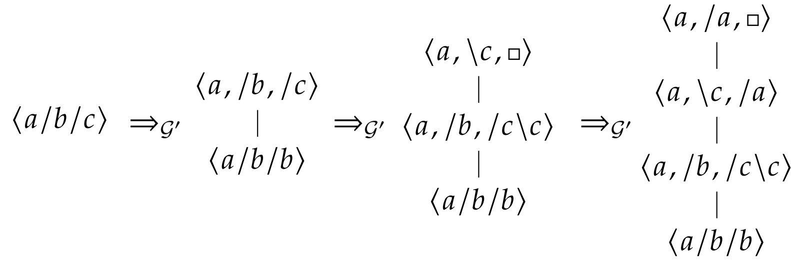


Figure 5.2: Derivation of the encoding enc(t) depicted in Figure 5.1. In each step the nonterminal at the root is expanded.

In the induction step, let $t = r(t_1, t_2)$ with $r = \frac{ax/c \quad c\gamma}{ax\gamma} \in R$. The case of a backward composition is analogous. Then we have $\mathrm{enc}(t) = \langle a, /c, \gamma\rangle(\mathrm{enc}(t_1))$ and $\mathrm{spinal}(t) = r(\mathrm{spinal}(t_1), \langle c\gamma\rangle)$. Consequently,

$$\begin{aligned}\mathrm{enc}(t) &= \langle a, /c, \gamma\rangle\big(\mathrm{enc}(t_1)\big)\\ &\Rightarrow^*_{\mathcal{G}'} \langle a, /c, \gamma\rangle\big(\mathrm{spinal}(t_1)\big)\\ &\Rightarrow_{\mathcal{G}'} r\big(\mathrm{spinal}(t_1), \langle c\gamma\rangle\big) = \mathrm{spinal}(t) \ ,\end{aligned}$$

where we used the induction hypothesis in the first step and then a production of type (5.2). □

Deriving the Encoding

We now turn to the first part and show that $\langle \mathrm{cat}_{\mathcal{G}}(t)\rangle \Rightarrow^*_{\mathcal{G}'} \mathrm{enc}(t)$ for every $t \in \mathrm{T}$ with $\mathrm{cat}_{\mathcal{G}}(t) \in \mathcal{C}_L(A, k)$. This will be dealt with in the next two lemmas. For this purpose, we need to introduce some additional notation. Recall the sets N_1 and N_0 of nonterminals from Definition 5.1.2. We let $\mathrm{Enc} = T_{\emptyset, N_1}(N_0)$ be the set of all potential encodings. Encodings are essentially strings, so $\mathrm{pos}(e) \subseteq \{1\}^*$ for all $e \in \mathrm{Enc}$. Instead of a position 1^h we write just h in an encoding. Moreover, we write $|e|$ instead of $|\mathrm{pos}(e)|$. In other words, we identify the set $\mathrm{pos}(e)$ with the corresponding set $\mathbb{Z}_{|e|}$ of nonnegative integers.

Definition 5.1.7 *Let $e \in \mathrm{Enc}$ be an encoding with* $\mathrm{pos}(e) = \mathbb{Z}_{\ell+1}$ *such that $e(\ell) = \langle a\alpha\rangle$ and $e(i) = \langle a_i, |_i c_i, \gamma_i\rangle$ for all $i < \ell$. It is* consistent *if*

- *$a = a_i$ for all $i < \ell$, and*
- *there exist $\alpha_0, \ldots, \alpha_{\ell-1}$ such that $\alpha = \alpha_{\ell-1}|_{\ell-1}c_{\ell-1}$ and $\alpha_i\gamma_i = \alpha_{i-1}|_{i-1}c_{i-1}$ for all $i \in [\ell - 1]$.*

We let $\mathrm{cat}_{\mathcal{G}}(e|_\ell) = a\alpha$ *and* $\mathrm{cat}_{\mathcal{G}}(e|_i) = a\alpha_i\gamma_i$ *for all $i < \ell$.*

Roughly speaking, an encoding is consistent if it represents a possible sequence of rule applications along some spine of primary categories.

Definition 5.1.8 *Let $e, e' \in$ Enc be two encodings. We write $e \prec e'$ if there exist positions $i \in \text{pos}(e)$ and $i' \in \text{pos}(e')$ with $i \leq i'$ such that*

- $e(j) = \langle a_j, |_j c_j, \square\rangle$ *for all* $j < i$, *and*
- $e|_i = \langle a, |c, \gamma\rangle(\overline{e})$ *and* $e'|_{i'} = \langle a, |c, \gamma'\rangle(\overline{e})$ *with* $\gamma \sqsubset \gamma'$.

Thus, encodings $e, e' \in$ Enc with $e \prec e'$ have the same subtree $\overline{e}$ below position i resp. i', all labels of e at positions $j < i$ have $\square$ in the third component, and at position i resp. i', the third component γ in e is a proper prefix of the third component γ' in e'.[3] It is clear that $\prec$ is a strict partial order (irreflexive and transitive) on encodings. Indeed the positions i and i' that demonstrate $e \prec e'$ are unique.

3: In other words, γ is a strict subtree on the left spine of γ'(see Section 3.1).

Finally, we let $f\colon \text{Enc}^2 \to \mathbb{Z}$ be such that $f(e, e') = |e'| - |e|$ for all $e, e' \in$ Enc. It is obvious that $f(e, e') \in \mathbb{N}$ provided that $e \prec e'$.

Lemma 5.1.9 *Let $e, e' \in$ Enc be two consistent encodings such that $\text{cat}_{\mathcal{G}}(e) = \text{cat}_{\mathcal{G}}(e')$ and $e \prec e'$. Then $e \Rightarrow^+_{\mathcal{G}'} e'$.*

Proof. Let $i \leq i'$ be the unique positions required to show that $e \prec e'$, and $e|_i = \langle a, |c, \gamma\rangle(\overline{e})$ and $e'|_{i'} = \langle a, |c, \gamma'\rangle(\overline{e})$. We additionally let $a\beta = \text{cat}_{\mathcal{G}}(e) = \text{cat}_{\mathcal{G}}(e')$ and $a\alpha|c = \text{cat}_{\mathcal{G}}(\overline{e})$. Then obviously $\text{cat}_{\mathcal{G}}(e|_i) = a\alpha\gamma$. Moreover, $i = |\alpha\gamma| - |\beta|$ because the third component is $\square$ for all labels at positions strictly smaller than i, which yields that $\text{cat}_{\mathcal{G}}(e|_{i-j}) = (a\alpha\gamma)|_j$ for all $j \leq i$. Similarly, we have $\text{cat}_{\mathcal{G}}(e'|_{i'}) = a\alpha\gamma'$ with $|\alpha\gamma'| > |\alpha\gamma|$. Since we can only remove a single argument in each step, we obtain that $|\alpha\gamma'| - |\beta| \leq i'$.

We now prove the statement by induction on $f(e, e')$. In the induction base, we assume that $f(e, e') = |e'| - |e| = 0$. Consequently, we have $i = i'$ and $|\alpha\gamma'| - |\beta| \leq i' = i = |\alpha\gamma| - |\beta| < |\alpha\gamma'| - |\beta|$, which is a contradiction. Hence this case cannot occur.

In the induction step, let $f(e, e') = |e'| - |e| > 0$. Thus, $i < i'$. Let $\gamma' = \gamma|'c'\gamma''$ for some $|'c' \in \text{args}(L)$ and $\gamma'' \in \mathcal{A}_L(A, k-1)$. Using an application of a production of type (5.5) we obtain

$$e|_i = \langle a, |c, \gamma\rangle(\overline{e}) \Rightarrow_{\mathcal{G}'} \langle a, |'c', \square\rangle\big(\langle a, |c, \gamma|'c'\rangle(\overline{e})\big) = \hat{e} \enspace .$$

Let $e'' = e[\hat{e}]_i$, which immediately yields $|e''| > |e|$. Then e'' is again a consistent encoding because $\text{cat}_{\mathcal{G}}(\hat{e}) = a\alpha\gamma$, which is also the category of the replaced subtree $e|_i$. Consequently, the newly constructed encoding e'' has the same category as e and e'.

Next we prove that $e'' \preceq e'$. If $e'' = e'$, then trivially $e'' \preceq e'$. Thus, let $e'' \neq e'$. Since $e''|_{i+2} = e'|_{i'+1}$, let $j'' \leq i+1$ be the largest integer such that $e''(j'') \neq e'(j')$, where $j' = i' - (i+1) + j''$. Clearly, such

an integer j'' must exist because $e'' \neq e'$. Now we prove by case analysis on $j'' \leq i+1$ that $e'' \preceq e'$.

- If $j'' = i+1$, then $j' = i'$. Since the labels of e'' at $j'' = i+1$ and of e' at $j' = i'$ differ, although their first two components are the same, we obtain that $\gamma|'c' \neq \gamma' = \gamma|'c'\gamma''$ and thus $\gamma'' \neq \square$. Then trivially $e'' \prec e'$ using the positions $j'' = i+1$ and $j' = i'$, for which we know $i+1 \leq i'$ and all labels of e'' at strict prefixes of $i+1$ have $\square$ in the third component (for all positions strictly smaller than i this is true since the labels of e'' and e coincide and for i it is true by the definition of e'').
- Otherwise, we have $j'' < i+1$. Let $e''|_{j''} = \langle a, |''c'', \square\rangle(h'')$ and $e'|_{j'} = \langle a, |'''c''', \alpha'\rangle(h')$. Obviously, we have $h'' = h'$ because we selected the maximal j''. Consequently, we also have $|'''c''' = |''c''$ by consistency. Since the labels are different, we must have $\square \sqsubset \alpha'$. Then $e'' \prec e'$ using the positions j'' and j', for which we know that $j'' = j' - i' + (i+1) \leq j'$ because $i+1 \leq i'$. Moreover, all labels of e'' at strict prefixes of $j'' \leq i$ have $\square$ in the third component since those labels coincide in e'' and e.

Hence $e'' \preceq e'$.

Now we return to the main statement. If we have $e'' = e'$, then clearly $e \Rightarrow^+_{\mathcal{G}'} e'' = e'$. Otherwise, we have consistent encodings e'' and e' with $\mathrm{cat}_{\mathcal{G}}(e'') = \mathrm{cat}_{\mathcal{G}}(e')$ such that $e'' \prec e'$. Since $|e''| > |e|$, we additionally have $f(e'', e') = |e'| - |e''| < |e'| - |e| = f(e, e')$. Consequently, we can apply the induction hypothesis to e'' and e' and obtain that there is a derivation $e'' \Rightarrow^+_{\mathcal{G}'} e'$, which together with $e \Rightarrow_{\mathcal{G}'} e''$ yields $e \Rightarrow^+_{\mathcal{G}'} e'$ as desired. □

It is obvious that for every $t \in \mathrm{T}$ with $\mathrm{cat}_{\mathcal{G}}(t) \in \mathcal{C}_L(A)$ the encoding $\mathrm{enc}(t)$ is consistent and has the same category $\mathrm{cat}_{\mathcal{G}}(t)$. For the proof of $\langle \mathrm{cat}_{\mathcal{G}}(t)\rangle \Rightarrow^*_{\mathcal{G}'} \mathrm{enc}(t)$ for all $t \in \mathrm{T}$ with $\mathrm{cat}_{\mathcal{G}}(t) \in \mathcal{C}_L(A, k)$, we define a modification of the encoding.

Definition 5.1.10 *The function* $\mathrm{enc}' \colon \mathrm{T} \longrightarrow T_{\emptyset, N_1}(N_0)$ *is defined as follows. For all* $t \in \mathrm{T}$ *with* $\mathrm{cat}_{\mathcal{G}}(t) \in \mathcal{C}_L(A, k)$*, we let*

$$\mathrm{enc}'(t) = \langle \mathrm{cat}_{\mathcal{G}}(t)\rangle \enspace .$$

Otherwise, we let

$$\mathrm{enc}'\Big(\frac{ax/c \quad c\gamma}{ax\gamma}(t_1, t_2)\Big) = \langle a, /c, \gamma\rangle\big(\mathrm{enc}'(t_1)\big)$$
$$\mathrm{enc}'\Big(\frac{c\gamma \quad ax\backslash c}{ax\gamma}(t_1, t_2)\Big) = \langle a, \backslash c, \gamma\rangle\big(\mathrm{enc}'(t_2)\big)$$

where $a \in A$, $|c \in \mathit{args}(L)$, $\gamma \in \mathcal{A}_L(A, k)$, *and* $t_1, t_2 \in \mathrm{T}$.

As for the encoding $\mathrm{enc}(t)$, for every $t \in \mathrm{T}$ with $\mathrm{cat}_{\mathcal{G}}(t) \in \mathcal{C}_L(A)$, the encoding $\mathrm{enc}'(t)$ is consistent and has the same category $\mathrm{cat}_{\mathcal{G}}(t)$. Note how this is different from the previous encoding $\mathrm{enc}(t)$. While before, we only added a nullary nonterminal if the input consisted of a single node labeled by $c \in L(\Sigma)$, we now compress a complete subtree whose category is in $\mathcal{C}_L(A,k)$ to a nullary nonterminal. In the following, we will show that we can derive the original encoding from this shortened variant.

Lemma 5.1.11 *For every* $t \in \mathrm{T}$ *with* $\mathrm{cat}_{\mathcal{G}}(t) \in \mathcal{C}_L(A)$ *there exists a derivation* $\mathrm{enc}'(t) \Rightarrow^*_{\mathcal{G}'} \mathrm{enc}(t)$.

Proof. We prove it by induction on t. In the induction base, we let $t \in L(\Sigma)$. As a consequence, we have $t = \mathrm{cat}_{\mathcal{G}}(t) \in \mathcal{C}_L(A,k)$ and $\mathrm{enc}'(t) = \langle \mathrm{cat}_{\mathcal{G}}(t)\rangle = \mathrm{enc}(t)$, which proves the induction base.

In the induction step, let $t = r(t_1, t_2)$ for some rule $r \in R$ and $t_1, t_2 \in \mathrm{T}$. We only consider forward compositions. Thus, let $r = \frac{ax/c \quad c\gamma}{ax\gamma}$. By assumption, we have $\mathrm{cat}_{\mathcal{G}}(t) \in \mathcal{C}_L(A)$, so let $\mathrm{cat}_{\mathcal{G}}(t) = a\alpha\gamma$ for some $\alpha \in \mathcal{A}_L(A)$. Now we distinguish three cases depending on the arities of the categories of t and t_1:

- Suppose that $\mathrm{cat}_{\mathcal{G}}(t) = a\alpha\gamma \notin \mathcal{C}(A,k)$. Then

$$\begin{aligned}\mathrm{enc}'(t) &= \langle a, /c, \gamma\rangle\big(\mathrm{enc}'(t_1)\big)\\ &\Rightarrow^*_{\mathcal{G}'} \langle a, /c, \gamma\rangle\big(\mathrm{enc}(t_1)\big) = \mathrm{enc}(t)\ ,\end{aligned}$$

  where we used the induction hypothesis applied to t_1.
- Now suppose that $\mathrm{cat}_{\mathcal{G}}(t) = a\alpha\gamma \in \mathcal{C}_L(A,k)$ and also suppose that $\mathrm{cat}_{\mathcal{G}}(t_1) = a\alpha/c \notin \mathcal{C}(A,k)$. Let $\langle a\beta|c'\rangle$ be the leaf of $\mathrm{enc}'(t_1)$, so $a\beta|c' \in \mathcal{C}_L(A,k)$. Note that the categories of subtrees of $\mathrm{enc}'(t_1)$ for all strictly smaller positions are not in $\mathcal{C}(A,k)$. Consequently, we have $a\beta \sqsubset a\alpha\gamma$, so let $\gamma' \in \mathcal{A}_L(A,k)$ be such that $a\beta\gamma' = a\alpha\gamma$. In addition,

$$\mathrm{enc}'(t) = \langle a\alpha\gamma\rangle \Rightarrow_{\mathcal{G}'} \langle a, |c', \gamma'\rangle\big(\langle a\beta|c'\rangle\big) = e\ .$$

  Clearly, $e = \langle a, |c', \gamma'\rangle\big(\langle a\beta|c'\rangle\big)$ is consistent and has category $a\alpha\gamma$, which is also true for $e' = \langle a, /c, \gamma\rangle\big(\mathrm{enc}'(t_1)\big)$. Next we show that $e \prec e'$. Let $\ell = |e'| - 1$. It is clear that $e|_1 = \langle a\beta|c'\rangle = e'|_\ell$ and due to e' being consistent also $e'_{\ell-1} = \langle a, |c', \gamma''\rangle$ for some $\gamma'' \in \mathcal{A}_L(A,k)$. It remains to prove that $\gamma' \sqsubset \gamma''$. For all non-zero positions strictly smaller than $\ell - 1$, the encoding e' has subtrees whose respective categories are not in $\mathcal{C}(A,k)$. Hence $a\beta\gamma' = a\alpha\gamma \in \mathcal{C}_L(A,k)$ must be a proper prefix of all those categories, including $a\beta\gamma''$. Consequently, we have $e \prec e'$ for these consistent

encodings with $\mathrm{cat}_{\mathcal{G}}(e) = \mathrm{cat}_{\mathcal{G}}(e')$, so we use Lemma 5.1.9 to conclude that $e \Rightarrow^{+}_{\mathcal{G}'} e'$. Thus, in summary we have

$$\begin{aligned}\mathrm{enc}'(t) &= \langle a\alpha\gamma\rangle \\ &\Rightarrow_{\mathcal{G}'} \langle a, |c', \gamma'\rangle\big(\langle a\beta|c'\rangle\big) = e \\ &\Rightarrow^{+}_{\mathcal{G}'} e' = \langle a, /c, \gamma\rangle\big(\mathrm{enc}'(t_1)\big) \\ &\Rightarrow^{*}_{\mathcal{G}'} \langle a, /c, \gamma\rangle\big(\mathrm{enc}(t_1)\big) = \mathrm{enc}(t)\enspace .\end{aligned}$$

- Finally, suppose that $\mathrm{cat}_{\mathcal{G}}(t) = a\alpha\gamma \in \mathcal{C}_L(A, k)$ and also suppose that $\mathrm{cat}_{\mathcal{G}}(t_1) = a\alpha/c \in \mathcal{C}_L(A, k)$. Then

$$\begin{aligned}\mathrm{enc}'(t) &= \langle a\alpha\gamma\rangle \\ &\Rightarrow_{\mathcal{G}'} \langle a, /c, \gamma\rangle\big(\langle a\alpha/c\rangle\big) = \langle a, /c, \gamma\rangle\big(\mathrm{enc}'(t_1)\big) \\ &\Rightarrow^{*}_{\mathcal{G}'} \langle a, /c, \gamma\rangle\big(\mathrm{enc}(t_1)\big) = \mathrm{enc}(t)\enspace .\end{aligned}$$

This concludes the proof of the statement. A particular case is the desired derivation $\langle \mathrm{cat}_{\mathcal{G}}(t)\rangle \Rightarrow^{*}_{\mathcal{G}'} \mathrm{enc}(t)$ for every $t \in \mathrm{T}$ with $\mathrm{cat}_{\mathcal{G}}(t) \in \mathcal{C}_L(A, k)$. □

An example of a derivation of the form $\langle \mathrm{cat}_{\mathcal{G}}(t)\rangle \Rightarrow^{*}_{\mathcal{G}'} \mathrm{enc}(t)$ is presented in Figure 5.2.

Deriving the Rule Tree

Combining Lemmas 5.1.6 and 5.1.11, we can conclude that there is a derivation $\langle \mathrm{cat}_{\mathcal{G}}(t)\rangle \Rightarrow^{*}_{\mathcal{G}'} \mathrm{enc}(t) \Rightarrow^{*}_{\mathcal{G}'} \mathrm{spinal}(t)$ for every $t \in \mathrm{T}$ with $\mathrm{cat}_{\mathcal{G}}(t) \in \mathcal{C}_L(A, k)$. This will prove extremely useful in the following.

Lemma 5.1.12 $\mathcal{R}(\mathcal{G}) \subseteq \mathcal{T}(\mathcal{G}')$

Proof. We first show the auxiliary statement that for every rule tree $t \in \mathrm{T}$ with $\mathrm{cat}_{\mathcal{G}}(t) \in \mathcal{C}_L(A, k)$ we have $\langle \mathrm{cat}_{\mathcal{G}}(t)\rangle \Rightarrow^{*}_{\mathcal{G}'} t$. The proof is by induction. In the induction base we have $t \in L(\Sigma)$, which also yields $t = \mathrm{cat}_{\mathcal{G}}(t) \in \mathcal{C}_L(A, k)$, and thus $\langle t\rangle \Rightarrow_{\mathcal{G}'} t$ by a production of type (5.1). In the induction step, we use Lemmas 5.1.6 and 5.1.11 to obtain $\langle c\rangle \Rightarrow^{*}_{\mathcal{G}'} \mathrm{spinal}(t)$, and we apply the induction hypothesis to the nullary nonterminals present in $\mathrm{spinal}(t)$ to conclude the proof of the auxiliary statement. With the help of the auxiliary statement, we immediately obtain that for every rule tree $t \in \mathcal{R}(\mathcal{G})$ we have $\langle \mathrm{cat}_{\mathcal{G}}(t)\rangle \Rightarrow^{*}_{\mathcal{G}'} t$ because $\mathrm{cat}_{\mathcal{G}}(t) \in I$. Moreover, $\langle \mathrm{cat}_{\mathcal{G}}(t)\rangle \in S$ and hence $t \in \mathcal{T}(\mathcal{G}')$. □

Theorem 5.1.13 *The rule tree language $\mathcal{R}(\mathcal{G})$ of a CCG $\mathcal{G}$ can be generated by an sCFTG.*

5.2 Proper Inclusion for Pure CCG

Recall that a CCG (Σ, A, R, I, L) is called *pure* if $R = \mathcal{R}(A, k)$ for some $k \in \mathbb{N}$.[4] In this section, we show that there exist CFG derivation tree languages that cannot be generated by any pure CCG. In particular, this shows that the inclusion demonstrated in the previous section is proper for pure CCG. We start with our counterexample CFG. We will use nonterminals that are pairs of integers, and we will use standard arithmetic. In fact, to make the text more readable, we assume henceforth that all computations with the integers inside of nonterminals are performed modulo 3.

4: As in the complete chapter, we ignore substitution rules here.

Example 5.2.1 Let us consider the CFG $\mathcal{G}_{\text{ex}} = \big(N, \Gamma, \{\langle 0, 0\rangle\}, P\big)$ with the nonterminals $N = \big\{\langle i, j\rangle \mid i, j \in \mathbb{Z}_3\big\}$, the terminals $\Gamma = \{\alpha\}$, and the set P of productions containing exactly $\langle i, j\rangle \to \langle i{+}1, j\rangle\,\langle i, j{+}1\rangle$ and $\langle i, j\rangle \to \alpha$ for every $\langle i, j\rangle \in N$. Clearly, the tree language $\mathcal{D}(\mathcal{G}_{\text{ex}})$ is not universally mht-bounded.

Theorem 4.1.9 already shows that the tree language $\mathcal{D}(\mathcal{G}_{\text{ex}})$ is not generatable by any 0-CCG. Similarly, it is impossible to generate $\mathcal{D}(\mathcal{G}_{\text{ex}})$ with a pure CCG. This follows from the transformation schemes of [50] that change the order of consecutive application and non-application operations, resulting in derivation trees with reordered subtrees and therefore with the wrong shape after relabeling. They are depicted in Figure 4.7 of Section 4.2.1, where we already have encountered them. Due to the absence of rule restrictions in pure CCG, the applicability of these transformations cannot be prevented.

Theorem 5.2.2 *The tree language $\mathcal{D}(\mathcal{G}_{\text{ex}})$ is not generatable by any pure CCG.*

Proof. For the sake of a contradiction, suppose that there exists a pure CCG $\mathcal{G} = \big(\Sigma, A, \mathcal{R}(A, k), I, L\big)$ and a (category) relabeling ρ such that $\mathcal{T}_\rho(\mathcal{G}) = \mathcal{D}(\mathcal{G}_{\text{ex}})$. Let $t \in \mathcal{D}(\mathcal{G}_{\text{ex}})$ be such that $\text{mht}(t) > \text{arity}(L)$. Since the derivation tree language $\mathcal{D}(\mathcal{G}_{\text{ex}})$ is not universally mht-bounded, such a tree exists. By Lemma 4.1.3, the CCG $\mathcal{G}$ has to use non-application operations to produce trees with the shape of t, so $k \geq 1$. Let $u \in \mathcal{D}(\mathcal{G})$ be such that $t \in \rho(u)$. Moreover, select the first (i.e., least with respect to the shortlex order[5]) position $w \in \text{pos}(u)$ at which an application rule is applied

5: In the *shortlex order*, strings are first ordered by their length, and strings of the same length are ordered lexicographically [81, page 14].

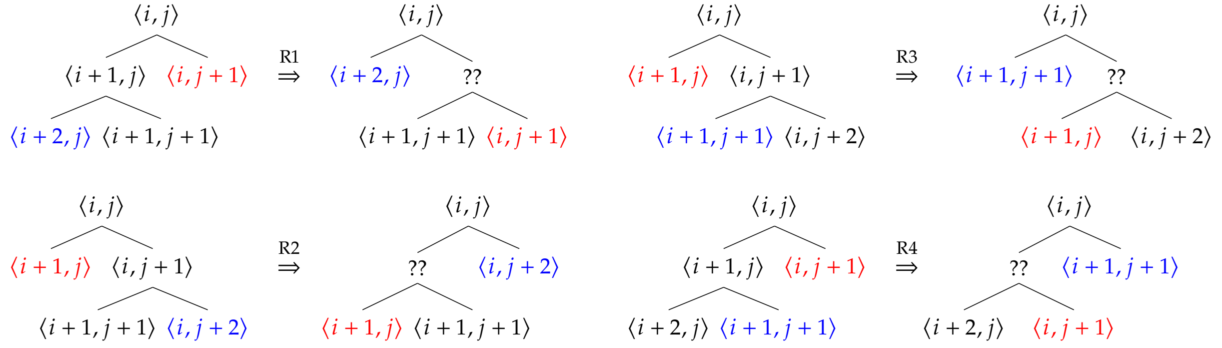


Figure 5.3: Rule schemes of [50] with the relabeling indicated based on the input tree. The roots of equal subtrees occurring at the wrong position are marked in red and blue. See Figure 4.7 for the rule schemes before relabeling.

followed by a non-application operation. Such a position must exist since the rule applied at the root is always an application rule (because every initial category is atomic). This yields one of the cases listed left of the ⇒-symbol in Figure 4.7. We apply the such identified transformation rule of Figure 4.7 that matches at w to obtain another tree $u' \in \mathcal{D}(\mathcal{G})$ [50]. However, as we illustrate in Figure 5.3, each transformation rule leads to a derivation tree of the wrong shape (i.e., one that can be relabeled in an undesirable manner).

Let us walk through one case in detail. Suppose that the first transformation rule of Figure 4.7 applies at position w. Moreover, assume that $u(w)$ relabels to $\langle i, j \rangle$ for some $i, j \in \mathbb{Z}_3$ because $t \in \rho(u) \subseteq \mathcal{D}(\mathcal{G}_{\mathrm{ex}})$. Then we know that $u(w2)$ relabels to $\langle i, j + 1 \rangle$ because $\rho(u) \subseteq \mathcal{D}(\mathcal{G}_{\mathrm{ex}})$. However, after applying the transformation (i.e., in the tree u'), the subtree $u|_{w2}$ now occurs at $u'|_{w22}$, which creates the wrong shape. We can conclude that $\rho(u') \nsubseteq \mathcal{D}(\mathcal{G}_{\mathrm{ex}})$, contradicting $\mathcal{T}_\rho(\mathcal{G}) = \mathcal{D}(\mathcal{G}_{\mathrm{ex}})$. □

5.3 Spine Grammar

In the following four sections, we will prove the inverse direction of Theorem 5.1.13. More precisely, we will show that each tree language generated by an sCFTG can also be generated by a CCG. We already outlined the general approach of the construction at the beginning of this chapter. Figure 5.4 depicts a schematic overview of the involved steps. The construction starts from a spine grammar, which is the formalism that we will introduce in the following. We also cover its normal form and how it is established, and lay the foundations for the decomposition of generated trees by introducing spinal trees.

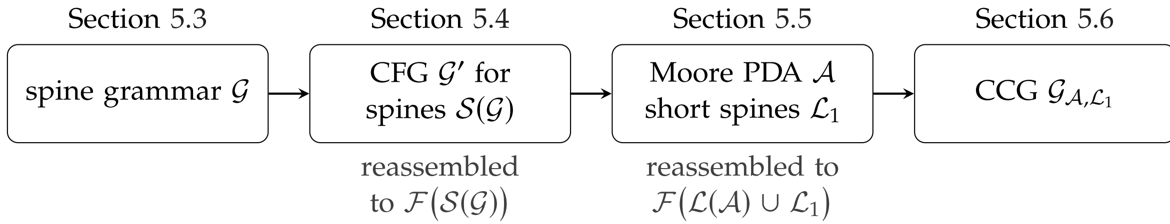


Figure 5.4: Overview of the conversion from spine grammar to CCG with the most relevant notations.

Spine grammars [21] were originally defined as a restriction of CFTG and possess the same expressive power as sCFTG, which follows from the normal form for spine grammars. Although sCFTG are more established, we elect to utilize spine grammars because of their essential notion of spines and use a variant of their normal form. Deviating from the original definition [21, Definition 3.2], we treat spine grammars as a restriction on sCFTG and equip the terminal symbols with a "spine direction" (instead of the nonterminals, which is not useful in sCFTG).[6] By creating copies of binary terminal symbols it can be shown that both variants are equivalent modulo relabeling. More specifically, under our definition, each spine grammar is clearly itself an sCFTG and for each sCFTG $\mathcal{G}$ there exist a spine grammar $\mathcal{G}'$ and a deterministic relabeling ρ such that $\mathcal{T}(\mathcal{G}) = \{\rho(t) \mid t \in \mathcal{T}(\mathcal{G}')\}$.

6: In the original definition, productions are not necessarily linear or nondeleting, and the nonterminals may have rank greater than 1. Nonterminals are equipped with a *head* that specifies the direction where the spine continues. The spine of the right-hand side of a production is the path from the root to the unique appearance of the variable that is in head direction of the (non-nullary) nonterminal on the left-hand side. All other variables on the left-hand side of productions have to appear as children of spinal nodes on the right-hand side if they appear at all.

Consider some sCFTG production $(n \to C) \in P$ with $n \in N_1$. The *spine* of C is simply the path from the root of C to the unique occurrence $\mathrm{pos}_\square(C)$ of $\square$. The special feature of a spine grammar is that the symbols along the spine indicate exactly in which direction the spine continues. Since only the binary terminal symbols offer branching, the special feature of spine grammars is the existence of a map $d\colon \Sigma_2 \to [2]$ that tells us for each binary terminal symbol $\sigma \in \Sigma_2$ whether the spine continues to the left, in which case $d(\sigma) = 1$, or to the right, in which case $d(\sigma) = 2$. This map d, called *spine direction*, applies to all instances of the binary terminal symbol σ in all productions with spines.

Definition 5.3.1 *Let $\mathcal{G} = (N, \Sigma, S, P)$ be an sCFTG. It is called* spine grammar *if there exists a map $d\colon \Sigma_2 \to [2]$ such that for every production $(n \to C) \in P$ with $n \in N_1$ and every position $w \in \mathrm{Pref}(\mathrm{pos}_\square(C))$ with $C(w) \in \Sigma_2$ also $wi \in \mathrm{Pref}(\mathrm{pos}_\square(C))$ with $i = d(C(w))$.*

We will use the term spine also to refer to the paths that follow the spine direction in a tree generated by a spine grammar. These paths do not necessarily have to start at the root. In this manner, each such tree can be decomposed into a set of spines. Henceforth, let $\mathcal{G} = (N, \Sigma, S, P)$ be a spine grammar with associated map $d\colon \Sigma_2 \to [2]$.

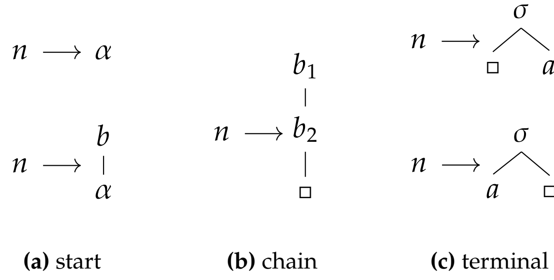


Figure 5.5: Types of productions of spine grammars in normal form (see Definition 5.3.2).

Normal Form

In spine grammars in normal form each production has one of the three production types that are illustrated in Figure 5.5. Additionally, there is a single start nonterminal s that is isolated and cannot occur on the right-hand sides.

Definition 5.3.2 *A spine grammar $\mathcal{G}$ is in* normal form *if $S = \{s\}$ and each $(n \longrightarrow r) \in P$ is of one of the forms*

(i) start: $r = b(\alpha)$ *or* $r = \alpha$ *for some* $b \in N_1$ *and* $\alpha \in \Sigma_0$*,*
(ii) chain: $r = b_1\big(b_2(\square)\big)$ *for some* $b_1, b_2 \in N_1$*, or*
(iii) terminal: $r = \sigma(\square, a)$ *or* $r = \sigma(a, \square)$ *for some* $\sigma \in \Sigma_2$ *and* $a \in N_0 \setminus S$*.*

Spinal Trees

Using a single start production followed by a number of chain and terminal productions, a nullary nonterminal n can be rewritten to a tree t that consists of a spine of terminals, where each non-spinal child is a nullary nonterminal. Formally, for every nullary nonterminal $n \in N_0$ let

$$I_{\mathcal{G}}(n) = \Big\{t \in T_{\Sigma_2,\emptyset}(\Sigma_0 \cup N_0) \;\Big|\; n \;(\Rightarrow_{\mathcal{G}} ; \Rightarrow^*_{\mathcal{G}'})\; t\Big\}$$

where $\mathcal{G}'$ is the spine grammar $\mathcal{G}$ without start productions; i.e., $\mathcal{G}' = (N, \Sigma, S, P')$ with productions $P' = \big\{(n \to r) \in P \mid n \in N_1\big\}$. So we perform a single derivation step using the productions of $\mathcal{G}$ followed by any number of derivation steps using only productions of $\mathcal{G}'$. The elements of $I_{\mathcal{G}}(n)$ are called *spinal trees* for n and their *spine generator* is n. By a suitable renaming of nonterminals we can always achieve that the spine generator does not occur in any of its spinal trees. Accordingly, the spine grammar $\mathcal{G}$ is *normalized* if it is in normal form and $I_{\mathcal{G}}(n) \subseteq T_{\Sigma_2,\emptyset}\big(\Sigma_0 \cup (N_0 \setminus \{n\})\big)$ for every nullary nonterminal $n \in N_0$.

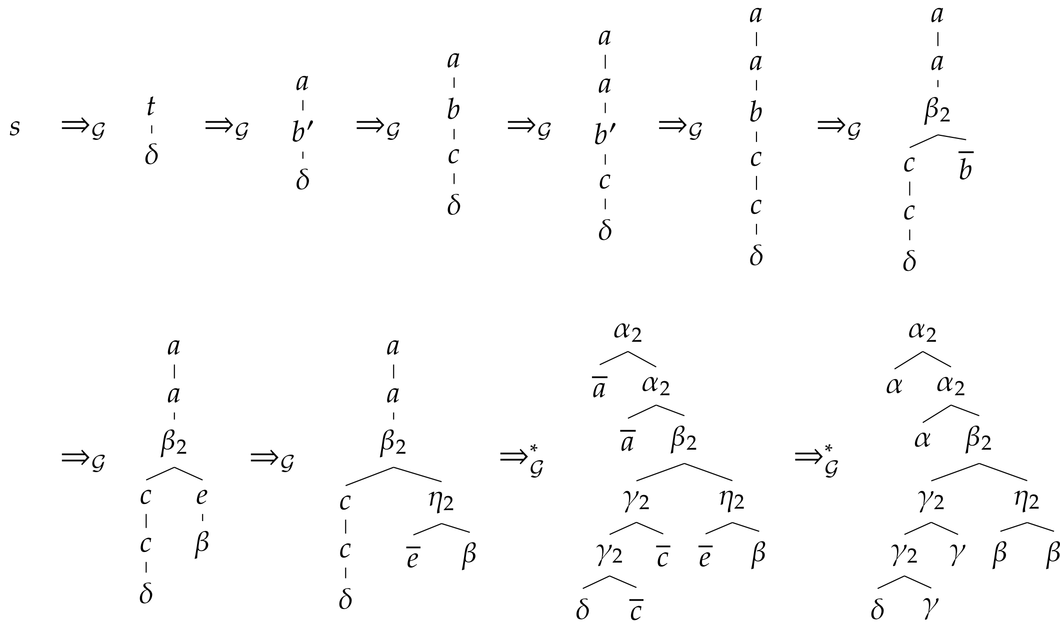


Figure 5.6: Derivation of the normalized spine grammar $\mathcal{G}$ of Example 5.3.3.

Example 5.3.3 The spine grammar $\mathcal{G} = (N, \Sigma, \{s\}, P)$ with $N_1 = \{t, a, b, c, b', e\}$, $N_0 = \{s, \bar{a}, \bar{b}, \bar{c}, \bar{e}\}$, $\Sigma_2 = \{\alpha_2, \beta_2, \gamma_2, \eta_2\}$, $\Sigma_0 = \{\alpha, \beta, \gamma, \delta\}$, and P as shown below is clearly normalized.

$$\begin{array}{llll} & \bar{a} \to \alpha & t \to a\big(b'(\square)\big) & a \to \alpha_2(\bar{a}, \square) \\ s \to t(\delta) & \bar{b} \to \beta & b' \to b\big(c(\square)\big) & b \to \beta_2(\square, \bar{b}) \\ \bar{b} \to e(\beta) & \bar{c} \to \gamma & b \to a\big(b'(\square)\big) & c \to \gamma_2(\square, \bar{c}) \\ & \bar{e} \to \beta & e \to e\big(e(\square)\big) & e \to \eta_2(\bar{e}, \square) \end{array}$$

Figure 5.6 shows a derivation using $\mathcal{G}$. The generated tree is also depicted in Figure 5.7a with the spines marked by thick edges. The spinal tree corresponding to the main spine (i.e., the spine containing the root) of the tree is shown in Figure 5.7b. The yield of $\mathcal{T}(\mathcal{G})$ is $\{\alpha^n\, \delta\, \gamma^n\, \beta^m \mid n, m \in \mathbb{N}_+\}$. Note that $\mathcal{T}(\mathcal{G})$ coincides with the tree language generated by the TAG $\mathcal{G}_3$ of Example 2.3.4.

Normalization

In the following, we will show that we can always transform a spine grammar into a strongly equivalent normalized spine grammar. This result is a variant of Theorem 1 of Fujiyoshi and Kasai [21].

Theorem 5.3.4 *For every spine grammar there is a strongly equivalent normalized spine grammar.*

Proof. The normal form of Fujiyoshi and Kasai [21, Definition 4.2] is different from that of Definition 5.3.2 in the following three aspects.

- The first case of productions of type (i) has right-hand side $r = b(a)$ with $b \in N_1$, $a \in N_0$.
- Productions of type (ii) have a right-hand side of the form $r = b_1(\cdots(b_m(\square))\cdots)$ with $m \in \mathbb{N}$ and $b_i \in N_1$ for $i \in [m]$.
- In productions of type (iii) the start nonterminal is not excluded from the set of nonterminals that can be produced.

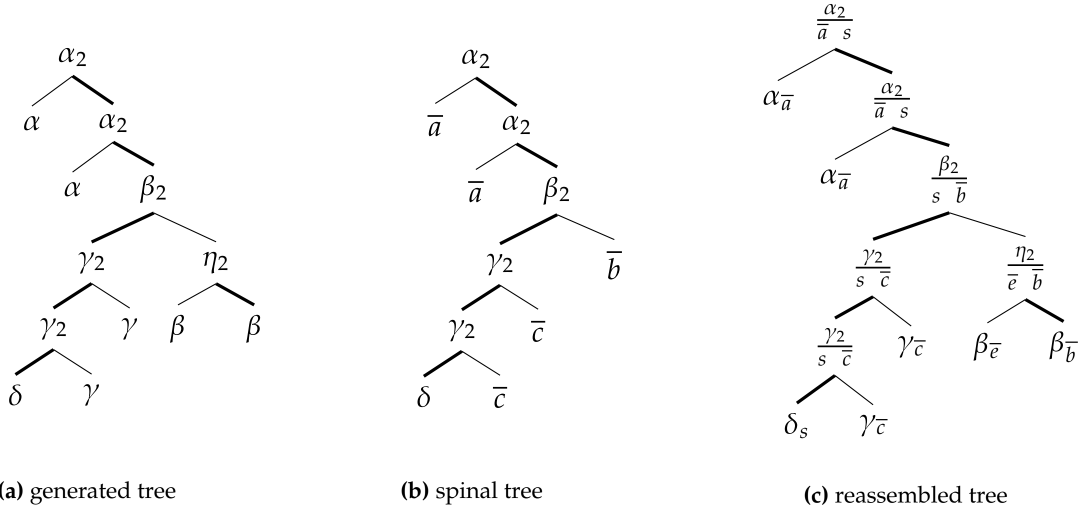
When starting from their normal form, standard techniques can be used to modify the grammar such that all productions $n \to r$ with $r = b_1(\cdots(b_m(\square))\ \cdots)$ have $m = 2$, that the start nonterminal is isolated, and that no spinal tree contains its own spine generator. We therefore assume that these conditions are already met.

Let $\mathcal{G} = (N, \Sigma, S, P)$ be a spine grammar that is already in the desired form except for $P_1 = \{n \to b(a) \mid n, a \in N_0,\ b \in N_1\} \subseteq P$. We define the spine grammar $\mathcal{G}' = (N_0 \cup N_1', \Sigma, S, P')$ with unary nonterminals $N_1' = N_1 \cup N_0 \times \Sigma_0$ and productions

$$
\begin{aligned}
P' = \big(P \setminus P_1\big) \cup \\
&\big\{n \to \langle n, \alpha\rangle(\alpha) \bigm| n \to b(a) \in P_1, \alpha \in \Sigma_0\big\} \cup \\
&\big\{\langle n, \alpha\rangle \to b\big(\langle a, \alpha\rangle(\square)\big) \bigm| n \to b(a) \in P_1, \alpha \in \Sigma_0\big\} \cup \\
&\big\{\langle a, \alpha\rangle \to \square \bigm| a \to \alpha \in P\big\} \ .
\end{aligned}
$$

When a nonterminal is expanded to a non-trivial spine, the terminal symbol at the bottom of that spine is guessed. That symbol is immediately produced and stored in its parent nonterminal. If the original nonterminal corresponding to the parent can be replaced by the guessed terminal symbol in the original grammar, the parent can be removed instead since the terminal symbol was already produced (see also [61, Section 5] for a similar construction). It is easy to verify that $\mathcal{G}'$ still generates the same tree language as $\mathcal{G}$.

The set P' contains collapsing productions $a \to \square$ with $a \in N_1$ that are not allowed in our normal form. They are subsequently removed using the standard techniques for removal of ε-productions and unit productions from CFG [27, Section 4.3] to obtain a spine grammar in the desired normal form. □



(a) generated tree **(b)** spinal tree **(c)** reassembled tree

Figure 5.7: Tree generated by spine grammar $\mathcal{G}$, a spinal tree in $I_{\mathcal{G}}(s)$ (see Example 5.3.3), and a tree in $\mathcal{F}(\mathcal{S}(\mathcal{G}))_s$ reassembled from spines (see Example 5.4.6).

5.4 Decomposition into Spines

In this section, we proceed with the construction starting from the normalized spine grammar $\mathcal{G}$. We construct a CFG $\mathcal{G}'$ whose generated string language—the set of spines $\mathcal{S}(\mathcal{G})$—is a representation of spinal trees of $\mathcal{G}$. Then, we define the operation $\mathcal{F}$ that reassembles these spines such that the resulting tree language is strongly equivalent to $\mathcal{T}(\mathcal{G})$ (up to relabeling). The reassembled spines as a representation of the original tree language will prove useful as an intermediate step later when showing the correctness of our CCG construction in Section 5.6.

Context-Free Grammar of Spines

First, we construct a CFG that captures all information of $\mathcal{G}$. It represents the spinal trees (from bottom to top) as strings and enriches the symbols with the spine generator (initialized by start productions and preserved by chain productions) and the non-spinal child (given by terminal productions). The order of these annotations depends on the spine direction of the symbol. The leftmost symbol of the generated strings has only a spine generator annotated since the bottom of the spine has no children. To simplify the notation, we write n_g for $(n, g) \in N^2$, α_n for $(\alpha, n) \in \Sigma_0 \times N$, and $\frac{\sigma}{n_1\ n_2}$ for $(\sigma, n_1, n_2) \in \Sigma_2 \times N^2$.

Definition 5.4.1 *Let spine grammar $\mathcal{G}$ be normalized and $\top \notin N$. The spines $\mathcal{S}(\mathcal{G}) = \mathcal{L}(\mathcal{G}')$ of $\mathcal{G}$ are the strings generated by the CFG $\mathcal{G}' = (\{\top\} \cup N^2, \Sigma', \{\top\}, P')$ with $\Sigma' = (\Sigma_0 \times N) \cup (\Sigma_2 \times N^2)$ and productions $P' = P_0 \cup P_1 \cup P_2$ given by*

$$P_0 = \left\{\top \to \alpha_n \;\middle|\; (n \to \alpha) \in P\right\} \cup \left\{\top \to \alpha_n\, b_n \;\middle|\; \big(n \to b(\alpha)\big) \in P\right\}$$

$$P_1 = \bigcup_{g \in N} \left\{n_g \to b'_g\, b_g \;\middle|\; \big(n \to b\big(b'(\square)\big)\big) \in P\right\}$$

$$P_2 = \bigcup_{g \in N} \left(\left\{n_g \to \frac{\sigma}{g\;\; n'} \;\middle|\; \big(n \to \sigma(\square, n')\big) \in P\right\} \cup \left\{n_g \to \frac{\sigma}{n'\;\; g} \;\middle|\; \big(n \to \sigma(n', \square)\big) \in P\right\}\right).$$

For each start production we obtain a single production since the nonterminal on the left-hand side becomes the spine generator. On the other hand, for each chain or terminal production we have to combine them with all nonterminals, as we do not know the spine generator of the nonterminal on the left-hand side of the original production. When a string is derived, the spine generators are pulled through originating from start productions and are consistent throughout the string. The construction is illustrated by the following example.

Example 5.4.2 We list some corresponding productions of the spine grammar $\mathcal{G}$ (left) of Example 5.3.3 and the CFG $\mathcal{G}'$ (right) for its spines $\mathcal{S}(\mathcal{G})$.

$$\begin{array}{lcllll}
\overline{a} \to \alpha & : & \top \to \alpha_{\overline{a}} & & & \\
s \to t(\delta) & : & \top \to \delta_s t_s & & & \\
t \to a\big(b'(\square)\big) & : & t_s \to b'_s a_s & t_a \to b'_a\, a_a & t_{\overline{b}} \to b'_{\overline{b}}\, a_{\overline{b}} & \dots \\
a \to \alpha_2(\overline{a}, \square) & : & a_s \to \dfrac{\alpha_2}{\overline{a}\;\; s} & a_a \to \dfrac{\alpha_2}{\overline{a}\;\; a} & a_{\overline{b}} \to \dfrac{\alpha_2}{\overline{a}\;\; b} & \dots
\end{array}$$

The language generated by $\mathcal{G}'$ is

$$\mathcal{S}(\mathcal{G}) = \mathcal{L}(\mathcal{G}') = \left\{\delta_s \frac{\gamma_2}{s\;\;\overline{c}}^{\,n} \frac{\beta_2}{s\;\;\overline{b}} \frac{\alpha_2}{\overline{a}\;\; s}^{\,n} \;\middle|\; n \in \mathbb{N}_+\right\} \cup \left\{\beta_{\overline{b}} \frac{\eta_2}{\overline{e}\;\;\overline{b}}^{\,m} \;\middle|\; m \in \mathbb{N}\right\} \cup \left\{\alpha_{\overline{a}}, \beta_{\overline{e}}, \gamma_{\overline{c}}\right\}.$$

Note that each string that is generated by the CFG belongs to $(\Sigma_0 \times N)(\Sigma_2 \times N^2)^*$. The spine generator of a symbol in such a string can be accessed as follows.

Definition 5.4.3 *The generator* $\mathrm{gen}\colon (\Sigma_0 \times N) \cup (\Sigma_2 \times N^2) \to N$ *is the nonterminal in spine direction and is given by*

$$\mathrm{gen}(a) = \begin{cases} n & \text{if } a = \alpha_n \in \Sigma_0 \times N \\ n_{d(\sigma)} & \text{if } a = \frac{\sigma}{n_1\ n_2} \in \Sigma_2 \times N^2 \end{cases} .$$

Reassembling Spines

Next, we define how to reassemble those spines to form trees again, which then relabel to the original trees generated by $\mathcal{G}$. The operation given in the following definition describes how a string generated by the CFG can be transformed into a tree by attaching subtrees in the non-spinal direction of each symbol, whereby the non-spinal child annotation of the symbol and the spinal annotation of the root of the attached tree have to match.

Definition 5.4.4 *We are given a set of trees* $\mathcal{T} \subseteq T_{\Sigma_2 \times N^2, \emptyset}(\Sigma_0 \times N)$ *and a string* $w \in W = (\Sigma_0 \times N)(\Sigma_2 \times N^2)^*$*. For* $n \in N$*, let* $\mathcal{T}_n = \{t \in \mathcal{T} \mid \mathrm{gen}(t(\varepsilon)) = n\}$ *be those trees of* $\mathcal{T}$ *whose root label stores spine generator* n*. We recursively define the tree language* $\mathrm{attach}_{\mathcal{T}}(w) \subseteq T_{\Sigma_2 \times N^2, \emptyset}(\Sigma_0 \times N)$ *by* $\mathrm{attach}_{\mathcal{T}}(\alpha_n) = \{\alpha_n\}$ *for all* $\alpha_n \in \Sigma_0 \times N$*, and*

$$\mathrm{attach}_{\mathcal{T}}\Big(w \frac{\sigma}{n_1\ n_2}\Big) = \left\{ \frac{\sigma}{n_1\ n_2}(t_1, t_2) \;\middle|\; \begin{matrix} t_{d(\sigma)} & \in & \mathrm{attach}_{\mathcal{T}}(w) \\ t_{3-d(\sigma)} & \in & \mathcal{T}_{n_{3-d(\sigma)}} \end{matrix} \right\}$$

for all $w \in W$ *and* $\frac{\sigma}{n_1\ n_2} \in \Sigma_2 \times N^2$.

To obtain the tree language defined by $\mathcal{G}$, it is necessary to apply this operation recursively on the set of spines.

Definition 5.4.5 *Let* $\mathcal{L} \subseteq (\Sigma_0 \times N)(\Sigma_2 \times N^2)^*$*. We inductively define the tree language* $\mathcal{F}(\mathcal{L})$ *generated by* $\mathcal{L}$ *as the smallest tree language* $\mathcal{F}$ *such that* $\mathrm{attach}_{\mathcal{F}}(w) \subseteq \mathcal{F}$ *for every* $w \in \mathcal{L}$*.*

Example 5.4.6 The CFG $\mathcal{G}'$ of Example 5.4.2 generates the set of spines $\mathcal{S}(\mathcal{G})$ and $\mathcal{F}(\mathcal{S}(\mathcal{G}))_s$ contains the correctly assembled trees formed from these spines. Figure 5.7c shows a tree of $\mathcal{F}(\mathcal{S}(\mathcal{G}))_s$ since the generator of the main spine is s, which is stored in spinal direction in the root label $\frac{\alpha_2}{a\ s}$. We can observe the correspondence of annotations in non-spinal direction and the spine generator of the respective child in the same direction.

Next we prove that $\mathcal{F}\big(\mathcal{S}(\mathcal{G})\big)_s$ and $\mathcal{T}(\mathcal{G})$ coincide modulo relabeling. This shows that the context-free language $\mathcal{S}(\mathcal{G})$ of spines completely describes the tree language $\mathcal{T}(\mathcal{G})$ generated by $\mathcal{G}$. The relabeling π simply removes the annotations that were added to the symbols when $\mathcal{S}(\mathcal{G})$ was constructed through $\mathcal{G}'$. The difference can be observed in the trees of Figures 5.7a and 5.7c.

Theorem 5.4.7 *Let spine grammar $\mathcal{G}$ be normalized, and let the relabeling $\pi\colon (\Sigma_0 \times N) \cup (\Sigma_2 \times N^2) \to \Sigma_0 \cup \Sigma_2$ be given by $\pi(\alpha_n) = \alpha$ and $\pi\big(\frac{\sigma}{n_1\ n_2}\big) = \sigma$ for all $\alpha \in \Sigma_0$, $\sigma \in \Sigma_2$, and $n, n_1, n_2 \in N$. Then $\pi\big(\mathcal{F}(\mathcal{S}(\mathcal{G}))_s\big) = \mathcal{T}(\mathcal{G})$.*

Proof. We will prove a more general statement. Let $\mathcal{G}'$ be the CFG constructed for $\mathcal{G}$ in Definition 5.4.1. Given $n \in N_0$, we will show that the tree language $\pi\big(\mathcal{F}(\mathcal{S}(\mathcal{G}))_n\big)$ coincides with the set of trees $\big\{t \in T_{\Sigma_2,\emptyset}(\Sigma_0) \mid n \Rightarrow_{\mathcal{G}}^* t\big\}$, which can be derived in $\mathcal{G}$ starting from nullary nonterminal n. To this end, we show inclusion in both directions.

The inclusion ($\subseteq$) is proved by induction on the size of $t \in \mathcal{F}\big(\mathcal{S}(\mathcal{G})\big)_n$. Clearly, the construction of t was carried out on the basis of a string $w = \alpha_n \frac{\sigma_1}{n_{1,1}\ n_{1,2}} \cdots \frac{\sigma_m}{n_{m,1}\ n_{m,2}} \in \mathcal{S}(\mathcal{G})$ with spine generator n and $n_{i,d(\sigma_i)} = n$ for all $i \in [m]$. Hence, there is a derivation $\top \Rightarrow_{\mathcal{G}'}^* w$, where $\top$ is the start nonterminal of $\mathcal{G}'$. Each production applied during this derivation uniquely corresponds to a production of the spine grammar $\mathcal{G}$. This yields a derivation $n \Rightarrow_{\mathcal{G}}^* t_w$ of a spinal tree $t_w \in I_{\mathcal{G}}(n)$, where the spine of t_w is labeled (from bottom to top) by $\pi(w)$. If t consists of a single node, we are finished. Else, besides the spine, t_w contains only leaf nodes that for $i \in [m]$ are attached below σ_i in the non-spinal direction $3 - d(\sigma_i)$ and are labeled by nullary nonterminals $n_{i,3-d(\sigma_i)}$, respectively. For better readability, let $n_i = n_{i,3-d(\sigma_i)}$ in the following. Each nonterminal annotation n_i in w implies the attachment of a tree $t_i \in \mathcal{F}\big(\mathcal{S}(\mathcal{G})\big)_{n_i}$. These attached trees are smaller than t, so we can use the induction hypothesis and conclude that there is a derivation $n_i \Rightarrow_{\mathcal{G}}^* \pi(t_i)$. Combining those derivations, we obtain a derivation $n \Rightarrow_{\mathcal{G}}^* \pi(t)$.

For the other direction ($\supseteq$), we use induction on the length of the derivation $n \Rightarrow_{\mathcal{G}}^* t$ to show that there exists a tree $t' \in \mathcal{F}\big(\mathcal{S}(\mathcal{G})\big)_n$ with $\pi(t') = t$. To this end, we reorder the derivation such that a spinal tree $u \in I_{\mathcal{G}}(n)$ is derived first (i.e., $n \Rightarrow_{\mathcal{G}}^* u \Rightarrow_{\mathcal{G}}^* t$). Suppose that this spinal tree u has the nullary terminal symbol α at the bottom and contains m binary terminal symbols $\sigma_1, \ldots, \sigma_m$ (from bottom to top). Let n_i be the non-spinal child of σ_i. It is attached in direction $3 - d(\sigma_i)$. Due to the construction of $\mathcal{G}'$, there is a corresponding derivation $\top \Rightarrow_{\mathcal{G}'}^* w$, for which the derived string $w \in \mathcal{S}(\mathcal{G})$ has the form $w = \alpha_n \frac{\sigma_1}{n_{1,1}\ n_{1,2}} \cdots \frac{\sigma_m}{n_{m,1}\ n_{m,2}}$ with

$n_{i,d(\sigma_i)} = n$ and $n_{i,3-d(\sigma_i)} = n_i$ for all $i \in [m]$. If $m = 0$, we are finished. Else, the remaining nonterminals in u are replaced by subderivations $n_i \Rightarrow^*_{\mathcal{G}} t_i$ for all $i \in [m]$. These subderivations are shorter than the overall derivation $n \Rightarrow^*_{\mathcal{G}} t$, so by the induction hypothesis, there exist trees $t'_i \in \mathcal{F}\big(\mathcal{S}(\mathcal{G})\big)_{n_i}$ such that $\pi(t'_i) = t_i$ for all $i \in [m]$. Attaching those trees t'_i to w, we obtain the tree $t' \in \mathcal{F}\big(\mathcal{S}(\mathcal{G})\big)_n$. As required, we have $\pi(t') = t$.

We thus have proved that the set $\pi\big(\mathcal{F}(\mathcal{S}(\mathcal{G}))_s\big)$ of reassembled and relabeled spines coincides with $\big\{t \in T_{\Sigma_2,\emptyset}(\Sigma_0) \mid s \Rightarrow^*_{\mathcal{G}} t\big\}$. Hence, $\pi\big(\mathcal{F}(\mathcal{S}(\mathcal{G}))_s\big) = \mathcal{T}(\mathcal{G})$. □

5.5 Moore Push-Down Automata

In this section, we will introduce a Moore variant of push-down automata [3] that is geared towards our needs and still accepts the context-free languages (of strings of length ≥ 2). It will be similar to the push-down Moore machines of Decker, Leucker, and Thoma [14]. After defining the automaton model and characterizing its generative power, we will discuss two properties that can always be established, namely a coherence between popped symbol and entered state, and a form of lookahead. Finally, we use the thus prepared automaton model to represent the spines defined in the previous section and show that, when used in conjunction with a set $\mathcal{L}_1$ for spines of length 1, its generated language can be reassembled in the same manner as before to yield a tree language that can be relabeled to the original tree language $\mathcal{T}(\mathcal{G})$.

Instead of processing input symbols as part of transitions (as in Mealy machines), Moore machines output a unique input symbol in each state [18]. Recall that for every alphabet Γ we have $\Gamma^{\leq 1} = \{\varepsilon\} \cup \Gamma$ and additionally let $\Gamma^{\geq 2} = \big\{w \in \Gamma^* \mid 2 \leq |w|\big\}$ be the strings of length at least 2.

Definition 5.5.1 *A* Moore push-down automaton *(MPDA) is defined as a tuple* $\mathcal{A} = (Q, \Sigma, \Gamma, \delta, \tau, I, F)$ *that consists of*

(i) finite sets Q, Σ, *and* Γ *of* states, input symbols, *and* stack symbols, *respectively,*
(ii) a set $\delta \subseteq \big(Q \times \Gamma^{\leq 1} \times \Gamma^{\leq 1} \times Q\big) \setminus \big(Q \times \Gamma \times \Gamma \times Q\big)$ *of* transitions,
(iii) an output function $\tau \colon Q \to \Sigma$, *and*
(iv) sets $I, F \subseteq Q$ *of* initial *and* final states, *respectively.*

Due to the definition of δ, as for the PDA of Definition 2.2.3, in a single step we can either push or pop a single stack symbol or ignore the stack. In the following, let $\mathcal{A} = (Q, \Sigma, \Gamma, \delta, \tau, I, F)$ be an

MPDA. On the set $\text{Conf}_{\mathcal{A}} = Q \times \Gamma^*$ of configurations of $\mathcal{A}$ the *move relation* $\vdash_{\mathcal{A}} \subseteq \text{Conf}_{\mathcal{A}}^2$ is

$$\vdash_{\mathcal{A}} = \bigcup_{\substack{(q,\gamma,\gamma',q')\in\delta \\ \alpha\in\Gamma^*}} \left\{(\langle q, \gamma\alpha\rangle, \langle q', \gamma'\alpha\rangle) \in \text{Conf}_{\mathcal{A}}^2 \;\middle|\; \gamma\alpha \neq \varepsilon\right\}$$

and a configuration $\langle q, \alpha\rangle \in \text{Conf}_{\mathcal{A}}$ is *initial* [respectively, *final*] if $q \in I$ and $\alpha \in \Gamma$ [respectively, $q \in F$ and $\alpha = \varepsilon$]. An *accepting run* is defined in the same manner as for PDAs. However, note that contrary to PDAs we can start with an arbitrary symbol on the stack. The language $\mathcal{L}(\mathcal{A})$ *accepted* by $\mathcal{A}$ contains exactly those strings $w \in \Sigma^*$ for which there exists an accepting run $\langle q_0, \alpha_0\rangle, \dots, \langle q_n, \alpha_n\rangle$ such that $w = \tau(q_0) \cdots \tau(q_n)$. Thus, we accept the strings that are output symbol-by-symbol by the states attained during an accepting run. As usual, two MPDAs are *equivalent* if they accept the same language. Because no initial configuration is final, each accepting run has length at least 2, so we can only accept strings of length at least 2. While we could adjust the model to remove this restriction, the presented version serves our later purposes best.

Theorem 5.5.2 *MPDA accept the context-free languages of strings of length at least* 2.

Proof. The straightforward part of the proof is to show that each language accepted by an MPDA is context-free. Given some MPDA, an equivalent PDA can easily be constructed by moving the output from a state to its outgoing transitions. As the MPDA starts with an arbitrary symbol on the stack, and the classical PDA starts with the bottom symbol $\perp$, we add two additional states. A new unique initial state has an outgoing ε-transition to each original initial state for each symbol $\gamma \in \Gamma$, pushing that symbol to the stack. A new unique final state is reachable by each original final state via a transition that pops $\perp$ and realizes the output of that original final state. Note that the inclusion of ε-entries does not increase the generative power of PDAs [3, page 143].[7] Hence, the language accepted by the constructed PDA is context-free.

7: That PDA with ε-transitions have the same expressivity as PDA without them (i.e., *real-time* PDA) is a consequence of the Greibach normal form for CFG (see page 36).

For the converse, let $\mathcal{L} \subseteq \Sigma^{\geq 2}$ be a context-free language and let $\mathcal{A} = (Q, \Sigma, \Gamma, \delta, I, \perp, F)$ be a PDA such that $\mathcal{L}(\mathcal{A}) = \mathcal{L}$. We assume without loss of generality that the initial states of $\mathcal{A}$ have no incoming transitions and that the final states of $\mathcal{A}$ have no outgoing transitions and all their incoming transitions pop $\perp$. We construct an MPDA $\mathcal{A}'$ with $\mathcal{L}(\mathcal{A}') = \mathcal{L}$ in the spirit of the classical conversion from Mealy to Moore machines [18]. The main idea is to shift the input symbol a from the transition $(q, a, \gamma, \gamma', q') \in \delta$ to the target state q'. Additionally, since there is always one more

configuration compared to the number of moves (and thus involved transitions) in an accepting run, the first move needs to be avoided in $\mathcal{A}'$. If the corresponding transition pushes a symbol to the stack, we have to store it in the target state of the transition. This state becomes an initial state of $\mathcal{A}'$. To be able to discern if the stored symbol can be deleted, $\mathcal{A}'$ needs to be aware whether the stack currently contains only one symbol, since in $\mathcal{A}$ the symbol pushed in the first transition lies above the bottom symbol. Since we clearly cannot store the size of the current stack in the state, we need to mark the symbol at the bottom of the stack.

Recall that $\Gamma_\perp = \Gamma \cup \{\perp\}$. Formally, we construct the MPDA $\mathcal{A}' = (Q', \Sigma, \Gamma', \delta', \pi_2, I', F')$ with

- $Q' = Q \times \Sigma \times \Gamma^{\leq 1} \times \{0,1\}$,
- $I' = \big\{\langle q', a, \gamma', 1\rangle \in Q' \mid \exists q \in I\colon (q, a, \gamma, \gamma', q') \in \delta\big\}$,
- $\Gamma' = \Gamma_\perp \times \{0,1\}$,
- $F' = F \times \Sigma \times \{\varepsilon\} \times \{1\}$,
- and the transitions $\delta' = \bigcup_{\substack{a' \in \Sigma,\, b \in \{0,1\} \\ (q,a,\gamma,\gamma',q') \in \delta \\ \gamma'' \in \Gamma_\perp^{\leq 1}}} \Bigg($

$$\Big\{\big(\langle q, a', \gamma'', b\rangle, \varepsilon, \varepsilon, \langle q', a, \gamma'', b\rangle\big) \,\Big|\, \gamma = \gamma' = \varepsilon\Big\} \cup \tag{5.6}$$

$$\Big\{\big(\langle q, a', \gamma'', b\rangle, \varepsilon, \langle \gamma', b\rangle, \langle q', a, \gamma'', 0\rangle\big) \,\Big|\, \gamma = \varepsilon, \gamma' \neq \varepsilon\Big\} \cup \tag{5.7}$$

$$\Big\{\big(\langle q, a', \gamma'', 0\rangle, \langle \gamma, b\rangle, \varepsilon, \langle q', a, \gamma'', b\rangle\big) \,\Big|\, \gamma \neq \varepsilon, \gamma' = \varepsilon\Big\} \cup \tag{5.8}$$

$$\Big\{\big(\langle q, a', \gamma, 1\rangle, \varepsilon, \varepsilon, \langle q', a, \varepsilon, 1\rangle\big) \,\Big|\, \gamma \neq \varepsilon, \gamma' = \varepsilon\Big\} \cup \tag{5.9}$$

$$\Big\{\big(\langle q, a', \varepsilon, 1\rangle, \langle \gamma, 1\rangle, \varepsilon, \langle q', a, \varepsilon, 1\rangle\big) \,\Big|\, \gamma \neq \varepsilon, \gamma' = \varepsilon\Big\}\Bigg) \ . \tag{5.10}$$

While transitions that ignore the stack (5.6) or push to the stack (5.7) can be adopted straightforwardly, we have three variants of transitions that pop from the stack (5.8)–(5.10). If we have not reached the bottom of the stack yet, then we can pop symbols without problems (5.8). However, when only the initial stack symbol is left, which is indicated by 1 in the last component of the state, we first have to remove the stored symbol (5.9) before we can pop the initial stack symbol (5.10).

Let $w = a_1 \cdots a_n$ with $a_1, \ldots, a_n \in \Sigma$ and $n \geq 2$ be an input string. First, we assume that the first move of $\mathcal{A}$ pushes the symbol γ_1 to the stack, which then gets popped in the i-th move. Any such sequence of configurations

$$\begin{array}{llll}
 & \langle q_1,\, a_1 \cdots a_n,\, \perp\rangle & & \\
\vdash_{\mathcal{A}} & \langle q_2,\, a_2 \cdots a_n,\, \gamma_1\perp\rangle & \vdash_{\mathcal{A}} \langle q_3,\, a_3 \cdots a_n,\, \gamma_2\gamma_1\perp\rangle & \vdash_{\mathcal{A}} \cdots \\
\vdash_{\mathcal{A}} & \langle q_i,\, a_i \cdots a_n,\, \gamma_1\perp\rangle & \vdash_{\mathcal{A}} \langle q_{i+1},\, a_{i+1} \cdots a_n,\, \perp\rangle & \vdash_{\mathcal{A}} \cdots \\
\vdash_{\mathcal{A}} & \langle q_n,\, a_n,\, \perp\rangle & \vdash_{\mathcal{A}} \langle q_{n+1},\, \varepsilon,\, \varepsilon\rangle &
\end{array}$$

in $\mathcal{A}$ with $q_1 \in I$ and $q_{n+1} \in F$ yields the corresponding sequence of configurations

$$\begin{aligned}
&\langle\langle q_2, a_1, \gamma_1, 1\rangle, \langle\bot, 1\rangle\rangle \\
\vdash_{\mathcal{A}'} &\langle\langle q_3, a_2, \gamma_1, 0\rangle, \langle\gamma_2, 1\rangle\langle\bot, 1\rangle\rangle \vdash_{\mathcal{A}'} \cdots \\
\vdash_{\mathcal{A}'} &\langle\langle q_i, a_{i-1}, \gamma_1, 1\rangle, \langle\bot, 1\rangle\rangle \vdash_{\mathcal{A}'} \langle\langle q_{i+1}, a_i, \varepsilon, 1\rangle, \langle\bot, 1\rangle\rangle \vdash_{\mathcal{A}'} \cdots \\
\vdash_{\mathcal{A}'} &\langle\langle q_n, a_{n-1}, \varepsilon, 1\rangle, \langle\bot, 1\rangle\rangle \vdash_{\mathcal{A}'} \langle\langle q_{n+1}, a_n, \varepsilon, 1\rangle, \varepsilon\rangle
\end{aligned}$$

which is an accepting run of $\mathcal{A}'$ and vice versa. Similarly, if the first move of $\mathcal{A}$ ignores the stack, then $\mathcal{A}'$ starts in configuration $\langle\langle q_2, a_1, \varepsilon, 1\rangle, \langle\bot, 1\rangle\rangle$ to simulate the moves of $\mathcal{A}$. The case where the first move is a popping transition does not occur since then we would have $|w| = 1$. Thus, each string of length at least 2 accepted by $\mathcal{A}$ is also accepted by $\mathcal{A}'$ and vice versa, which proves $\mathcal{L}(\mathcal{A}') = \mathcal{L} = \mathcal{L}(\mathcal{A})$. □

Pop-Normalization

An MPDA $\mathcal{A}$ is *pop-normalized* if there exists a map $\mathrm{pop}\colon \Gamma \to Q$ such that $q' = \mathrm{pop}(\gamma)$ for every transition $(q, \gamma, \varepsilon, q') \in \delta$. In other words, for each stack symbol $\gamma \in \Gamma$ there is a unique state $\mathrm{pop}(\gamma)$ that the MPDA enters whenever γ is popped from the stack.

Later on, we will simulate the runs of an MPDA in a CCG such that subsequent configurations are represented by subsequent primary categories. Pop transitions are modeled by removing the last argument of a category. Thus, the target state has to be stored in the previous argument. This argument is added when the corresponding push transition is simulated, so at that point we already have to be aware in which state the MPDA will end up after popping the symbol again. This will be explained in more detail in Section 5.6.

We can easily establish this property by storing a state in each stack symbol. Each push transition is replaced by one variant for each state (i.e., we guess a state when pushing), but when a symbol is popped, this is only allowed if the state stored in it coincides with the target state.

Lemma 5.5.3 *For every MPDA we can construct an equivalent pop-normalized MPDA.*

Proof. Given an MPDA $\mathcal{A} = (Q, \Sigma, \Gamma, \delta, \tau, I, F)$, we extend each stack symbol by a state and let $\Gamma' = \Gamma \times Q$ as well as $\mathrm{pop} = \pi_2$, i.e., $\mathrm{pop}(\langle\gamma, q\rangle) = q$ for all $\langle\gamma, q\rangle \in \Gamma'$. All transitions that push a symbol to the stack guess the state that is entered when that symbol

is eventually popped. Hence we construct $\mathcal{A}' = (Q, \Sigma, \Gamma', \delta', \tau, I, F)$ with

$$\delta' = \bigcup_{(q,\gamma,\gamma',q')\in\delta} \Bigg(\Big\{(q, \varepsilon, \varepsilon, q') \;\Big|\; \gamma = \gamma' = \varepsilon\Big\} \cup \Big\{(q, \langle\gamma, q'\rangle, \varepsilon, q') \;\Big|\; \gamma' = \varepsilon\Big\} \cup \Big\{(q, \varepsilon, \langle\gamma', q''\rangle, q') \;\Big|\; \gamma = \varepsilon,\; q'' \in Q\Big\} \Bigg) .$$

It is obvious that for every accepting run of $\mathcal{A}$ there is an accepting run of $\mathcal{A}'$, in which all the guesses were correct. Note that $\mathcal{A}'$ starts with an arbitrary symbol on the stack, so we can find a run where the second component of this symbol coincides with the final state that is reached by popping this symbol. Similarly, every accepting run of $\mathcal{A}'$ can be translated into an accepting run of $\mathcal{A}$ by projecting each stack symbol to its first component. Hence $\mathcal{A}$ and $\mathcal{A}'$ are equivalent and $\mathcal{A}'$ is clearly pop-normalized. □

Lookahead

The next statement shows that we can provide a form of lookahead on the output. In each new symbol we store the current as well as the next output symbol. We will briefly sketch why this lookahead is necessary. Before constructing the CCG, the MPDA will be used to model the spine grammar. The next output symbol of the MPDA corresponds to the label of the parent node along the spine of a tree generated by the spine grammar. From this parent node we can determine the label of its other child, which is located next to the spine. This information will be used in the CCG to control which secondary categories are allowed as neighboring combination partners.

Lemma 5.5.4 *Let $\mathcal{L} \subseteq \Sigma^*$ with $\triangleleft \notin \Sigma$ be a context-free language. Then the language* $\mathrm{Next}(\mathcal{L})$ *is context-free as well, where*

$$\mathrm{Next}(\mathcal{L}) = \bigcup_{n\in\mathbb{N}} \Big\{\langle\sigma_2, \sigma_1\rangle \cdots \langle\sigma_n, \sigma_{n-1}\rangle\langle\triangleleft, \sigma_n\rangle \;\Big|\; \sigma_1, \ldots, \sigma_n \in \Sigma,\; \sigma_1 \cdots \sigma_n \in \mathcal{L}\Big\} .$$

Proof. We define $\Sigma' = \Sigma \cup \{\triangleleft\}$ and $\Sigma'' = \Sigma' \times \Sigma$. Let us consider the homomorphism $\pi_2 \colon (\Sigma'')^* \to \Sigma^*$ given by $\pi_2\big(\langle\sigma', \sigma\rangle\big) = \sigma$ for all $\langle\sigma', \sigma\rangle \in \Sigma''$. Since the context-free languages are closed under inverse homomorphisms [9, Chapter 2, Theorem 2.1], the

language $\mathcal{L}'' = \pi_2^{-1}(\mathcal{L})$ is also context-free. Moreover, the language

$$\mathcal{R} = \bigcup_{n \in \mathbb{N}} \big\{ \langle \sigma_2, \sigma_1 \rangle \langle \sigma_3, \sigma_2 \rangle \cdots \langle \sigma_n, \sigma_{n-1} \rangle \langle \triangleleft, \sigma_n \rangle \;\big|\; \sigma_1, \dots, \sigma_n \in \Sigma \big\}$$

is a regular language because it is recognized by the NFA given by $\mathcal{A} = \big(\Sigma', \Sigma'', \delta, \Sigma', \{\triangleleft\}\big)$ with state set and initial states Σ', input alphabet Σ'', final states $\{\triangleleft\}$, and transitions

$$\delta = \big\{ (\sigma, \langle \sigma', \sigma \rangle, \sigma') \;\big|\; \sigma \in \Sigma,\ \sigma' \in \Sigma' \big\} \ .$$

Finally, $\mathrm{Next}(\mathcal{L}) = \mathcal{L}'' \cap \mathcal{R}$ is context-free because the intersection of a context-free language with a regular language is again context-free [9, Chapter 2, Theorem 2.1]. □

Corollary 5.5.5 *For every context-free language $\mathcal{L} \subseteq \Sigma^{\geq 2}$ there exists a pop-normalized MPDA $\mathcal{A}$ such that $\mathcal{L}(\mathcal{A}) = \mathrm{Next}(\mathcal{L})$.*

Representing Spines

We will now add the aforementioned lookahead to the set of spines and use the MPDA to recognize this language. Since the automaton only accepts words of length at least 2, words of length 1 have to be treated as a separate set $\mathcal{L}_1$. The words with lookahead can be recursively assembled in the same manner as those without. Through an appropriate projection to the current symbol before removing the other annotations, we again obtain the original tree language $\mathcal{T}(\mathcal{G})$.

Corollary 5.5.6 *Let $\mathcal{G}$ be a normalized spine grammar. Additionally, let $\mathcal{L}_1 = \big\{ w \in \mathrm{Next}\big(\mathcal{S}(\mathcal{G})\big) \mid |w| = 1 \big\}$. Then there exists a pop-normalized MPDA $\mathcal{A}$ with $\mathcal{L}(\mathcal{A}) \cup \mathcal{L}_1 = \mathrm{Next}\big(\mathcal{S}(\mathcal{G})\big)$. Moreover, the tree languages $\mathcal{F}\big(\mathcal{L}(\mathcal{A}) \cup \mathcal{L}_1\big)_s$ and $\mathcal{T}(\mathcal{G})$ coincide modulo relabeling.*

Proof. Without loss of generality (see Theorem 5.3.4), we may assume that $\mathcal{G} = (N, \Sigma, S, P)$ is a normalized spine grammar. Due to Definition 5.4.1, the spines $\mathcal{S}(\mathcal{G})$ are a context-free subset of $A_0 A_2^*$ with $A_0 = \Sigma_0 \times N$ and $A_2 = \Sigma_2 \times N^2$. Using Corollary 5.5.5, we obtain a pop-normalized MPDA $\mathcal{A}$ recognizing the language $\mathcal{L}(\mathcal{A}) = \big\{ w \in \mathrm{Next}\big(\mathcal{S}(\mathcal{G})\big) \mid |w| \geq 2 \big\}$. Moreover, we observe that $\mathcal{L}(\mathcal{A}) \cup \mathcal{L}_1 \subseteq (A_2' \times A_0)(A_2' \times A_2)^*$ with $A_2' = A_2 \cup \{\triangleleft\}$. Clearly, $\mathcal{L}(\mathcal{A}) \cup \mathcal{L}_1$ relabels to $\mathcal{S}(\mathcal{G})$ via the projection to the components of A_0 and A_2. Consider the ranked alphabet Σ' given by $\Sigma_0' = A_2' \times \Sigma_0$, $\Sigma_1' = \emptyset$, and $\Sigma_2' = A_2' \times \Sigma_2$. Then the tree language $\mathcal{F}\big(\mathcal{L}(\mathcal{A}) \cup \mathcal{L}_1\big) \subseteq T_{\Sigma_2' \times N^2, \emptyset}(\Sigma_0' \times N)$ can be relabeled

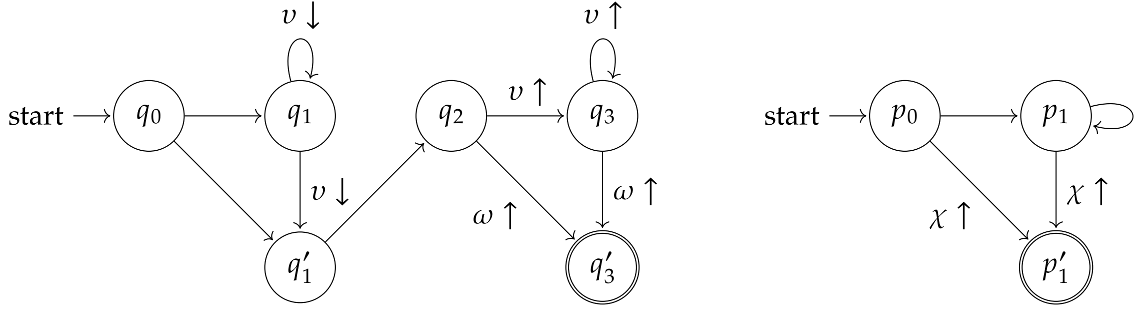


Figure 5.8: Sample MPDA (see Example 5.5.7).

to $\mathcal{F}(\mathcal{S}(\mathcal{G})) \subseteq T_{\Sigma_2\times N^2,\emptyset}(\Sigma_0\times N)$ via the projection to the components of $\Sigma_0\times N$ and $\Sigma_2\times N^2$. By Theorem 5.4.7, the tree language $\mathcal{F}(\mathcal{S}(\mathcal{G}))_s$ can be relabeled to $\mathcal{T}(\mathcal{G})$, proving that $\mathcal{F}(\mathcal{L}(\mathcal{A}) \cup \mathcal{L}_1)_s$ and $\mathcal{T}(\mathcal{G})$ coincide modulo relabeling. □

Example 5.5.7 The MPDA constructed in Corollary 5.5.6 for the spine grammar $\mathcal{G}$ of Example 5.3.3 is depicted in Figure 5.8. Initial states are indicated using a start marker and final states are marked by a double circle. Push and pop stack operations are written with downwards and upwards arrows, respectively. The MPDA consists of two components. The larger one describes the main spine, and the smaller one describes the side spine. The distinction between the three stack symbols is necessary due to pop-normalization, and the distinction between q_1 and q_1' (and similar states) is necessary because of the lookahead provided by $\text{Next}(\mathcal{S}(\mathcal{G}))$. For example, $\tau(q_1) = \big(\frac{\gamma_2}{s\ \ \bar{c}}, \frac{\gamma_2}{s\ \ \bar{c}}\big)$ and $\tau(q_1') = \big(\frac{\beta_2}{s\ \ \bar{b}}, \frac{\gamma_2}{s\ \ \bar{c}}\big)$. Similarly, $\tau(p_1) = (z, z)$ and $\tau(p_1') = (\triangleleft, z)$ where $z = \frac{\eta_2}{\bar{e}\ \ \bar{b}}$. To completely capture the behavior of $\mathcal{G}$, we additionally require the set $\mathcal{L}_1 = \{(\triangleleft, \alpha_{\bar{a}}), (\triangleleft, \beta_{\bar{b}}), (\triangleleft, \beta_{\bar{e}}), (\triangleleft, \gamma_{\bar{c}})\}$, which contains the spines of length 1.

5.6 CCG Construction

In this section, let $\mathcal{G} = (N, \Sigma, \{s\}, P)$ be a normalized spine grammar with spine direction $d\colon \Sigma_2 \to [2]$. Using Corollary 5.5.6, the pop-normalized MPDA $\mathcal{A} = (Q, \Delta, \Gamma, \delta, \tau, I, F)$ with the mapping $\text{pop}\colon \Gamma \to Q$ is constructed for $\mathcal{G}$. We note that $\Delta = \Sigma' \times \Sigma''$ with $\Sigma' = \{\triangleleft\} \cup (\Sigma_2 \times N^2)$ and $\Sigma'' = (\Sigma_0 \times N) \cup (\Sigma_2 \times N^2)$. Moreover, let $\bot \notin Q$ be a special symbol. To provide better access to the components of the MPDA $\mathcal{A}$, we define some additional maps.

The spine generator $\text{gen}\colon Q \to N$ is given by $\text{gen}(q) = \text{gen}(s_2)$ for every state $q \in Q$, where $\tau(q) = (s_1, s_2) \in \Delta$. Since $\mathcal{A}$ cannot accept strings of length 1, we have to treat them separately. Let

$\mathcal{L}_1 = \{w \in \mathrm{Next}(\mathcal{S}(\mathcal{G})) \mid |w| = 1\}$ and $\mathrm{gen}\colon \mathcal{L}_1 \to N$ be given by $\mathrm{gen}(w) = n$ for all $w = (\triangleleft, \alpha_n) \in \mathcal{L}_1$. We extend $\tau\colon Q \to \Delta$ to $\tau'\colon (Q \cup \mathcal{L}_1) \to \Delta$ by $\tau'(q) = \tau(q)$ for all $q \in Q$ and $\tau'(a) = a$ for short strings $a \in \mathcal{L}_1$.

Recall that $D = \{/, \backslash\}$. Given state $q \in Q \setminus F$, the slash type $\mathrm{slash}\colon (Q \setminus F) \to D$ specifies whether the symbol $\tau(q)$ produced by state q occurs as the first or second child of its parent symbol. The combining nonterminal $\mathrm{comb}\colon (Q \setminus F) \cup \{\bot\} \to N$ denotes which spine generator the symbol $\tau(q)$ is combined with. Let $\tau(q) = \left(\frac{\sigma}{n_1\ n_2}, s_2\right)$ with $\frac{\sigma}{n_1\ n_2} \in \Sigma_2 \times N^2$ and $s_2 \in \Sigma''$. The slash type and the combining nonterminal can be determined from the next symbol $\frac{\sigma}{n_1\ n_2}$. Formally, $\mathrm{slash}(q) = /$ if $d(\sigma) = 1$ and $\mathrm{slash}(q) = \backslash$ otherwise. Further, $\mathrm{comb}(q) = n_{3-d(\sigma)}$ and $\mathrm{comb}(\bot) = s$.

We simulate the accepting runs of $\mathcal{A}$ in the spines consisting of primary categories of the CCG. These *primary spines* are paths in a CCG derivation tree that start with a lexical category at a leaf of the derivation tree and consist of a (possibly empty) sequence of primary categories followed by a secondary or initial category, which we consider as the end of the spine. The shortest possible primary spine is a single lexical category that serves as a secondary or initial category. The main idea is that each category on the primary spine stores a configuration of $\mathcal{A}$. The last argument stores the current state in the first component and the top of the stack in the second component. The previous arguments store in their second components the preceding stack symbols and in their first components the state the automaton returns to when the stack symbol stored in the subsequent argument is popped. For each push transition, an additional argument is added, whereas for each pop transition, an argument is removed. With this intuition, the rule system directly implements the transitions of $\mathcal{A}$. The translation from transitions to combinatory rules is illustrated in the following simplified examples, where we omitted an additional third component of each atom, which is needed for relabeling. The stack symbol $\psi \in \{\omega, \upsilon, \chi\}$ is arbitrary and remains unchanged. For the pushing transition, note that $\mathrm{pop}(\upsilon) = q_3$.

$$q_0 \longrightarrow q_1 \qquad \frac{ax/\binom{q_0}{\psi} \quad \binom{q_0}{\psi}/\binom{q_1}{\psi}}{ax/\binom{q_1}{\psi}}$$

$$q_1 \xrightarrow{\upsilon\downarrow} q_1' \qquad \frac{ax/\binom{q_1}{\psi} \quad \binom{q_1}{\psi}\backslash\binom{q_3}{\psi}/\binom{q_1'}{\upsilon}}{ax\backslash\binom{q_3}{\psi}/\binom{q_1'}{\upsilon}}$$

$$p_0 \xrightarrow{\chi\uparrow} p_1' \qquad \frac{ax/\binom{p_0}{\chi} \quad \binom{p_0}{\chi}}{ax}$$

As can be seen above, to implement the required transformations of consecutive primary categories, the secondary categories need to have a specific structure. This mandates that the categories at the end of a spine cannot store their corresponding automaton state in the first component of the last argument as usual, but instead utilize the third component of their target. Thus, each argument uses the third component to store the final state or symbol corresponding to the combining category of the attaching side spine.[8] The set of atoms is defined in such a way that the symbol associated with the third component is a valid sibling of the symbol associated with the first component. This is verified by means of the annotated lookahead, as the generator of the sibling is the non-spinal child of the successor symbol. The third component also allows us to decide whether a category is primary: A category is a primary category if and only if the spine generator of the state stored in the first component of the last argument and the spine generator of the third component of the target coincide. This is possible since $\mathcal{G}$ is normalized, which yields that attaching spines have a spine generator that is different from the spine generator of the spine that they attach to.

8: This is similar to the approach employed in Section 4.1 for 0-CCG. Since the relabeling of the primary and secondary category was based on the same atom, an additional component was required to relabel the secondary category, which constituted the end of a spinal run.

The lexicon assigns categories to symbols that can label leaves, thus these symbols are taken from the nullary terminal symbols. The spines of length 1 are translated directly to secondary categories or initial categories and added to the lexicon. For non-trivial spines, the assigned categories consist of a category that appears at the end of such a spine plus an additional argument for the initial state and initial stack symbol of an accepting run. This stack symbol has the property that, when it is popped, $\mathcal{A}$ enters the final state stored in the target of the category.

Definition 5.6.1 *We define the CCG* $\mathcal{G}_{\mathcal{A},\mathcal{L}_1} = (\Delta_0, A, R, I', L)$ *as follows.*

We define the set of atoms $A = \{(q, \gamma, f) \in A' \mid \mathrm{gen}(f) = \mathrm{comb}(q)\}$ *with* $A' = (Q \cup \{\bot\}) \times \Gamma^{\leq 1} \times (F \cup \mathcal{L}_1)$. *We write* a_i *to refer to the i-th component of an atom* $a \in A$. *Additionally, let the initial atomic categories given by* $I' = \{(\bot, \varepsilon, f) \in A \mid \mathrm{gen}(f) = s\}$.

In the rules $R = \bigcup_{| \in D} (R_1^{|} \cup R_2^{|} \cup R_3^{|})$ *we overline the primary category* ax/b, *which always needs to fulfill* $\mathrm{gen}(a_3) = \mathrm{gen}(b_1)$.

$$R_1^{/} = \bigcup_{\substack{a,b,c \in A,\, | \in D \\ (b_1,\varepsilon,\varepsilon,c_1) \in \delta \\ b_2 = c_2}} \left\{ \frac{\overline{ax/b} \qquad b|c}{ax|c} \right\}$$

$$R_2^/ = \bigcup_{\substack{a,b,c,e \in A,\, |,|' \in D \\ (b_1,\varepsilon,e_2,e_1) \in \delta \\ b_2 = c_2 \\ c_1 = \mathrm{pop}(e_2)}} \left\{ \frac{\overline{ax/b} \qquad b|c|'e}{ax|c|'e} \right\}$$

$$R_3^/ = \bigcup_{\substack{a,b \in A \\ (b_1,b_2,\varepsilon,q) \in \delta}} \left\{ \frac{\overline{ax/b} \qquad b}{ax} \right\}$$

We listed all the forward rules, but for each forward rule there also exists a symmetric backward rule yielding the rule sets $R_1^\backslash$, $R_2^\backslash$, and $R_3^\backslash$.

We require some notions for the lexicon. A category $c \in \mathcal{C}(A)$ is well-formed *if it is first-order and we have $| = \mathrm{slash}(b_1)$, $b_1 \in Q$, and $b_2 \in \Gamma$ for every $i \in [\mathrm{arity}(c)]$ with $|b = \mathrm{arg}(c, i)$. Let them be denoted by $C_{\mathrm{wf}} = \{c \in \mathcal{C}(A) \mid c \text{ well-formed}\}$. Clearly $I' \subseteq C_{\mathrm{wf}}$.*

In addition, we introduce sets $\top_{\mathcal{L}_1}$ and $\top_{\mathcal{A}}$ of end-of-spine categories derived from the short strings of $\mathcal{L}_1$ and the strings accepted by $\mathcal{A}$, respectively, where $\mathrm{sec}(r)$ to refers to the secondary category of rule r:

$$\top_{\mathcal{L}_1} = \left\{a \in I' \;\middle|\; a_3 \in \mathcal{L}_1\right\} \cup \bigcup_{\substack{r \in R \\ ax = \mathrm{sec}(r)}} \left\{ax \in C_{\mathrm{wf}} \;\middle|\; a_3 \in \mathcal{L}_1\right\}$$

$$\top_{\mathcal{A}} = \left\{a \in I' \;\middle|\; a_3 \in F\right\} \cup \bigcup_{\substack{r \in R \\ ax = \mathrm{sec}(r)}} \left\{ax \in C_{\mathrm{wf}} \;\middle|\; a_3 \in F\right\}$$

Note that $\top_{\mathcal{L}_1} \cup \top_{\mathcal{A}} \subseteq C_{\mathrm{wf}}$. For all $\alpha \in \Delta_0 = \Sigma' \times (\Sigma_0 \times N)$ we define the lexicon as follows:

$$L(\alpha) = \left\{ ax \;\middle|\; \begin{array}{l} ax \in \top_{\mathcal{L}_1} \\ \tau'(a_3) = \alpha \end{array} \right\} \cup \left\{ ax|b \in C_{\mathrm{wf}} \;\middle|\; \begin{array}{ll} ax \in \top_{\mathcal{A}}, & \mathrm{gen}(a_3) = \mathrm{gen}(b_1) \\ b_1 \in I, & \mathrm{pop}(b_2) = a_3 \\ & \tau'(b_1) = \alpha \end{array} \right\}$$

The relabeling $\rho \colon C_{\mathrm{wf}} \to \Delta$ is defined for every $a \in A$ by $\rho(a) = \tau'(a_3)$ and for every $ax|b \in C_{\mathrm{wf}}$ by

$$\rho(ax|b) = \begin{cases} \tau'(b_1) & \text{if } \mathrm{gen}(a_3) = \mathrm{gen}(b_1) \\ \tau'(a_3) & \text{otherwise} \end{cases}$$

We will first be concerned with some general properties to establish that the relabeling actually operates as intended. Let us start with two general observations that hold for all categories that appear in derivation trees of $\mathcal{G}_{\mathcal{A},\mathcal{L}_1}$.

Observation 5.6.2 For all categories that appear in derivation trees of $\mathcal{G}_{\mathcal{A},\mathcal{L}_1}$ holds the following:

1. All categories are well-formed. This follows from the fact that only well-formed categories occur in the lexicon and all categories in the derivation trees consist of atoms and arguments that were already present in the lexicon (see Proposition 3.3.10).
2. All primary categories $ax|b$ obey $\text{gen}(a_3) = \text{gen}(b_1)$. This is directly required by the rule system.

The following property holds because the spine grammar $\mathcal{G}$ is normalized, so a spine never has the same spine generator as its attached spines.

Lemma 5.6.3 *For all secondary categories $ax|b$ appearing in derivation trees of $\mathcal{G}_{\mathcal{A},\mathcal{L}_1}$ we have* $\text{gen}(a_3) \neq \text{gen}(b_1)$.

Proof. We have $\text{gen}(a_3) = \text{comb}(a_1)$ by the definition of atoms A. Additionally, we have $\text{gen}(a_1) = \text{gen}(b_1)$ by the construction of the rule system, since $a_1, b_1 \in Q$ occur in a single transition of $\mathcal{A}$. However, the spine generator $\text{gen}(a_1)$ never coincides with the spine generator $\text{comb}(a_1)$ of an attaching spine due to the normalization of $\mathcal{G}$. This means that $\text{comb}(a_1) \neq \text{gen}(a_1)$. We can conclude that $\text{gen}(a_3) = \text{comb}(a_1) \neq \text{gen}(a_1) = \text{gen}(b_1)$. □

We are now ready to describe the general form of primary spines of $\mathcal{G}_{\mathcal{A},\mathcal{L}_1}$. Given a primary spine $c_0 \cdots c_n$ with $n \geq 1$ read from lexicon entry towards the root, we know that it starts with a lexicon entry $c_0 = ax \mid b \in L(\Delta_0)$ and ends with the non-primary category ax, which as such cannot be further modified. Hence each of the categories $c \in \{c_0, \dots, c_{n-1}\}$ has the form $ax|_1 b_1 \cdots |_m b_m$ with $m \in \mathbb{N}_+$. Let $b_i = (q_i, \gamma_i, f_i)$ for every $i \in [m]$. The category c_n is relabeled to $\tau'(a_3)$ and c is relabeled to $\tau'(q_m)$. Additionally, unless $a_1 = \bot$, which applies to the main spine, the first components of all atoms in ax have the same spine generator $\text{gen}(a_1)$ and $\text{gen}(q_1) = \cdots = \text{gen}(q_m)$, but $\text{gen}(a_1) \neq \text{gen}(q_1)$. Finally, neighboring arguments $|_{i-1} b_{i-1} |_i b_i$ in the suffix are coupled such that $\text{pop}(\gamma_i) = q_{i-1}$ for all $i \in [m] \setminus \{1\}$. This coupling is introduced in the rules of second degree and preserved by the other rules.

Using these observations, it can be proved that the primary spines of $\mathcal{G}_{\mathcal{A},\mathcal{L}_1}$ are relabeled to strings of $\text{Next}(\mathcal{S}(\mathcal{G}))$ and vice versa. Moreover, spines attach in essentially the same manner in the CCG and using $\mathcal{F}$. This yields the result that, given a spine grammar, it is possible to construct a CCG that generates the same tree language. We will prove the correctness of our construction in the

Figure 5.9: Derivation tree of $\mathcal{G}_{\mathcal{A},\mathcal{L}_1}$ (see Example 5.6.4).

following two subsections. But first, we will illustrate it by means of an example.

Example 5.6.4 Figure 5.9 shows the derivation tree of CCG $\mathcal{G}_{\mathcal{A},\mathcal{L}_1}$ that corresponds to the tree of Figure 5.7a, which is generated by the spine grammar $\mathcal{G}$ of Example 5.3.3. We use the following abbreviations: $\alpha = (\triangleleft, \alpha_{\bar{a}})$, $\beta = (\triangleleft, \beta_{\bar{e}})$, and $\gamma = (\triangleleft, \gamma_{\bar{c}})$. The labelings of the (non-trivial) spines are $\delta\, \gamma_2\, \gamma_2\, \beta_2\, \alpha_2\, \alpha_2$ for the main spine and $\beta\, \eta_2$ for the side spine (see Figure 5.7a). They correspond to the following runs of $\mathcal{A}$ (see Example 5.5.7, Figure 5.8):

$$(\langle q_0, \omega\rangle, \langle q_1, \omega\rangle, \langle q_1', \upsilon\omega\rangle, \langle q_2, \upsilon\omega\rangle, \langle q_3, \omega\rangle, \langle q_3', \varepsilon\rangle)$$
$$(\langle p_0, \chi\rangle, \langle p_1', \varepsilon\rangle)$$

The components storing the configurations of runs are highlighted in red for the main spine and blue for the side spine. In secondary categories that constitute trivial spines of length 1, the components responsible for relabeling are highlighted in green.

Let us observe how the transitions of $\mathcal{A}$ are simulated by $\mathcal{G}_{\mathcal{A},\mathcal{L}_1}$. The first transition $(q_0, \varepsilon, \varepsilon, q_1)$ on the main spine does not modify the stack. It is implemented by replacing the last argument $/(q_0, \omega, \gamma)$ by $/(q_1, \omega, \gamma)$. The next transition $(q_1, \varepsilon, \upsilon, q_1')$ pushes the symbol υ to the stack. The argument $/(q_1, \omega, \gamma)$ is thus replaced by two arguments $\backslash(q_3, \omega, \alpha)/(q_1', \upsilon, p_1')$. As the stack grows, an additional argument with the new state and stack symbol is added. The previous argument stores $\mathrm{pop}(\upsilon) = q_3$ to ensure that we enter the correct state after popping υ. It also contains the previous unchanged stack symbol ω. The pop transition $(p_0, \chi, \varepsilon, p_1')$ on the side spine run is realized by removing $\backslash(p_0, \chi, \beta)$.

The third components are required to relabel the non-primary categories. At the start of the main spine, $c_1 = (\bot, \varepsilon, q_3')/(q_0, \omega, \gamma)$ is a primary category because q_0 and q_3' are associated with the same spine generator s. Thus, c_1 gets relabeled to $\tau'(q_0)$. However, for $c_2 = (q_0, \omega, \gamma)/(q_1, \omega, \gamma)$, the spine generators of γ and of the output of q_1 are different ($\overline{c}$ and s). Hence it is a non-primary category and gets relabeled to γ.

Concerning the lexicon, c_1 is a lexical category due to the fact that $(\bot, \varepsilon, q_3') \in \top_{\mathcal{A}}$ can appear at the end of a spine as an initial category with $q_3' \in F$ in its third component, while the appended $/(q_0, \omega, \gamma)$ represents an initial configuration of $\mathcal{A}$. Similarly, c_2 is a well-formed secondary category of a rule and the third component of its target is in $\mathcal{L}_1$. Therefore, it is an element of $\top_{\mathcal{L}_1}$, which is a subset of the lexicon.

Let us illustrate how the attachment of the side spine to the main spine is realized. In category $(q_1', \upsilon, p_1')\backslash(q_2, \upsilon, \alpha)\backslash(p_0, \chi, \beta)$, which is contained in the lexicon, the first two atoms are responsible for performing a transition on the main spine. This part cannot be modified since the rule system disallows it. The target stores the final state p_1' of the side spine run in its third component. The appended argument models the initial configuration of the side spine run starting in state p_0 with χ on the stack.

5.6.1 Relating CCG Spines and Automaton Runs

We assume the symbols that were introduced at the beginning of the section. In particular, let $\mathcal{A} = (Q, \Delta, \Gamma, \delta, \tau, I, F)$ be the pop-normalized MPDA, let $\mathcal{L}_1 = \{w \in \text{Next}(\mathcal{S}(\mathcal{G})) \mid |w| = 1\}$ be the short strings not captured by $\mathcal{A}$, and let $\mathcal{G}_{\mathcal{A},\mathcal{L}_1} = (\Delta_0, A, R, I', L)$ be the constructed CCG. We start with discussing the spines before we move on to the discussion of how those spines attach to each other in the next subsection.

Lemma 5.6.5 *Every primary spine of a derivation tree of $\mathcal{D}(\mathcal{G}_{\mathcal{A},\mathcal{L}_1})$ read from leaf to initial or secondary category relabels via ρ to a string $w \in \text{Next}\big(\mathcal{S}(\mathcal{G})\big)$.*

Proof. We start with spines of length 1. Their single category is obviously taken from the lexicon and thus either an initial atomic category a or a secondary category ax. Both of those categories are relabeled to $a_3 \in \mathcal{L}_1 \subseteq \text{Next}\big(\mathcal{S}(\mathcal{G})\big)$.

Now consider a primary spine $c_0 \cdots c_n$ with $n \geq 1$. We have to show that there is an accepting run of $\mathcal{A}$ corresponding to this spine. We have already described the general form of these spines.

There exists a category bx, and for each $i \in \{0, \dots, n\}$, there exist $m \in \mathbb{N}$, slashes $|_1, \dots, |_m \in D$, and atoms $a_1, \dots, a_m \in A$ such that category c_i has the form $bx|_1a_1 \cdots |_m a_m$. In particular, we have $c_0 = bx|a$ for some $| \in D$ and $a \in A$ as well as $c_n = bx$. For better readability, we address the j-th component of atom a_i by $a_{i,j}$ as an abbreviation for $(a_i)_j$, where $i \in [m]$ and $j \in [3]$. We translate each category $bx|_1a_1 \cdots |_m a_m$ to a configuration of $\mathcal{A}$ via the mapping conf: $\mathcal{C}_0(A) \to \mathrm{Conf}_{\mathcal{A}}$ in the following manner:

$$\mathrm{conf}(bx|_1a_1 \cdots |_m a_m) = \begin{cases} \langle b_3,\ \varepsilon \rangle & \text{if } m = 0 \\ \langle a_{m,1},\ a_{m,2} \cdots a_{1,2} \rangle & \text{otherwise} \end{cases}$$

In other words, the state of the configuration corresponding to c_n is the third component of the target b, whereas all other categories $c_0, \dots, c_{n-1}$ store the state in the first component of the last argument. The stack content is represented by the second components of the suffix $|_1a_1 \cdots |_m a_m$. Each category relabels to the input symbol produced by its respective stored state. Thus, if $\big(\mathrm{conf}(c_0), \dots, \mathrm{conf}(c_n)\big)$ is an accepting run of $\mathcal{A}$, then it generates the same string that the spine is relabeled to.

It remains to show that $\big(\mathrm{conf}(c_0), \dots, \mathrm{conf}(c_n)\big)$ is actually an accepting run. Since c_0 is assigned by the lexicon and has a suffix behind bx consisting of a single argument with the first component in I, whereas c_n has an empty suffix and $b_3 \in F$, it is easy to see that c_0 and c_n correspond to an initial and a final configuration, respectively. Hence we only need to prove that configurations corresponding to subsequent categories c_i and c_{i+1} with $i \in \{0, \dots, n-1\}$ are connected by valid moves. To this end, we distinguish three cases based on the rule r that is used to derive output category c_{i+1} from primary category c_i:

1. Let $r \in R_1^{|}$ with $| \in D$. Moreover, let $c_i = bx|_1a_1 \cdots |_m a_m$ with $|_m = |$ and $c_{i+1} = bx|_1a_1 \cdots |_{m-1}a_{m-1}|'_m a'_m$. These categories correspond to configurations $\langle a_{m,1},\ a_{m,2} \cdots a_{1,2} \rangle$ and $\langle a'_{m,1},\ a'_{m,2}a_{m-1,2} \cdots a_{1,2} \rangle$, respectively. The definition of $R_1^{|}$ implies $a_{m,2} = a'_{m,2}$ as well as the existence of the transition $(a_{m,1}, \varepsilon, \varepsilon, a'_{m,1}) \in \delta$ of $\mathcal{A}$ that enables a valid move.
2. Let $r \in R_2^{|}$ with $| \in D$. Moreover, let $c_i = bx|_1a_1 \cdots |_m a_m$ with $|_m = |$ and $c_{i+1} = bx|_1a_1 \cdots |_{m-1}a_{m-1}|'_m a'_m|'_{m+1}a'_{m+1}$. The corresponding configurations are $\langle a_{m,1},\ a_{m,2} \cdots a_{1,2} \rangle$ and $\langle a'_{m+1,1},\ a'_{m+1,2}a'_{m,2}a_{m-1,2} \cdots a_{1,2} \rangle$, respectively. The definition of $R_2^{|}$ implies $a_{m,2} = a'_{m,2}$ as well as the existence of the transition $(a_{m,1}, \varepsilon, a'_{m+1,2}, a'_{m+1,1}) \in \delta$, which again enables a valid move.
3. Let $r \in R_3^{|}$ with $| \in D$. Moreover, let $c_i = bx|_1a_1 \cdots |_m a_m$ with $|_m = |$ and $c_{i+1} = bx|_1a_1 \cdots |_{m-1}a_{m-1}$. We distinguish two subcases. First, let $m > 1$. Then the corresponding config-

urations are $\langle a_{m,1},\ a_{m,2} \cdots a_{1,2}\rangle$ and $\langle a_{m-1,1},\ a_{m-1,2} \cdots a_{1,2}\rangle$, respectively. Due to the coupling of neighboring arguments, we have $\mathrm{pop}(a_{m,2}) = a_{m-1,1}$, which ensures that we reach the correct target state. Since $\mathcal{A}$ is pop-normalized, the target state of the pop-transition is completely determined by the popped symbol $a_{m,2}$. The definition of the rule set $R_3^{|}$ implies the existence of the transition $(a_{m,1}, a_{m,2}, \varepsilon, a_{m-1,1}) \in \delta$, which again makes the move valid.
Now let $m = 1$, which yields $c_i = bx \,|\, a_1$ and $c_{i+1} = bx$. The corresponding configurations are $\langle a_{1,1},\ a_{1,2}\rangle$ and $\langle b_3,\ \varepsilon\rangle$, where $\mathrm{pop}(a_{1,2}) = b_3$ because the initial stack symbol $a_{1,2}$ was assigned by the lexicon and cannot be modified without removing the argument. The definition of $R_3^{|}$ implies the existence of the transition $(a_{1,1}, a_{1,2}, \varepsilon, b_3) \in \delta$, which also concludes this subcase.

We have seen that each spine $c_0 \cdots c_n$ with $n \geq 1$ corresponds to an accepting run $\mathrm{conf}(c_0) \vdash_{\mathcal{A}} \cdots \vdash_{\mathcal{A}} \mathrm{conf}(c_n)$ whose output $w \in \mathcal{L}(\mathcal{A}) \subseteq \mathrm{Next}\big(\mathcal{S}(\mathcal{G})\big)$ coincides with the relabeling of the spine via ρ. □

We now turn our attention to the inverse direction. More precisely, we will show that, given a string $w = w_0 \cdots w_n \in \mathrm{Next}\big(\mathcal{S}(\mathcal{G})\big)$ with $w_i \in \Delta$ for $i \in \{0, \ldots, n\}$, we can find a primary spine $c_0 \cdots c_n$ of $\mathcal{G}_{\mathcal{A},\mathcal{L}_1}$ (i.e., a sequence of primary categories starting at a category that belongs to $L(\Delta_0)$ and ending in a non-primary category) that gets relabeled to it via ρ. Further, for this spine we have some freedom in the selection of c_n, which is the category at the end. More precisely, for every well-formed secondary or initial category that relabels to w_n, we can essentially choose this category, but it might be necessary to change the third component of the target to another value (that still outputs w_n) to ensure consistency with the rest of the spine. Additionally, the third component of the last argument in the suffix (so in all categories except for c_n) can be chosen freely from the set of strings of length 1 or final states under the condition that it has the correct spine generator. This will be of great importance when we combine these spines to a complete derivation tree. We will start with the short spines, which correspond to lexicon entries.

Lemma 5.6.6 *Let $w \in \mathcal{L}_1$ and $c \in \big\{ax \in L(\Delta_0) \mid a_3 = w\big\}$. Then there exists a primary spine of $\mathcal{G}_{\mathcal{A},\mathcal{L}_1}$ that is labeled by c and relabeled to w via ρ.*

Proof. The set $C = \big\{ax \in L(\Delta_0) \mid a_3 = w\big\}$ is clearly a subset of $L(\Delta_0)$ and each $ax \in C$ is either a secondary category or an

initial category by the construction of L since $a_3 \in \mathcal{L}_1$. In either case ax is relabeled to $a_3 = w$ and cannot be modified by the rules of $\mathcal{G}_{\mathcal{A},\mathcal{L}_1}$. Consequently, these categories themselves constitute complete primary spines of length 1. □

For longer spines, we assume that we are given a run of $\mathcal{A}$. We can choose a category c that is either a secondary or initial category and will appear at the end of the corresponding primary spine. This category has to store the final state of the run in the third component of the target, but the freedom we have otherwise will be needed later to attach to other spines in different ways or to use it as the main spine. Additionally, we can choose $e_0, \dots, e_{n-1} \in F \cup \mathcal{L}_1$ from the possible siblings that can appear next to the symbols on the spine, respectively. This will be needed later for the relabeling of the respective secondary categories, which again are the end of some attaching spine. Using these prerequisites, a primary spine is constructed starting from the lexicon entry at the leaf.

Lemma 5.6.7 *Assume we are given*

- *an accepting run* $(\langle q_0, \beta_0 \rangle, \dots, \langle q_n, \beta_n \rangle)$ *of* $\mathcal{A}$,
- *well-formed category* $c \in \{ax \mid ax|b \in L(\Delta_0), a_3 = q_n\}$, *and*
- $e_0, \dots, e_{n-1} \in F \cup \mathcal{L}_1$ *such that* $\mathrm{gen}(e_i) = \mathrm{comb}(q_i)$ *for all* $i \in \{0, \dots, n-1\}$.

Then there exists a primary spine $c_0 \cdots c_{n-1} c$ *with well-formed categories* $c_0, \dots, c_{n-1}$ *such that*

- *it relabels to* $\rho(c_0) \cdots \rho(c_{n-1}) \rho(c) = \tau(q_0) \cdots \tau(q_n)$ *and*
- *category* c_i *has prefix* c, *ends with an argument* $|a$ *with* $a_3 = e_i$, *and* $|\beta_i| = \mathrm{arity}(c_i) - \mathrm{arity}(c)$ *for all* $i \in \{0, \dots, n-1\}$.

Proof. We will describe how the primary spine $c_0 \cdots c_{n-1} c$ is constructed by induction on $i \in \{0, \dots, n\}$ that additionally obeys the following invariants:

(i) The categories are well-formed.
(ii) Subsequent arguments $|_{j-1} a_{j-1} |_j a_j$ in the suffix (i.e., the argument sequence after c in each category $c|_1 a_1 \cdots |_m a_m$) are coupled, so $\mathrm{pop}((a_j)_2) = (a_{j-1})_1$ for all $j \in \{2, \dots, m\}$.

We already noted that the suffix of a category stores the stack in the second components, such that the last argument contains the topmost stack symbol in its second component. Also note that all states $q_0, \dots, q_n$ have the same spine generator.

In the induction base, we let $c_0 = c|(q_0, \beta_0, e_0)$ with $| = \mathrm{slash}(q_0)$. We note that $\mathrm{pop}(\beta_0) = \mathrm{target}(c)_3 = q_n$. Obviously, $c_0 \in L(\Delta_0)$ and

$\rho(c_0) = \tau(q_0)$. It is also clear that c_0 obeys the invariants and has the correct arity.

In the induction step, we assume that c_{i-1} already fulfills the conditions, and we let $(q_{i-1}, \gamma, \gamma', q_i) \in \delta$ be a transition that permits the move $\langle q_{i-1}, \beta_{i-1}\rangle \vdash_{\mathcal{A}} \langle q_i, \beta_i\rangle$. We again distinguish three cases for the construction of a suitable category c_i:

1. *Ignore stack:* Suppose that $\gamma = \gamma' = \varepsilon$. We start from a category $c_{i-1} = c|_1 a_1 \cdots |_m a_m$ and apply a rule of $R_1^{|_m}$ to obtain $c_i = c|_1 a_1 \cdots |_{m-1} a_{m-1}|b$, where $| = \text{slash}(q_i)$ and $b = \big(q_i, (a_m)_2, e_i\big)$. Since $b_1 = q_i$, as a primary category, c_i gets relabeled to $\tau(q_i)$. It is clearly well-formed. The stack symbol is not changed, so subsequent arguments are still coupled. Additionally, neither the stack size nor the arity of the category have changed.
2. *Push symbol:* Suppose that $\gamma' \neq \varepsilon$ and $c_{i-1} = c|_1 a_1 \cdots |_m a_m$. Now let $j \in \{i+1, \dots, n\}$ be minimal such that $\beta_j = \beta_{i-1}$ (i.e., j is the index of the configuration in which the stack symbol γ' is removed again). Clearly, we apply a rule of $R_2^{|_m}$ to obtain $c_i = c|_1 a_1 \cdots |_{m-1} a_{m-1}|b|'b'$ such that $| = \text{slash}(q_j)$, $|' = \text{slash}(q_i)$, $b = \big(q_j, (a_m)_2, e_j\big)$, and $b' = (q_i, \gamma', e_i)$. Note that $q_j = \text{pop}(\gamma')$. Hence c_i gets relabeled to $\tau(q_i)$. The mentioned conditions ensure that c_i is well-formed and obeys the second invariant. The increase in stack size is properly accounted for by an increased arity of c_i.
3. *Pop symbol:* Suppose that $\gamma \neq \varepsilon$. We further distinguish the cases $i < n$ and $i = n$. We start with $i < n$. Suppose that $c_{i-1} = c|_1 a_1 \dots |_m a_m$. Note that $m \geq 2$. Since c_{i-1} obeys the invariants, subsequent arguments in the suffix are coupled, so we have $(a_{m-1})_1 = \text{pop}\big((a_m)_2\big) = \text{pop}(\gamma) = q_i$. Moreover, $(a_{m-1})_3 = e_i$ as prepared in the corresponding push transition. We apply a rule of $R_3^{|_m}$ to obtain $c_i = c|_1 a_1 \dots |_{m-1} a_{m-1}$, which is trivially well-formed and still obeys the second invariant. It relabels to $\tau(q_i)$ as required due to $(a_{m-1})_1 = q_i$. The stack size and arity both decrease by 1.
   For $i = n$ we have $c_{n-1} = c|_1 a_1$ since the stack size is necessarily 1. We also apply a rule of $R_3^{|_1}$ and obtain the category c. This category is trivially well-formed and also trivially fulfills the second invariant. Additionally, it relabels to $\tau(q_n)$ since it is a non-primary category and $\text{target}(c)_3 = q_n$.

□

We observe that the secondary categories that are needed to perform the category transformations corresponding to the automaton run are well-formed as well. Thus, they can be chosen as the category at the end of an appropriate primary spine (unless the third

component of the target constitutes an unreachable state of $\mathcal{A}$). This will be relevant in the next step, in which we combine the spines to obtain a complete derivation tree.

5.6.2 Combining Spines

We continue to use the introduced symbols. Moreover, we write $\mathcal{D}'(\mathcal{G}_{\mathcal{A},\mathcal{L}_1}) = \{t \in \mathcal{D}(\mathcal{G}_{\mathcal{A},\mathcal{L}_1}) \mid t(\varepsilon) \in \top_{\mathcal{L}_1} \cup \top_{\mathcal{A}}\}$ to refer to the derivation trees whose root nodes are labeled by end-of-spine categories. We will show that $\rho\big(\mathcal{D}'(\mathcal{G}_{\mathcal{A},\mathcal{L}_1})\big) = \mathcal{F}\big(\text{Next}(\mathcal{S}(\mathcal{G}))\big)$. In other words, we will show that when these derivation trees are relabeled using ρ, this yields exactly the trees obtained by reassembling the annotated spines of the normalized spine grammar $\mathcal{G}$. We prove the inclusion in both directions.

Lemma 5.6.8 $\rho\big(\mathcal{D}'(\mathcal{G}_{\mathcal{A},\mathcal{L}_1})\big) \subseteq \mathcal{F}\big(\text{Next}(\mathcal{S}(\mathcal{G}))\big)$

Proof. We prove the statement by induction on the size of the derivation tree $t \in \mathcal{D}'(\mathcal{G}_{\mathcal{A},\mathcal{L}_1})$. Let $c_0 \cdots c_n$ be the primary spine of t that starts at a lexicon entry $c_0 \in L(\Delta_0)$ and ends at the root (i.e., $c_n = t(\varepsilon)$). By Lemma 5.6.5, this spine gets relabeled to a string $w = w_0 \cdots w_n \in \text{Next}\big(\mathcal{S}(\mathcal{G})\big)$. If $n = 0$, this finishes the induction base. Otherwise, except for the root category c_n, each of the spinal categories $c_0, \ldots, c_{n-1}$ gets combined with a (well-formed) secondary category that itself is the root of a subtree $t' \in \mathcal{D}'(\mathcal{G}_{\mathcal{A},\mathcal{L}_1})$. Since t' is a proper subtree of t, we can utilize the induction hypothesis to conclude that $\rho(t') \in \mathcal{F}\big(\text{Next}(\mathcal{S}(\mathcal{G}))\big)$. It remains to show that each such tree fulfills the requirements necessary to attach it to the spine. Suppose that the primary category is $c_i = ax|b$, so it can only be combined with a secondary category of the form $b\gamma$, where $\gamma \in \mathcal{A}(A)$ is some argument context. This secondary category gets relabeled to $\rho(b\gamma) = \tau(b_3)$. Suppose further that $\rho(c_i) = \big(\langle \sigma', n'_1, n'_2 \rangle, \langle \sigma, n_1, n_2 \rangle\big)$. Clearly, $\text{gen}(b_3) = \text{comb}(b_1)$, where $\text{gen}(b_3)$ is the spine generator at the root of $\rho(t')$, and $\text{comb}(b_1) = n'_{3-d(\sigma')}$ is the generator of the non-spinal child of the succeeding parent symbol $\rho(c_{i+1})$. Since they coincide, the attachment of $\rho(t')$ is possible and the directionality of the attachment is $3 - d(\sigma')$, which is guaranteed by the requirement that $| = \text{slash}(b)$ for argument $|b$. We conclude that all attachments of subtrees are consistent with the definition of $\mathcal{F}\big(\text{Next}(\mathcal{S}(\mathcal{G}))\big)$. Thus, $\rho(t) \in \mathcal{F}\big(\text{Next}(\mathcal{S}(\mathcal{G}))\big)$. □

Lemma 5.6.9 $\mathcal{F}\big(\text{Next}(\mathcal{S}(\mathcal{G}))\big) \subseteq \rho\big(\mathcal{D}'(\mathcal{G}_{\mathcal{A},\mathcal{L}_1})\big)$

Proof. Since we have to distinguish between short and longer spines, we indeed prove the following statement for all $t \in \mathcal{F}\big(\mathrm{Next}(\mathcal{S}(\mathcal{G}))\big)$. If t consists of a single node, then for each $c \in \{ax \in \top_{\mathcal{L}_1} \mid a_3 = t\}$ there is a tree $t' \in \mathcal{D}'(\mathcal{G}_{\mathcal{A},\mathcal{L}_1})$ such that $t'(\varepsilon) = c$ and $\rho(t') = t$. If t has more than one node, then for each $c \in \{ax \in \top_{\mathcal{A}} \mid a_3 = q_n\}$, where q_n is the final state of an accepting run of $\mathcal{A}$ corresponding to the main spine of t, there is a tree $t' \in \mathcal{D}'(\mathcal{G}_{\mathcal{A},\mathcal{L}_1})$ such that $t'(\varepsilon) = c$ and $\rho(t') = t$. We perform an induction on the size of t.

In the induction base, t consists of a single node $t \in \mathcal{L}_1$. By Lemma 5.6.6, all $c \in \big\{ax \in L(\Delta_0) \mid a_3 = t\big\} = \big\{ax \in \top_{\mathcal{L}_1} \mid a_3 = t\big\}$ are complete primary spines that get relabeled to t. By the definition of the lexicon, this set is nonempty for all $t \in \mathcal{L}_1$. Thus, we have $t \in \rho\big(\mathcal{D}'(\mathcal{G}_{\mathcal{A},\mathcal{L}_1})\big)$.

In the induction step, let $t \in \mathcal{F}\big(\mathrm{Next}(\mathcal{S}(\mathcal{G}))\big)$ be a tree with $|t| > 1$. We identify the main spine labeled by $w = w_0 \cdots w_n \in \mathrm{Next}\big(\mathcal{S}(\mathcal{G})\big)$ that was used to create t. This string w is generated by an accepting run $\big(\langle q_0, \beta_0\rangle, \dots, \langle q_n, \varepsilon\rangle\big)$ of $\mathcal{A}$. Likewise, there exists a primary spine $c_0 \cdots c_n$ of $\mathcal{G}_{\mathcal{A},\mathcal{L}_1}$ that gets relabeled to w and we can choose the category c_n at the end of the spine freely from the set given by $\big\{ax \mid ax|b \in L(\Delta_0), a_3 = q_n\big\} = \big\{ax \in \top_{\mathcal{A}} \mid a_3 = q_n\big\}$ according to Lemma 5.6.7. Similarly, we can freely choose the third component of the last argument of each category $c_0, \dots, c_{n-1}$ under the condition that this yields a valid atom. Consider an arbitrary $0 \leq i \leq n-1$, and let $w_i = \big(\langle \sigma', n'_1, n'_2\rangle, \langle \sigma, n_1, n_2\rangle\big)$; the case of $w_i = \big(\langle \sigma', n'_1, n'_2\rangle, \langle \sigma, n\rangle\big)$ is analogous. At the parent node with label w_{i+1}, in the direction $3 - d(\sigma')$, a subtree $t' \in \mathcal{F}\big(\mathrm{Next}(\mathcal{S}(\mathcal{G}))\big)$ with spine generator $n'_{3-d(\sigma')}$ stored in its root label is attached. Let q be the final state of an accepting run of $\mathcal{A}$ for the main spine of t' when $|t'| > 1$ or let $q = t'$ when $|t'| = 1$. Moreover, suppose that $c_i = ax|b$ with $b_1 = q_i$, so the required secondary category has the shape $b\gamma$ with $\gamma \in \mathcal{A}(A)$. By the induction hypothesis, there exists $t'' \in \mathcal{D}'(\mathcal{G}_{\mathcal{A},\mathcal{L}_1})$ such that $t''(\varepsilon) = b\gamma \in \top_{\mathcal{L}_1} \cup \top_{\mathcal{A}}$, $\rho(t'') = t'$, and $b_3 = q$. This choice of b_3 is permitted for c_i since $\mathrm{comb}(b_1) = \mathrm{comb}(q_i) = n'_{3-d(\sigma')} = \mathrm{gen}(q) = \mathrm{gen}(b_3)$, which confirms that $\mathrm{gen}(b_3) = \mathrm{comb}(b_1)$. The directionality $3 - d(\sigma')$ of the attachment of t'' is guaranteed by the relationship $| = \mathrm{slash}(b)$, which holds since c_i is well-formed. In conclusion, for each attached subtree $t' \in \mathcal{F}\big(\mathrm{Next}(\mathcal{S}(\mathcal{G}))\big)$ of t we can find a suitable $t'' \in \mathcal{D}'(\mathcal{G}_{\mathcal{A},\mathcal{L}_1})$ whose root category can be combined with the neighboring primary category of the spine $c_0 \cdots c_n$. Putting the primary spine and the derivation trees for subtrees together again yields a tree in $\mathcal{D}'(\mathcal{G}_{\mathcal{A},\mathcal{L}_1})$. Its root c_n can be chosen freely from the desired set $\big\{ax \mid ax|b \in L(\Delta_0),\ a_3 = q_n\big\}$. ☐

Now we have to restrict both sets of trees in the following manner. For the reassembled spines, only those trees whose main spine is generated by the start nonterminal s may be considered, since only these are part of $\mathcal{T}(\mathcal{G})$ (see Theorem 5.4.7). Regarding the derivation trees of $\mathcal{G}_{\mathcal{A},\mathcal{L}_1}$, we are interested only in those rooted in an initial category, as these contribute to the generated tree language. We will show that these restricted sets coincide as well. For the first direction, in order to gain the necessary freedom regarding the root category of the derivation tree, we will actually apply the stronger statement that we established in the proof of Lemma 5.6.9.

Lemma 5.6.10 $\mathcal{F}\big(\text{Next}(\mathcal{S}(\mathcal{G}))\big)_s = \mathcal{T}_\rho(\mathcal{G}_{\mathcal{A},\mathcal{L}_1})$

Proof. Recall that $\mathcal{G}_{\mathcal{A},\mathcal{L}_1}$ together with relabeling ρ generates the tree language $\mathcal{T}_\rho(\mathcal{G}_{\mathcal{A},\mathcal{L}_1}) = \big\{\rho(t) \in T_{\Delta,\emptyset}(\Delta) \mid t \in \mathcal{D}(\mathcal{G}_{\mathcal{A},\mathcal{L}_1}),\ t(\varepsilon) \in I'\big\}$. It follows that $\mathcal{T}_\rho(\mathcal{G}_{\mathcal{A},\mathcal{L}_1}) \subseteq \rho\big(\mathcal{D}'(\mathcal{G}_{\mathcal{A},\mathcal{L}_1})\big)$. Also recall that the initial atomic categories of $\mathcal{G}_{\mathcal{A},\mathcal{L}_1}$ are $I' = \big\{(\bot, \varepsilon, f) \in A \mid \text{gen}(f) = s\big\}$.

First, let $t \in \mathcal{F}\big(\text{Next}(\mathcal{S}(\mathcal{G}))\big)_s$. By Lemma 5.6.9, there exists a tree $t' \in \mathcal{D}(\mathcal{G}_{\mathcal{A},\mathcal{L}_1})$ with $\rho(t') = t$, whose root category can be any category from either $\{ax \in \top_{\mathcal{A}} \mid a_3 = f\}$, where f is the final state of an accepting run for the main spine of t, or from $\{ax \in \top_{\mathcal{L}_1} \mid a_3 = t\}$, if t consists of a single node. Hence we can select the category $t'(\varepsilon) = (\bot, \varepsilon, f)$ in the former and $t'(\varepsilon) = (\bot, \varepsilon, t)$ in the latter case. Since both of these categories are initial, we obtain that $\rho(t') \in \mathcal{T}_\rho(\mathcal{G}_{\mathcal{A},\mathcal{L}_1})$.

Now let $t \in \mathcal{T}_\rho(\mathcal{G}_{\mathcal{A},\mathcal{L}_1})$. Hence there is a tree $t' \in \mathcal{D}(\mathcal{G}_{\mathcal{A},\mathcal{L}_1})$ such that $\rho(t') = t$ and $t'(\varepsilon) \in I' = \big\{(\bot, \gamma, f) \in A \mid \text{gen}(f) = s\big\}$. By Lemma 5.6.8, we also have $t \in \mathcal{F}\big(\text{Next}(\mathcal{S}(\mathcal{G}))\big)$. Because after relabeling, the root is labeled by $\rho\big(t'(\varepsilon)\big) = t(\varepsilon) = f$ with $\text{gen}(f) = s$, we obtain $t \in \mathcal{F}\big(\text{Next}(\mathcal{S}(\mathcal{G}))\big)_s$. $\square$

Together with Corollary 5.5.6, this concludes the proof of the following main theorem.

Theorem 5.6.11 *Given a spine grammar $\mathcal{G}$, we can construct a CCG that can generate $\mathcal{T}(\mathcal{G})$.*

5.7 Strong Equivalence

In this section we will show that CCG and sCFTG as well as CCG and TAG are strongly equivalent.[9] We will also cover the implications regarding the role of ε-entries, rule degree, and the use of first-order categories.

9: Since the tree languages generated by CCG are already defined via a relabeling of categories, we do not require the qualification "modulo relabeling" here.

To show strong equivalence of sCFTG and CCG, we combine Theorems 5.1.13 and 5.6.11, which we have proven in this chapter. This leads us to the following main theorem.

Theorem 5.7.1 *CCG and sCFTG are strongly equivalent.*

Proof. Given a CCG $\mathcal{G}$, by Theorem 5.1.13, its rule tree language $\mathcal{R}(\mathcal{G})$ can be generated by an sCFTG $\mathcal{G}'$. The tree language $\mathcal{T}_\rho(\mathcal{G})$ accepted by $\mathcal{G}$ is the relabeled set of derivation trees $\mathcal{D}(\mathcal{G})$ rooted in an initial category. The relabeling ρ can be transferred to the rule tree language $\mathcal{R}(\mathcal{G})$ since it only depends on the target and the last argument of each category, which can both be figured out by looking at the output category of the rule label of the respective rule tree node. Conversely, given an sCFTG $\mathcal{G}$, we can first convert it into an equivalent spine grammar (up to deterministic relabeling) and then construct a CCG that is equivalent to $\mathcal{G}$ by Theorem 5.6.11. □

Kepser and Rogers [45] proved that TAG and sCFTG are strongly equivalent, which shows that TAG is also strongly equivalent to CCG. We briefly sketch the transformation from TAG to sCFTG that they presented. TAG is a notational variant of footed simple CFTG. This is a simple CFTG where all variables in right-hand sides of productions appear in order directly below a designated *foot node*. To obtain an sCFTG, the footed simple CFTG is first converted into a spine grammar, where the spine is the path from the root to the foot node, and then brought into normal form using the construction of Fujiyoshi and Kasai [21]. The spine grammar of Example 5.3.3 is strongly equivalent to the TAG shown in Figure 2.2.

Corollary 5.7.2 *CCG and TAG are strongly equivalent.*

Clearly, from strong equivalence we can conclude weak equivalence as well. Weak equivalence of CCG and TAG was famously proven by Vijay-Shanker and Weir [93], but Theorem 3 of Kuhlmann, Koller, and Satta [50] highlights a problem with the original construction.[10] However, Weir [95] provides an alternative construction that does not suffer from that issue. Our contribution provides a stronger form (and proof) of this old equivalence result. It avoids the ε-entries that the original construction heavily relies on. An ε-entry

10: Kuhlmann, Koller, and Satta [50] show that prefix-closed CCG without target restrictions is less expressive than TAG. But the CCG constructed by Vijay-Shanker and Weir [93] to simulate an arbitrary TAG has exactly these properties.

is a category assigned to the empty string (see page 30); these interspersed categories form the main building block in the original constructions. The necessity of these ε-entries is an interesting and important question that naturally arises and has been asked by Kuhlmann, Koller, and Satta [50]. We settle this question and demonstrate that they can be avoided.

Corollary 5.7.3 *CCG and TAG are weakly equivalent. Moreover, CCG with ε-entries and CCG without ε-entries generate the same (ε-free) languages.*

Proof. The weak equivalence of CCG and TAG is clear from the previous corollary. Similarly, each ε-free language generated by a CCG can trivially also be generated by a CCG with ε-entries. For the converse direction, let $\mathcal{G}$ be a CCG with ε-entries. We convert it into an sCFTG accepting the rule tree language of $\mathcal{G}$ by applying Definition 5.1.2. This sCFTG is already in the spine grammar normal form of Fujiyoshi and Kasai [21, Definition 4.2]. Therefore, we can apply the ε-removal procedure of Fujiyoshi [20, Theorem 4.1] to obtain a weakly equivalent spine grammar. In our sCFTG, the productions that require our attention are those of the form $\langle c \rangle \to c$ with $c \in L('\varepsilon')$. To establish the necessary preconditions to properly perform the ε-removal, we add productions of the form $\langle c \rangle \to '\varepsilon'$ for each $c \in L('\varepsilon')$. For each $c \in L(\Sigma)$, unless $c \in L(\alpha)$ for some $\alpha \in \Sigma$, we remove the rule $\langle c \rangle \to c$ from the set of productions. We call the grammar modified in this way $\mathcal{G}' = \big(N_1 \cup N_0, R \cup L(\Sigma), S, P\big)$.

In the following, we present an ε-removal procedure that essentially follows Fujiyoshi [20]. Since that construction introduces a new unary terminal symbol, which we wish to avoid, we present an adjusted version. First, two sets E_0 and E_1 are constructed, which contain the nullary and unary nonterminals that can derive a tree with yield ε or a context with yield $(\varepsilon, \varepsilon)$, respectively. They are constructed iteratively starting from $E_0 = \{n \in N_0 \mid n \to '\varepsilon' \in P\}$ and $E_1 = \emptyset$. The following steps are repeated until a fixpoint is reached:

- Add n to E_0 if there is $n \to b(a) \in P$ with $b \in E_1$ and $a \in E_0$.
- Add n to E_1 if there is $n \to \sigma(\square, a) \in P$ or $n \to \sigma(a, \square) \in P$ with $a \in E_0$.
- Add n to E_1 if there is $n \to b_1(b_2(\square)) \in P$ with $b_1, b_2 \in E_1$.

After finishing the setup of E_0 and E_1, the weakly equivalent spine grammar is defined as $\mathcal{G}'' = \big(N_0' \cup N_1, R \cup L(\Sigma), S, P'\big)$

with $N_0' = N_0 \cup \{\overline{n} \mid n \in N_1\}$ and the set of productions

$$
\begin{aligned}
P' = \big(P \setminus \{n \to {}'\varepsilon' \mid n \in N_0\}\big) \cup \\
&\big\{n \to \overline{b} \mid n \to b(a) \in P \text{ with } a \in E_0\big\} \cup \\
&\big\{\overline{n} \to b_1(\overline{b_2}) \mid n \to b_1(b_2(\square)) \in P\big\} \cup \\
&\big\{\overline{n} \to \overline{b_1} \mid n \to b_1(b_2(\square)) \in P \text{ with } b_2 \in E_1\big\} \cup \\
&\big\{n \to \square \mid n \to \sigma(\square, a) \in P \text{ or } n \to \sigma(a, \square) \in P \text{ with } a \in E_0\big\} \cup \\
&\big\{\overline{n} \to a \mid n \to \sigma(\square, a) \in P \text{ or } n \to \sigma(a, \square) \in P\big\} \enspace .
\end{aligned}
$$

For a detailed correctness proof, we refer to Fujiyoshi [20].

Then, $\mathcal{G}''$ can be converted into a strongly equivalent CCG by Theorem 5.6.11. This CCG accepts the same ε-free string language as the original CCG $\mathcal{G}$ that used ε-entries. □

The tree expressive power of CCG with restricted rule degrees has already been investigated in Chapter 4. We showed that 0-CCG accepts a proper subset of the regular tree languages, whereas 1-CCG accepts exactly the regular tree languages. It remained open whether there is a k such that k-CCG and $(k+1)$-CCG has the same expressive power. Our construction in Definition 5.6.1 establishes that 2-CCG is as expressive as k-CCG for arbitrary $k \geq 2$. The construction also shows that first-order categories are sufficient.

Corollary 5.7.4 *2-CCG with first-order categories has the same expressive power as k-CCG with $k > 2$.*

Proof. We only argue the nontrivial inclusion. Let $\mathcal{G}$ be a CCG whose categories have arbitrary order and whose rules have degree at most k. Using Definition 5.1.2, we construct the sCFTG $\mathcal{G}'$ generating the rule tree language $\mathcal{R}(\mathcal{G})$. After performing the intermediate steps as discussed in the preceding sections, using Definition 5.6.1, we construct the CCG $\mathcal{G}''$ that generates the same tree language as $\mathcal{G}'$. By construction, $\mathcal{G}''$ uses only first-order categories and rules with degree at most 2. As already argued, the rule trees can be relabeled to the tree language generated by $\mathcal{G}$. Since the composition of two (deterministic) relabelings again is a (deterministic) relabeling, the tree language generated by $\mathcal{G}$ is generatable by $\mathcal{G}''$ as well. □

Computational Complexity for Bounded Rule Degree 6

6.1 Parsing Algorithm . 98
6.2 Correctness 108
6.3 Runtime Analysis . . 115
6.4 From Parse Tree to Derivation Tree . . . 121
6.5 Parser Extensions and Improvements 125

In this chapter, we study a certain aspect of the computational complexity of CCG. In particular, we are interested in the role of the maximum rule degree. This research is joint work with Marco Kuhlmann and Giorgio Satta (see Section 1.5). The main result of this chapter is that when the rule degree of a CCG is bounded by a constant, i.e., if we fix the maximum rule degree to some $k \in \mathbb{N}$, the universal recognition problem can be solved in polynomial time. This means that parsing can be performed in time polynomial not only in the length of the input string, but also in the size of the grammar. We show this by designing a parsing algorithm that has a runtime exponential solely in the maximum rule degree k.

Throughout this chapter, we assume that substitution rules and ε-entries are allowed in general. To simplify the presentation, we also assume that the CCG we are given is pure. However, our algorithm can easily be extended to CCG that uses only a subset of combinatory rules up to some degree and specifically allows rule restrictions. This will be discussed in Section 6.5.2. These properties therefore do not limit the validity of the main result. Thus, in this chapter, if not stated otherwise, let $\mathcal{G} = (\Sigma, A, R, I, L)$ be a CCG with these properties. As in Section 5.1, we will generally restrict ourselves to lexical arguments $\text{args}(L)$ and argument contexts $\mathcal{A}_L(A)$ consisting thereof, which is permitted by Proposition 3.3.10.

A full complexity analysis of practical CCG would cover not only application, composition, and substitution, but also *type-raising* (see Section 3.2.5). However, here we follow Steedman's assumption that this rule cannot be used recursively and is implemented in the lexicon [85].[1] Similarly, we assume a purely lexical treatment of *coordination* via (appropriately restricted) lexicon entries such as $X \backslash X / X \in L(\text{'and'})$ [87, page 91].

1: Note that there are different approaches for the implementation of type-raising under discussion (see page 142).

The remainder of this chapter is structured as follows. In Section 6.1, after the introduction of some additional notation and definitions, the new parsing algorithm is presented. Section 6.2 contains the correctness proof of the algorithm, divided into the proofs of soundness and completeness. Section 6.3 provides a comprehensive runtime analysis. There, we also show that if ε-entries are allowed, the universal recognition problem for CCG of bounded rule degree is PTIME-complete under logspace-reduction. Section 6.4 describes how to construct CCG derivation trees from the parse trees produced by our algorithm. Finally, Section 6.5 explores several potential extensions and improvements of the algorithm,

including the elimination of spurious ambiguity, support for rule restrictions, support for multi-modal CCG, and an algorithm for the universal recognition problem running in polynomial time if all secondary categories in the rule set of the grammar are instantiated, i.e., if they do not contain any variables.

6.1 Parsing Algorithm

In this section we develop a tabular algorithm for parsing based on CCG. Our algorithm extends the approach of Kuhlmann and Satta [54] 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. Before moving on with the technical presentation, we informally discuss 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 already been pointed out in the literature, for instance by Kuhlmann and Satta [54, §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 h of the topmost arguments in its categories, where h 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 contexts of length at most h. 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 [54].

As a second technical point, consider an instance of a composition rule having the form $\frac{c/b \quad b\alpha}{c\alpha}$ or $\frac{b\alpha \quad c\backslash b}{c\alpha}$, where b, c are categories and α is an argument context. In accordance with the factorization of CCG derivations that we have mentioned above, the number of arguments in α is bounded by our constant h. However, the arity of category b is bounded by the maximum arity m of an argument from the lexicon, which is independent of h. In pathological cases, where m is much larger than h, 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 [54], which exhaustively produces all possible categories $b\alpha$ with arity bounded by $m + h$. As a solution for this we observe here that the category b must match a finite number of possible arguments from the lexicon of the CCG. Using this restriction, we avoid the additional exponential factor of m in the running time of our parsing algorithm.

6.1.1 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 $\mathcal{G}$ and a string $w = w_1 \cdots w_n$ to be parsed. We write $w[i, j] = w_{i+1} \cdots w_j$ to denote the substring of w from (fencepost) position i to position j, for $0 \leq i \leq j \leq |w|$, and assume $w[i, i] = \varepsilon$. Given some string w, we write w_i for $w[i-1, i]$, the one-element substring containing the i-th symbol.

Extended Derivation Tree We extend CCG derivation trees by adding additional nodes labeled by input symbols as children of leaf categories. The input symbol labeling such a node is associated with the lexical category of its respective parent node via the lexicon. We add this information because our parsing algorithm can determine it and due to its importance for the notion of spurious ambiguity that is discussed in Section 6.5.1. Throughout this chapter, when talking about derivation trees, we refer to these extended derivation trees.

Definition 6.1.1 *An* extended derivation tree *of CCG $\mathcal{G}$ is a tree $t \in T_{\mathcal{C}(A),L(\Sigma^{\leq 1})}(\Sigma^{\leq 1})$ such that there exists a tree $t' \in \mathcal{D}(\mathcal{G})$ where* $\mathrm{pos}(t) = \mathrm{pos}(t') \cup \{u1 \mid u \in \mathrm{leaves}(t')\}$ *and for $u \in \mathrm{pos}(t')$ we have $t(u) = t'(u)$ and for $u \in \mathrm{leaves}(t')$ we have $t'(u) \in L(t(u1))$.*

Spine As in the previous chapters, the notion of spine is essential here as well. The *spine* of a derivation tree is the path that starts at the root node and at each node continues to the child labeled by the primary category until it reaches some unary node. This unary node is called *lexical anchor*. The spine therefore corresponds to the term “main spine” of the previous chapter, and excludes the additional leaf nodes, which are labeled by an input symbol or by 'ε'. Thus, each *spinal node* is labeled by a CCG category. Accordingly, the *length* of a spine is defined as its number of spinal nodes. To address a specific spine segment whose nodes are located properly

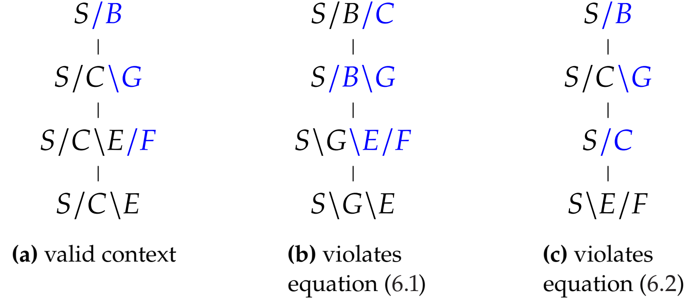


Figure 6.1: Examples of spines complying with or violating the context definition. The bridging arguments of primary categories are highlighted to emphasize the downstep arities. As usual, the foot node is drawn at the top and the root node at the bottom.

between two spinal nodes u and v with $|u| < |v|$ (i.e., u is closer to the root), we write $\text{between}(u, v) = \text{Pref}(v) \setminus (\text{Pref}(u) \cup \{v\})$.

Downstep Arity We consider a combinatory rule application as consisting of two phases, where in the first phase the bridging arguments of the primary category are removed, and in the second phase the excess is added. Consider some derivation tree t with node u labeled by primary category $t(u) = c|b\alpha$, the sibling labeled by $b\alpha\beta$, and the parent v by $t(v) = c\alpha\beta$. The *downstep arity* of u concerns the category that is obtained from $c|b\alpha$ by removing the bridging arguments, so $\text{downarity}(t, u) = \text{arity}(c)$. The category c is considered as the result of the first phase of the rule application. If u is labeled by a non-primary category, its downstep arity is undefined. Also note that it cannot be determined based on an individual category, but only within a given tree. The result of the second phase of the rule application is the output category, and its arity is the usual $\text{arity}(t(v)) = \text{arity}(c\alpha\beta)$. We informally say that a node has a certain arity when referring to its labeling category.

Example 6.1.2 Let t be a derivation tree with some node u labeled by $t(u) = S/C\backslash E$ and its children labeled by $t(u1) = S/B/C$ and $t(u2) = B/C\backslash E$. The category labeling u is clearly obtained by using an instance of the substitution rule $\frac{Sx/B/C \quad B/C\backslash E}{Sx/C\backslash E}$ with $x = \square$. Then we have $\text{arity}(t(u)) = \text{arity}(t(u1)) = 2$ and $\text{downarity}(t, u1) = 0$.

Derivation Context

In accordance with the classical definition of context (see Section 2.3), a derivation context is a derivation tree with a hole. However, there are three distinctive features. First, the hole is located on the spine of the underlying derivation tree. Second, the position of the special character $\square$ is associated with a category. Third, there are two conditions regarding the arities and downstep arities along the spine of the context. Throughout this chapter, the term context refers to derivation contexts.

Definition 6.1.3 *Let* (C, c) *be a tuple with* $C \in C_{\mathcal{C}(A), L(\Sigma^{\leq 1})}(\Sigma^{\leq 1})$ *and* $c \in \mathcal{C}(A)$. *Let the node* $f = \mathrm{pos}_{\square}(C)$ *be called* foot node. *For convenience, we access the category* c *by writing* $C(f)$ *and also treat* c *as the label of the foot node otherwise.*

We call (C, c) *a* derivation context of $\mathcal{G}$ *if*

- $\frac{C(u1) \quad C(u2)}{C(u)} \Pi$ *for every* $u \in \mathrm{pos}(C)$ *with* $u1, u2 \in \mathrm{pos}(C)$,
- *all nodes in* $\mathrm{between}(\varepsilon, f)$ *are labeled by primary categories,*
- *for all nodes* $u \in \mathrm{between}(\varepsilon, f)$ *the following conditions hold:*

$$\mathrm{downarity}(C, f) \leq \mathrm{downarity}(C, u) \tag{6.1}$$

$$\mathrm{arity}(C(\varepsilon)) \leq \mathrm{arity}(C(u)) \tag{6.2}$$

The spine *of the derivation context contains all nodes in* $\mathrm{Pref}(f)$. *We usually write* (C, c) *simply as* C.

Example 6.1.4 Figure 6.1 shows some examples of spines in order to illustrate the context definition. For the sake of simplicity, the respective secondary categories are omitted, although in general they would be necessary to determine downstep arities. We highlighted the bridging arguments to emphasize the downstep arities. According to our convention, the foot node is drawn at the top and the root node at the bottom. Figure 6.1a depicts a valid context. In Figure 6.1b, the node labeled by $S/B\backslash G$ has downstep arity 0 due to substitution, but the downstep arity at the foot is 1. Therefore, the spine violates condition (6.1) of the context definition. Finally, in Figure 6.1c the node labeled by S/C has lower arity than root category $S\backslash E/F$. Therefore, the spine violates condition (6.2) of the context definition.

In the following, we have $\alpha, \gamma \in \mathcal{A}_L(A)$. Given a context C, let us write the category at f as $c\alpha$, where $\mathrm{arity}(c) = \mathrm{downarity}(C, f)$. This means that $1 \leq |\alpha| \leq 2$. Condition (6.1) implies that the category at each spinal node can be written as $c\gamma$. Note that only the arguments in γ may affect the derivation along the spine of the context. In other words, the arguments in c are not needed and we can represent the context without any record of c itself. We exploit 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. We can in fact replace the prefix c along the spine by some other category c' to obtain another valid context with the same yield. Therefore, we may think of the prefix c as a placeholder or variable that any category can be substituted for.

Lemma 6.1.5 *Let C be a derivation context with foot node f such that $C(f) = c\alpha$, where $\mathrm{arity}(c) = \mathrm{downarity}(C, f)$, and let $c' \in \mathcal{C}(A)$. Let us define C' with $\mathrm{pos}(C') = \mathrm{pos}(C)$ as $C'(u) = C(u)$ for $u \in \mathrm{pos}(C') \setminus \mathrm{Pref}(f)$, and $C'(u) = c'\gamma$ for $u \in \mathrm{Pref}(f)$ with $C(u) = c\gamma$. Then C' is a derivation context of $\mathcal{G}$ as well.*

Proof. It is easy to check that the conditions of Definition 6.1.3 still hold. First, each node in C' together with its children still constitutes a valid combinatory rule of $\mathcal{G}$. If $u, u1, u2 \in \mathrm{pos}(C') \setminus \mathrm{Pref}(f)$, then $\frac{C'(u1) \quad C'(u2)}{C'(u)}\Pi$ holds since $C'(w) = C(w)$ for $w \in \mathrm{pos}(C') \setminus \mathrm{Pref}(f)$. Otherwise, we have two spinal nodes $u, ud \in \mathrm{Pref}(f)$ with $d \in [2]$ and $u(3-d) \in \mathrm{pos}(C') \setminus \mathrm{Pref}(f)$. Assume these nodes in C are labeled by $C(ud) = c\delta|b\alpha$, $C(u) = c\delta\alpha\beta$, and $C(3-u) = b\alpha\beta$ with $\frac{C(u1) \quad C(u2)}{C(u)}\Pi$, where $\mathrm{arity}(c) = \mathrm{downarity}(C, f)$. Then the respective nodes in C' are labeled by $C'(ud) = c'\delta|b\alpha$, $C'(u) = c'\delta\alpha\beta$, and $C'(u(3-d)) = b\alpha\beta$. Since we assumed that $\mathcal{G}$ is pure, they can be combined by a combinatory rule in the same manner and we have $\frac{C'(u1) \quad C'(u2)}{C'(u)}\Pi$.[2] Second, from the above inspection it becomes clear that each node in $\mathrm{between}(\varepsilon, f)$ is still a primary category. Third, let $\ell = \mathrm{arity}(c') - \mathrm{arity}(c)$. Then each node $u \in \mathrm{Pref}(f)$ has $\mathrm{arity}(C'(u)) = \mathrm{arity}(C(u)) + \ell$ and each node $u \in \mathrm{between}(\varepsilon, f)$ has $\mathrm{downarity}(C', u) = \mathrm{downarity}(C, u) + \ell$. Conditions 6.1 and 6.2 are clearly fulfilled as each side of the inequalities is increased by ℓ in comparison with C. □

2: If $\mathcal{G}$ uses rule restrictions and c' has another target than c, a target restriction might prevent this combination. Therefore, in that case, we need to associate each context with the target along the spine and may only insert categories c' with that target (see Section 6.5.2).

Condition (6.2) is also very important for storing contexts in a compact form, and is at the basis of the proof of the following lemma, where f is defined as above.

Lemma 6.1.6 *Let C be a context with root label $C(\varepsilon) = c\beta$ such that $\mathrm{arity}(c) = \mathrm{downarity}(C, f)$. Assume that the combinatory rules applied along the spine of C have degree at most k. Then $|\beta| \leq k$.*

Proof. Let $C(f) = c\alpha$ and $C(p) = c\gamma$, where p is the parent node of f, so $f = pd$ with $d \in [2]$. Since $C(p)$ is obtained by applying a rule of degree at most k on primary category $c\alpha$, we have $|\gamma| \leq k$. If $p = \varepsilon$, we immediately have $\beta = \gamma$ and thus $|\beta| \leq k$. Otherwise, $p \in \mathrm{between}(\varepsilon, f)$, and by Condition (6.2) in the context definition, we have $\mathrm{arity}(C(\varepsilon)) \leq \mathrm{arity}(C(p))$. Observe that $\mathrm{arity}(C(\varepsilon)) = \mathrm{arity}(c) + |\beta|$ and $\mathrm{arity}(C(p)) = \mathrm{arity}(c) + |\gamma|$. We can then write $|\beta| \leq |\gamma| \leq k$. □

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 $|\alpha| = 2$.

Root Categories

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

- $\mathcal{H}_1$ contains all categories $c\alpha$ with $c \sqsubseteq c\beta$ for some lexical category $c\beta \in L(\Sigma^{\leq 1})$ such that $\alpha \in \mathcal{A}_L(A, 2)$, and $\text{arity}(c\alpha) \leq \text{arity}(c\beta)$.
- $\mathcal{H}_2$ contains all categories $c\alpha$ with $c \sqsubseteq c\beta$ for some *instantiation* $c\beta$ of a secondary category of a rule in R such that $\alpha \in \mathcal{A}_L(A, 2)$, and $\text{arity}(c\alpha) \leq \text{arity}(c\beta)$.

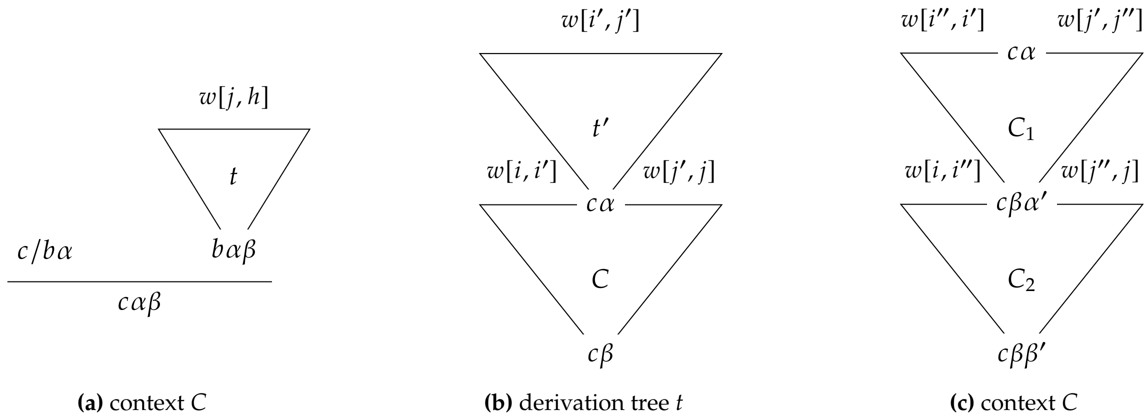
In other words, each category in $\mathcal{H}$ is based on either a lexical category or an instantiated secondary category, as it consists of a (not necessarily proper) prefix of such a category, and at most two additional lexical arguments, without exceeding the arity of the underlying category. The α component of $c\alpha$ represents bridging arguments. Categories in $\mathcal{H}_1$ are used by the parser when derivation trees are introduced from lexical categories, and when topmost arguments of these categories are "consumed" in a CCG derivation. Categories in $\mathcal{H}_2$ are instead used in the process of producing a category that will serve as a secondary category in a CCG derivation. 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.

6.1.2 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 [80].

Items

We use a logic with two types of items. Derivation trees whose root is labeled by a category in $\mathcal{H}$ 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.



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

Figure 6.2: Decomposition of derivations, where b and c are categories, and α, β, α', β' are (possibly empty) argument contexts satisfying the restrictions specified in the inference rules.

Tree Items These have the form $\langle c, i, j \rangle$, consisting of a category $c \in \mathcal{H}$ and two indices $0 \leq i \leq j \leq |w|$. The intended interpretation of such an item is: It is possible to build a derivation tree with yield $w[i, j]$ and root category c. The goal of the algorithm is the construction of an item of the form $\langle a_0, 0, |w| \rangle$ with $a_0 \in I$, which asserts the existence of a derivation tree that spans the entire input string and whose root node is labeled by an initial atomic category.

Context Items These have the form $\langle \alpha, \beta, i, i', j', j \rangle$, consisting of two argument contexts $\alpha, \beta \in \mathcal{A}_L(A, k)$ with $1 \leq |\alpha| \leq 2$ and four indices $0 \leq i \leq i' \leq j' \leq j \leq |w|$. The intended interpretation of these items is: For any choice of a category c, *if* it is possible to build a derivation tree t' with yield $w[i', j']$ and whose root node is labeled by $c\alpha$, *then* it is also possible to build a derivation tree t with yield $w[i, j]$ and whose root node is labeled by $c\beta$. In line with the usage of these terms for contexts, we refer to α as *bridging arguments* and to β as *excess*.

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 a context item, and depending on whether the consequent item is obtained as the extension of an existing item of the same type or 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 input position $1 \leq i \leq |w|$ and every lexicon entry $c \in L(w_i)$, there is an axiom $\langle c, i-1, i \rangle$. For every lexicon entry $c \in L('\varepsilon')$ and every fencepost position $0 \leq i \leq |w|$, there is an axiom $\langle c, i, i \rangle$.

$$\frac{c \in L(w_i)}{\langle c, i-1, i \rangle} \qquad \frac{c \in L('\varepsilon')}{\langle c, i, i \rangle} \qquad \text{(rule 0)}$$

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

$$\frac{\langle b\alpha\beta, j, h \rangle}{\langle /b\alpha, \alpha\beta, i, i, j, h \rangle} \quad \text{and} \quad \frac{\langle b\alpha\beta, j, h \rangle}{\langle \backslash b\alpha, \alpha\beta, j, h, \ell, \ell \rangle} \qquad \text{(rule 1)}$$

This rule type converts a tree item into a context item that models the effect of $b\alpha\beta$ when applied as a secondary category. The context has a spine of length 2 and is illustrated in Figure 6.2a.

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

$$\frac{\langle c\alpha, i', j' \rangle \langle \alpha, \beta, i, i', j', j \rangle}{\langle c\beta, i, j \rangle} \qquad \text{(rule 2)}$$

This rule type models the combination of a derivation tree and a context, such that the derivation tree rooted in $c\alpha$ is inserted at the foot node of the context, resulting in a derivation tree rooted in $c\beta$. This is depicted in Figure 6.2b.

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

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

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 6.2c.

The use of restriction $|\beta'| \leq |\alpha'|$ in rule 3 deserves some discussion here. This restriction guarantees that C fulfills condition (6.2) in the context definition. 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 6.2, 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 6.1.7 We consider a CCG $\mathcal{G} = (\Sigma, A, \mathcal{R}_s(A, 2), \{S\}, L)$, which allows all unrestricted composition and substitution rules of degree at most 2, and the input string $w_1 \cdots w_7$. The symbols in Σ and A can be inferred from the lexicon:

$$L(w_1) = \{B/E\} \quad L(w_2) = \{C/E/F\} \quad L(w_3) = \{S\backslash B/H\}$$
$$L(w_4) = \{H\backslash C/E\} \quad L(w_5) = \{F\} \quad L(w_6) = \{E/G\}$$
$$L(w_7) = \{G\}$$

Figure 6.3 depicts a derivation of the CCG, showing that the input is generated by the grammar. We can observe that on the spine between the root and the lexical category $S\backslash B/H$, there are two categories that are not in $\mathcal{H}$ because they violate the arity restriction. Thus, they cannot be represented using tree items.

Figure 6.4 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 category first has to be converted into a context item using deduction rule 1. For instance, $\langle H\backslash C/E, 3, 4\rangle$ is converted into $\langle /H, \backslash C/E, 2, 2, 3, 4\rangle$, which models the effect of the secondary category on a primary 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 $\langle S\backslash B/H, 2, 3\rangle$ and $\langle /H, \backslash C/E, 2, 2, 3, 4\rangle$ because the resulting category is not in $\mathcal{H}$. Instead, the excess of the context item is reduced by combining it with another context item using deduction rule 3 first, resulting in $\langle /H, /E, 1, 2, 3, 5\rangle$. This item models the effect that the categories of the three involved axioms have when applied successively to a category ending in $/H$. These are exactly the first three secondary categories that are applied along the considered spine. The item covers the spans $[1, 2]$ and $[3, 5]$ of the input with a gap at $[2, 3]$,

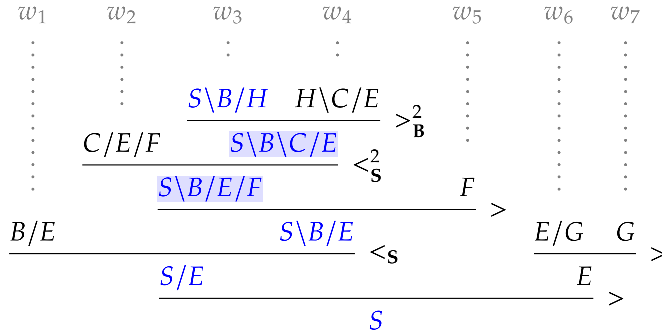


Figure 6.3: CCG derivation with spine nodes in blue text. Categories on the spine that are not in $\mathcal{H}$ are highlighted with blue background. The conventional abbreviations are annotated for each rule.

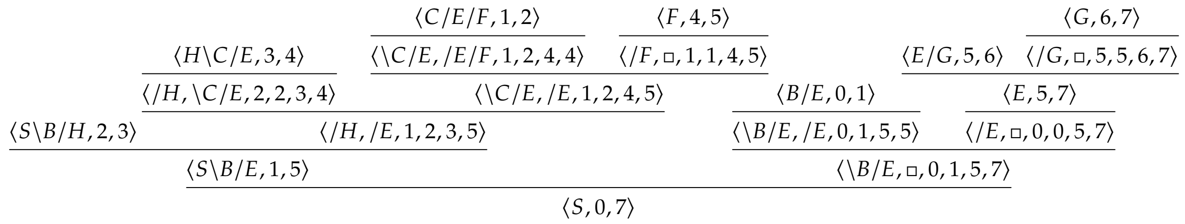


Figure 6.4: Example deduction of the parsing algorithm.

such that $\langle S\backslash B/H, 2, 3\rangle$ can be inserted using deduction rule 2, resulting in $\langle S\backslash B/E, 1, 5\rangle$. The interpretation of this item is as follows: There exists a CCG derivation tree with root category $S\backslash B/E \in \mathcal{H}$ that involves the input symbols of the span $[1, 5]$. This can be verified in Figure 6.3. Note that it is also possible to change the combination order of context items and to first combine $\langle /H, \backslash C/E, 2, 2, 3, 4\rangle$ with $\langle \backslash C/E, /E/F, 1, 2, 4, 4\rangle$ and then to combine the resulting item with $\langle /F, \square, 1, 1, 4, 5\rangle$.

The spinal categories in $\mathcal{H}$ 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 category (and each initial category) of a derivation is represented by some tree item. We can uniquely decompose each CCG derivation tree into a set of primary spines in the sense of the previous chapter (see page 81). At least the lexical anchor and the root of each primary spine are represented by tree items. In the above example, category E as a secondary category is the root of a primary spine consisting of two nodes and accordingly represented by tree item $\langle E, 5, 7\rangle$. Note that for trivial spines consisting of a single node, the lexical anchor and the root coincide.

6.2 Correctness

In this section we prove the correctness of the deduction system by first showing its soundness and then its completeness. For this, we introduce the notion of signature, which associates some specific pieces of derivations with items of our deduction system.

Definition 6.2.1 *A derivation tree t has* signature $\langle c, i, j\rangle$ *if*

1. *the yield of t is $w[i, j]$ and*
2. *the root is labeled by $t(\varepsilon) = c$ with $c \in \mathcal{H}$.*

Definition 6.2.2 *A derivation context C with foot node f has* signature $\langle \alpha, \beta, i, i', j', j\rangle$ *if*

1. *the yield of C is $(w[i, i'], w[j', j])$ and*
2. *for some category $c \in \mathcal{C}(A)$ with* $\mathrm{arity}(c) = \mathrm{downarity}(C, f)$ *we have $C(f) = c\alpha$ and $C(\varepsilon) = c\beta$.*

6.2.1 Soundness

In the following, we will show that all items inferred by the deduction system are valid. More precisely, we will show that for each inferred item, there exists a corresponding derivation tree or context of $\mathcal{G}$.

Theorem 6.2.3 *For each item Z inferred by the deduction system when given input grammar $\mathcal{G}$ and input string w, there exists a derivation tree or derivation context of $\mathcal{G}$ based on w with signature Z.*

Proof. We will prove the soundness of our deduction system by induction on the number of inference steps applied. The base case is clear from the correctness of the axioms: For each input symbol or the empty string, 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 symbol or by '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: Assume that item $\langle /b\alpha, \alpha\beta, i, i, j, h\rangle$ is inferred from $\langle b\alpha\beta, j, h\rangle$. By the induction hypothesis, there is a derivation tree t with signature $\langle b\alpha\beta, j, h\rangle$, so $t(\varepsilon) = b\alpha\beta$. We will use

this category as a secondary category to construct context C with signature $\langle /b\alpha, \alpha\beta, i, i, j, h\rangle$. Thus, let $C = c\alpha\beta(c/b\alpha,\ t)$ with foot node 1, where $c \in \mathcal{C}(A)$ can be chosen arbitrarily (see Figure 6.2a). In the case where $\langle \backslash b\alpha, \alpha\beta, j, h, \ell, \ell\rangle$ is inferred, we have $C = c\alpha\beta(t,\ c\backslash b\alpha)$ with foot node 2. It is clearly a context of $\mathcal{G}$.

Deduction rule 2: Assume that item $\langle c\beta, i, j\rangle$ is inferred from the antecedents $\langle c\alpha, i', j'\rangle$ and $\langle \alpha, \beta, i, i', j', j\rangle$. By the induction hypothesis, there are a derivation tree t' and a context C with foot node f that have these antecedents as their respective signature. By Lemma 6.1.5, we may choose the prefix along the spine of C and therefore assume that $C(f) = c\alpha = t(\varepsilon)$. We obtain the desired derivation tree $t = C[t']$ with signature $\langle c\beta, i, j\rangle$ by inserting t' at the foot node of C (see Figure 6.2b).

Deduction rule 3: Now assume that $\langle \alpha, \beta\beta', i, i', j', j\rangle$ is inferred from $\langle \alpha, \beta\alpha', i'', i', j', j''\rangle$ and $\langle \alpha', \beta', i, i'', j'', j\rangle$. Again, by the induction hypothesis, there exist two derivation contexts C_1 with foot node f_1 and C_2 with foot node f_2 that have these antecedents as signatures. We also assume that $C_2(f_2) = c\beta\alpha' = C_1(\varepsilon)$ by Lemma 6.1.5. We obtain $C = C_2[C_1]$ with signature $\langle \alpha, \beta\beta', i, i', j', j\rangle$ by inserting C_1 at the foot node of C_2 (see Figure 6.2c). Let us regard the spinal node $s = f_2$ and the foot node $f = f_2 f_1$ of C. Then we have $C(f) = c\alpha$, $C(s) = c\beta\alpha'$, and $C(\varepsilon) = c\beta\beta'$. To show that C really is a valid context, we have to verify that the context conditions (6.1) $\text{downarity}(C, f) \leq \text{downarity}(C, u)$ and (6.2) $\text{arity}(C(\varepsilon)) \leq \text{arity}(C(u))$ both hold, where $u \in \text{between}(\varepsilon, f)$. That the other requirements are met, i.e., the presence of combinatory rules involving the categories of a parent node and its children, and the presence of primary categories along the spine, is directly clear from the contexts C_1 and C_2.

- Condition (6.1) is immediately fulfilled for every node in $\text{between}(s, f)$ since C_1 is a context. For node s, note that $\text{downarity}(C, f) = \text{arity}(c) \leq \text{arity}(c\beta) = \text{downarity}(C, s)$. Due to the fact that for every node $v \in \text{between}(\varepsilon, s)$ we have $\text{downarity}(C, s) \leq \text{downarity}(C, v)$, it also follows that $\text{downarity}(C, f) \leq \text{downarity}(C, v)$, so the condition is fulfilled for each node in $\text{between}(\varepsilon, f)$
- For condition (6.2), note that rule 3 is restricted such that $|\beta'| \leq |\alpha'|$. As a result, we have $\text{arity}(C(\varepsilon)) \leq \text{arity}(C(s))$. Since C_2 is a context, each node in $\text{between}(\varepsilon, s)$ has at least the arity of the root as well. Further, since C_1 is a context, each node in $\text{between}(s, f)$ has at least the arity of s, which has lower bound $\text{arity}(C(\varepsilon))$. In summary, we have shown that for each node $u \in \text{between}(\varepsilon, f)$ the condition $\text{arity}(C(r)) \leq \text{arity}(C(u))$ is satisfied.

In particular, we can conclude that if the deduction system yields

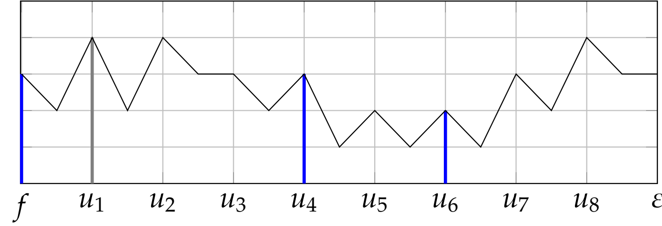


Figure 6.5: Diagram of arities and downstep arities along the spine of a derivation tree with interesting split nodes highlighted.

an item $\langle a_0, 0, |w| \rangle$ with $a_0 \in I$, there is a derivation tree rooted in an initial atomic category a_0 that comprises the entire input. □

6.2.2 Completeness

In the following we prove the completeness of our deduction system. Namely, we show that if there exists a CCG derivation rooted in an initial atomic category $a_0 \in I$ and generating the entire input string w, then item $\langle a_0, 0, |w| \rangle$ 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 Z, the deduction system can infer item Z. 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 6.1.2 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 composed 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 *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 6.2.4 Figure 6.5 illustrates the splitting strategy for derivation trees and contexts. The diagram shows the arities and downstep arities along the spine of a derivation tree t with lexical anchor f. Assume a sequence of spinal nodes denoted by $u_0, u_1, \dots, u_n$, where $u_0 = f$ and $u_n = \varepsilon$. Then each application of a combinatory rule at a node u_i with $i \in \{0, \dots, n-1\}$ can be viewed as having the following steps: the arity before the rule application (arity($t(u_i)$)), the arity after the removal of bridging arguments (downarity(t, u_i)), and the arity after adding the excess (arity($t(u_{i+1})$)). In this way, the progression of arities along the spine is plotted as the sequence

$$\begin{aligned} &\mathrm{arity}(t(u_0)), \quad \mathrm{downarity}(t, u_0), \quad \dots, \\ &\mathrm{arity}(t(u_{n-1})), \ \mathrm{downarity}(t, u_{n-1}), \ \mathrm{arity}(t(u_n)) \ . \end{aligned}$$

The vertical lines of the grid mark arity($t(u_i)$) for $i \in \{0, \dots, n\}$, whereas the respective downstep arities are placed in between two lines of the grid. In this example, we have $n = 9$.

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 arity($t(\varepsilon)$), thus among these nodes, u_6 is chosen as the one closer to the root ε. Accordingly, the derivation tree is split into a smaller derivation tree t' that is rooted in u_6 and a context with foot node u_6. All spinal nodes in t' have arity at least arity($t(u_6)$). 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 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.

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 CCG rule application.

Now we are ready to show the completeness of the deduction system using the procedure discussed and illustrated above.

Theorem 6.2.5 *Let w be an input string. For every derivation tree or derivation context of $\mathcal{G}$ that is based on w and has signature Z, the deduction system can infer item Z.*

Proof. As outlined above, we prove the statement by induction on the number of nodes in the derivation tree or derivation context, where for contexts the foot node is excluded from this count.

Base Case: We start with derivation trees with a single spinal node as the smallest possible derivation tree. Consider a derivation tree t with spine length 1 and span $[i, j]$, in which the root label $t(\varepsilon) = c$ is a lexical category. The child of the root is labeled by an input symbol or by the empty string $'\varepsilon'$. We distinguish two cases depending on this label: Either there exists a lexicon entry $c \in L(w_j)$ and $j = i+1$, or there exists a lexicon entry $c \in L('\varepsilon')$ and $j = i$. In either case, the item $\langle c, i, j\rangle$ is one of the axioms of our deduction system.

Inductive Case 1: We regard contexts with two spinal nodes. Consider a context C with spine length 2 and using spans $[i, i']$ and $[j', j]$ with $0 \leq i \leq i' \leq j' \leq j \leq |w|$. This context takes one of the following two forms, where $c \in \mathcal{C}(A)$, $|b \in \text{args}(L)$, $\alpha \in \mathcal{A}_L(A, 1)$, and $\alpha\beta \in \mathcal{A}_L(A, k)$:

$$\frac{c/b\alpha \quad b\alpha\beta}{c\alpha\beta} \qquad\qquad \frac{b\alpha\beta \quad c\backslash b\alpha}{c\alpha\beta}$$

We write f for the foot node of C. The category $C(f)$ takes the form $c|b\alpha$ and $C(\varepsilon)$ takes the form $c\alpha\beta$. We have $i = i'$ in the left case and $j' = j$ in the right case. We regard the subtree with root category $b\alpha\beta$ that ends at the secondary child of the root node. We have $b\alpha\beta \in \mathcal{H}$ and therefore can infer item $\langle b\alpha\beta, j', j\rangle$ (left case) or $\langle b\alpha\beta, i, i'\rangle$ (right case) by the induction hypothesis. Then, using rule 1 of the deduction system, we can infer item $\langle |b\alpha, \alpha\beta, i, i', j', j\rangle$, the desired signature of the context.

Inductive Case 2: We regard derivation trees of arbitrary size. This case corresponds to rule 2 of the deduction system. Assume a derivation tree t with lexical anchor f, having signature $\langle c\beta, i, j\rangle$ 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 $\langle c\alpha, i', j'\rangle$ and a context C with signature $\langle \alpha, \beta, i, i', j', j\rangle$, such that $t = C[t']$ with $t' = t|_s$, thus s is the foot node of C. Recall that $c\beta \in \mathcal{H}$ by the definition of signature. Note that we can also choose f as the split node s. In this case the resulting tree t' is trivial and corresponds to an axiom of the deduction system. 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 the root label of t' is a category from $\mathcal{H}$ and that C fulfills the context conditions. For

the context, we will only check conditions (6.1) and (6.2), since the other requirements are immediately clear.

- *Subcase 1:* For this subcase, assume there is at least one spinal node n with $\operatorname{arity}(t(n)) < \operatorname{arity}(t(\varepsilon))$. We choose as split node s the spinal node closest to the root with this property. This node can also be f. Formally, the split node is the node u from the set $\{u \in \operatorname{Pref}(f) \setminus \{\varepsilon\} \mid \operatorname{arity}(t(u)) < \operatorname{arity}(t(\varepsilon))\}$ where $|u|$ is minimal. We can write $t(s) = c\alpha$ and $t(\varepsilon) = c\beta$, where α are the bridging arguments of the combinatory rule applied at s and $|\alpha| < |\beta|$ by the assumption of this subcase. Because we chose the spinal node closest to the root with the given property, the arguments in c are not modified at any spinal node between s and ε.
  Since $|\alpha| \in [2]$ and $|\alpha| < |\beta|$, we have $|\beta| \geq 2$. From $c\beta \in \mathcal{H}$ it follows that c is a prefix of a lexical category ($c\beta \in \mathcal{H}_1$) or c is a prefix of an instantiation of a secondary category ($c\beta \in \mathcal{H}_2$). From $|\alpha| \leq 2$ and $|c\alpha| < |c\beta|$ it follows that $c\alpha \in \mathcal{H}$, where $c\alpha \in \mathcal{H}_1$ if $c\beta \in \mathcal{H}_1$ and $c\alpha \in \mathcal{H}_2$ if $c\beta \in \mathcal{H}_2$. Consequently, $\langle c\alpha, i', j'\rangle$ is a valid tree item.
  Furthermore, we need to verify that C is a valid context by checking that (6.1) $\operatorname{downarity}(C, s) \leq \operatorname{downarity}(C, u)$ and (6.2) $\operatorname{arity}(C(\varepsilon)) \leq \operatorname{arity}(C(u))$ for each $u \in \operatorname{between}(\varepsilon, s)$. First, we observe that two nodes v, v' that are labeled by primary categories with $\operatorname{arity}(C(v)) < \operatorname{arity}(C(v'))$ always have $\operatorname{downarity}(C, v) \leq \operatorname{downarity}(C, v')$. By the choice of s, for each $u \in \operatorname{between}(\varepsilon, s)$ we have $\operatorname{arity}(C(s)) < \operatorname{arity}(C(u))$, and thus follows that $\operatorname{downarity}(C, s) \leq \operatorname{downarity}(C, u)$. Second, we have $\operatorname{arity}(C(\varepsilon)) \leq \operatorname{arity}(C(u))$, again because s is the spinal node closest to the root with the property $\operatorname{arity}(C(s)) < \operatorname{arity}(C(\varepsilon))$. We can conclude that C is a valid context.
- *Subcase 2:* For this subcase, assume that every spinal node n has $\operatorname{arity}(t(n)) \geq \operatorname{arity}(t(\varepsilon))$. Among the spinal nodes with minimal downstep arity, we then choose the split node s as the spinal node closest to f. This node can also be f itself. Formally, let $m = \min_{u \in \operatorname{Pref}(f)\setminus\{\varepsilon\}} \{\operatorname{downarity}(t, u)\}$ be the minimal downstep arity. Then the split node is chosen from the set $\{u \in \operatorname{Pref}(f) \setminus \{\varepsilon\} \mid \operatorname{downarity}(t, u) = m\}$ as the node u where $|u|$ is maximal. In other words, when starting at the lexical anchor and moving towards the root node, we choose the last position where an argument of the lexical category is removed. Again, we can write $t(s) = c\alpha$ and $t(\varepsilon) = c\beta$, where α are the bridging arguments of the rule applied at s, and c is the prefix that is not modified at any spinal node between s and ε. By the assumption of this subcase, we have $|\alpha| \geq |\beta|$.

By the choice of s, we know that c is a prefix of the lexical category labeling f. This is because each node $u \in \text{between}(s, f)$ has $\text{downarity}(t, s) < \text{downarity}(t, u)$ and additionally we have $\text{downarity}(t, s) < \text{downarity}(t, f)$ (unless $s = f$). Moreover, since nodes with lower arities than s have at most the downstep arity of s, if there existed such spinal nodes closer to f, they would have been preferred as the split node. Thus, we have $\text{arity}(t(s)) \leq \text{arity}(t(f))$ and $\text{arity}(c\alpha)$ with $|\alpha| \in [2]$ does not exceed the arity of the lexical category labeling f. It follows that $c\alpha \in \mathcal{H}_1 \subseteq \mathcal{H}$.
Next, we show that C is a context. Let $u \in \text{between}(\varepsilon, s)$. The first condition demands $\text{downarity}(t, s) \leq \text{downarity}(t, u)$, which is fulfilled because split node s was picked from the nodes with minimal downstep arity. The second condition requires that $\text{arity}(t(\varepsilon)) \leq \text{arity}(t(u))$ and is already fulfilled by the assumption of this subcase. Therefore, C is a valid context.

Finally, by the induction hypothesis, the deduction system can infer the items $\langle c\alpha, i', j' \rangle$ and $\langle \alpha, \beta, i, i', j', j \rangle$. We conclude that, through an application of rule 2, it can also infer $\langle c\beta, i, j \rangle$.

Inductive Case 3: We regard derivation contexts of arbitrary size. This case corresponds to deduction rule 3. Given a context C with more than two spinal nodes, having foot node f and signature $\langle \alpha, \beta\beta', i, i', j', j \rangle$, we will show that there is a spinal node s such that we can decompose C into two valid contexts C_1 with signature $\langle \alpha, \beta\alpha', i'', i', j', j'' \rangle$ and C_2 with signature $\langle \alpha', \beta', i, i'', j'', j \rangle$, such that $C = C_2[C_1]$ and $C_1 = C|_s$. The node s therefore is the foot node of C_2. We can write $C(f) = c\alpha$, $C(s) = c\beta\alpha'$, and $C(\varepsilon) = c\beta\beta'$, where β might be empty. We choose as the split node s from the nodes in $\text{between}(\varepsilon, f)$ with minimal downstep arity the one closest to f. This corresponds to the approach of subcase 2 for tree splitting, with the difference that foot node f cannot be chosen as the split node. Again, we will only check conditions (6.1) and (6.2) to verify that C_1, C_2 are contexts.

To show that C_1 is a context, let $u \in \text{between}(s, f)$. The first condition (6.1) $\text{downarity}(C, f) \leq \text{downarity}(C, u)$ already holds in C and thus also for the corresponding nodes in C_1. For the second condition (6.2) $\text{arity}(C(s)) \leq \text{arity}(C(u))$, let us first note that $\text{downarity}(C, s) < \text{downarity}(C, u)$, because s is the spinal node closest to f with minimal downstep arity. If there existed a node v with $\text{arity}(C(v)) < \text{arity}(C(s))$ in $\text{between}(s, f)$, it would have $\text{downarity}(C, v) \leq \text{downarity}(C, s)$, a contradiction to the previous observation.

As for C_2, let $u \in \text{between}(\varepsilon, s)$. First, because s was chosen among the spinal nodes with lowest downstep arity, condition

(6.1) $\text{downarity}(C, s) \le \text{downarity}(C, u)$ is fulfilled. Second, condition (6.2) $\text{arity}(C(\varepsilon)) \le \text{arity}(C(u))$ already holds in C.

By the induction hypothesis, the items $\langle \alpha, \beta\alpha', i'', i', j', j'' \rangle$ and $\langle \alpha', \beta', i, i'', j'', j \rangle$ can be inferred. It remains to show that deduction rule 3 can actually be applied. For this, we only have to verify that $|\alpha'| \ge |\beta'|$ holds. This is easy to see as $\text{arity}(C(s)) \ge \text{arity}(C(\varepsilon))$ with $C(s) = c\beta\alpha'$ and $C(\varepsilon) = c\beta\beta'$. Consequently, we can also infer the desired item $\langle \alpha, \beta\beta', i, i', j', j \rangle$. □

6.3 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 $|\mathcal{G}|$ for the size of the input grammar $\mathcal{G}$, defined as the number of characters that we need to write down the lexicon and all the rules in some reasonable representation. We also write k to denote the maximum degree of composition and substitution rules in $\mathcal{G}$.

6.3.1 Argument Contexts and Root Categories

We start our analysis by deriving bounds for the size of sets $\mathcal{A}_L(A, k)$ and $\mathcal{H}$, to be used later. Recall that $\text{args}(L)$ is the set of all arguments in the lexical categories of $\mathcal{G}$. Since every argument appears in our string representation of $\mathcal{G}$, we have $|\text{args}(L)| \in \mathcal{O}(|\mathcal{G}|)$. Using the definition of $\mathcal{A}_L(A, k)$, we obtain $|\mathcal{A}_L(A, k)| \le \sum_{i=0}^{k} |\mathcal{G}|^i$. The right-hand side is the sum of the first $k + 1$ terms of a geometric series. Using the closed-form formula for this sum, we can write

$$\sum_{i=0}^{k} |\mathcal{G}|^i = \frac{|\mathcal{G}|^{k+1} - 1}{|\mathcal{G}| - 1} < \frac{|\mathcal{G}|^{k+1}}{|\mathcal{G}| - 1} = \frac{|\mathcal{G}|}{|\mathcal{G}| - 1} \cdot |\mathcal{G}|^k \le 2 \cdot |\mathcal{G}|^k ,$$

which holds for $|\mathcal{G}| \ge 2$. We thus conclude $|\mathcal{A}_L(A, k)| \in \mathcal{O}(|\mathcal{G}|^k)$.

As for set $\mathcal{H}$, we separately analyze the two subsets $\mathcal{H}_1$ and $\mathcal{H}_2$, according to the definition in Section 6.1.1. Consider a category $c\alpha \in \mathcal{H}_1$, where c is a prefix of a lexical category and $\alpha \in \mathcal{A}_L(A, 2)$. Since every lexical category appears in our string representation of $\mathcal{G}$, 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 $|\mathcal{G}|$. Since $|\mathcal{A}_L(A,2)| \in \mathcal{O}(|\mathcal{G}|^2)$, we conclude that $|\mathcal{H}_1| \in \mathcal{O}(|\mathcal{G}|^3)$.

Consider now set $\mathcal{H}_2$, whose definition is based on the notion of instantiations of secondary categories in rules of $\mathcal{G}$. Recall that according to our convention, and ignoring the ordering of the antecedents, the general form of an instantiated rule of $\mathcal{G}$ is $\frac{b|d\alpha' \quad d\alpha'\beta'}{b\alpha'\beta'}$, where $d\alpha'\beta'$ is the instantiated secondary category, $|d \in \text{args}(L)$, $\alpha' \in \mathcal{A}_L(A,1)$, and $\alpha'\beta' \in \mathcal{A}_L(A,k)$.

Let $c\alpha \in \mathcal{H}_2$, where $c \sqsubseteq d\alpha'\beta'$ for some instantiated secondary category $d\alpha'\beta'$, and $\alpha \in \mathcal{A}_L(A,2)$. We distinguish two cases on the basis of the arity of c.

- arity$(c) \leq$ arity(d): In this case, we have $c \sqsubseteq d$. Every category b with $|b \in \text{args}(L)$ must appear in our string representation of $\mathcal{G}$, and thus each prefix of such category can be associated with some position of the string. Hence, the number of prefixes of these categories does not exceed $|\mathcal{G}|$. Since $\alpha \in \mathcal{A}_L(A,2)$, we conclude that the number of categories $c\alpha \in \mathcal{H}_2$ such that arity$(c) \leq$ arity(d) is in $\mathcal{O}(|\mathcal{G}|^3)$.
- arity$(c) >$ arity(d): In this case, prefix c of $d\alpha'\beta'$ spans over some of the arguments in $\alpha'\beta'$. We can then write c in the form $d\gamma$ for some non-empty argument context γ. Let us associate $d\alpha'\beta'$ with an argument context $|(d)\alpha'\beta'$; similarly, we associate c with an argument context $|(d)\gamma$. Since $|d \in \text{args}(L)$ and $\alpha'\beta' \in \mathcal{A}_L(A,k)$, we have $|(d)\alpha'\beta' \in \mathcal{A}_L(A,k+1)$. By the definition of $\mathcal{H}_2$, the condition arity$(c\alpha) \leq$ arity$(d\alpha'\beta')$ holds. We thus derive $|\gamma\alpha| \leq |\alpha'\beta'|$, yielding $|(d)\gamma\alpha \in \mathcal{A}_L(A,k+1)$. We therefore conclude that the number of categories $c\alpha \in \mathcal{H}_2$ such that arity$(c) >$ arity(d) is in $\mathcal{O}(|\mathcal{G}|^{k+1})$.[3]

3: We may observe that we are overcounting the number of secondary category instantiations. For instance, consider rule instantiations $\frac{D|(A|B) \quad A|B|C}{D|C}$ and $\frac{D|A \quad A|B|C}{D|B|C}$, sharing the same secondary category $A|B|C$. In the first rule, we associate $A|B|C$ with $|(A|B)|C$, while in the second rule, we associate it with $|(A)|B|C$, counting the same secondary category twice. Of course, this is not a problem for the construction of an upper bound.

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

6.3.2 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 $\langle c,i,j \rangle$ satisfies $c \in \mathcal{H}$, the total number of tree items must be in $\mathcal{O}(|\mathcal{G}|^{k+1} \cdot |w|^2)$ for $k \geq 2$ and in $\mathcal{O}(|\mathcal{G}|^3 \cdot |w|^2)$ for $k < 2$. Consider now a context item $\langle \alpha,\beta,i,i',j',j \rangle$. Since $1 \leq |\alpha| \leq 2$ and $\beta \in \mathcal{A}_L(A,k)$, the total number of context items is in $\mathcal{O}(|\mathcal{G}|^{k+2} \cdot |w|^4)$.

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 $\langle \alpha,\beta,i,i',j',j \rangle$, we have $|\alpha\beta| \leq k+2$. Regarding tree items, we need to introduce some auxiliary notation.

Let ℓ be the maximum arity of a lexical category in $\mathcal{G}$ and let m be the maximum arity of a category d for all possible $|d \in \text{args}(L)$. Consider a tree item $\langle c, i, j\rangle$. According to the definition of $\mathcal{H}$, if $c \in \mathcal{H}_1$, then $\text{arity}(c) \leq \text{arity}(e)$ for some lexical category e. If $c \in \mathcal{H}_2$, then $\text{arity}(c) \leq \text{arity}(d\alpha\beta)$ for some $|d \in \text{args}(L)$ and $\alpha\beta \in \mathcal{A}_L(A, k)$. We thus conclude that, for every tree item $\langle c, i, j\rangle$, $\text{arity}(c) \leq \max\{\ell, m + k\} = \rho$.

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

6.3.3 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.

Rule 0 is the simplest rule, producing tree items of the form $\langle c, i-1, i\rangle$ or of the form $\langle c, i, i\rangle$, where c is a lexical category. The total number of lexical categories is in $\mathcal{O}(|\mathcal{G}|)$, 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 $\mathcal{O}(|\mathcal{G}| \cdot |w|)$.

Considering rule 1, let us focus on the case of tree items of the form $\langle b\alpha\beta, j, h\rangle$ producing context items of the form $\langle /b\alpha, \alpha\beta, i, i, j, h\rangle$; 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 $\mathcal{O}(|\mathcal{G}|^{k+1} \cdot |w|^3)$, because of the duplicate occurrences of 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 $\mathcal{O}(|\mathcal{G}|^{k+1} \cdot |w|^3)$.

Rule 2 has premise items $\langle c\alpha, i', j'\rangle$ and $\langle \alpha, \beta, i, i', j', j\rangle$, and consequent item $\langle c\beta, i, j\rangle$. Assume that $k \geq 2$.[4] We have already established that then the number of possible consequent items is in $\mathcal{O}(|\mathcal{G}|^{k+1} \cdot |w|^2)$. Since $|\beta| \leq k$, category $c\beta$ can be split into c and β in at most $k + 1$ ways. Finally, recall that $1 \leq |\alpha| \leq 2$, thus the number of choices for α is in $\mathcal{O}(|\mathcal{G}|^2)$. 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 $\mathcal{O}(k \cdot |\mathcal{G}|^{k+3} \cdot |w|^4)$.

4: If $k < 2$, the number of consequent items is in $\mathcal{O}(|\mathcal{G}|^3 \cdot |w|^4)$. For $k = 1$, this results in $\mathcal{O}(|\mathcal{G}|^5 \cdot |w|^4)$ many instantiations of rule 2. For $k = 0$, we can assume $|\alpha| = 1$, resulting in $\mathcal{O}(|\mathcal{G}|^4 \cdot |w|^4)$ many instantiations of rule 2. In both of these cases the bound is dominated by the number of instantiations of rule 3, as discussed below.

Finally, consider rule 3 with premise items $\langle \alpha, \beta\alpha', i'', i', j', j''\rangle$, $\langle \alpha', \beta', i, i'', j'', j\rangle$ and with consequent item $\langle \alpha, \beta\beta', i, i', j', j\rangle$. We have already established in Section 6.3.2 that the number of possible consequent items is in $\mathcal{O}(|\mathcal{G}|^{k+2} \cdot |w|^4)$. From the side conditions of rule 3, we know that the argument context $\beta\beta'$ 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 $\mathcal{O}(|\mathcal{G}|^2)$. Accounting for the possible range of the indices i'', j'', we then conclude that the number of instantiations of rule 3 is in $\mathcal{O}(|\mathcal{G}|^{k+4} \cdot |w|^6)$.

Putting everything together, and observing that $k \leq |\mathcal{G}|$, 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 $\mathcal{O}(|\mathcal{G}|^{k+4} \cdot |w|^6)$.

6.3.4 Implementation and 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 chapter, 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 $\mathcal{G}$ 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. Since there are $\mathcal{O}(|\mathcal{G}|^{k+4} \cdot |w|^6)$ instantiations, and since the size of each item is $\mathcal{O}(\rho)$, the space requirement for $\mathcal{R}$ is in $\mathcal{O}(\rho \cdot |\mathcal{G}|^{k+4} \cdot |w|^6)$.

Furthermore, for each item Z we construct a list $\mathcal{L}(Z)$ of all rules in $\mathcal{R}$ where item Z occurs as an antecedent. Note that each rule in $\mathcal{R}$ can appear in at most two lists $\mathcal{L}(Z)$. Therefore the total space requirement for all lists $\mathcal{L}(Z)$ is in $\mathcal{O}(\rho \cdot |\mathcal{G}|^{k+4} \cdot |w|^6)$. It is not difficult to see that the data structures $\mathcal{R}$ and $\mathcal{L}(Z)$ can be constructed in time $\mathcal{O}(\rho \cdot |\mathcal{G}|^{k+4} \cdot |w|^6)$ 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 $\mathcal{O}(|\mathcal{G}|^{k+2} \cdot |w|^4)$, and thus the space requirement for $\mathcal{C}$ is in $\mathcal{O}(\rho \cdot |\mathcal{G}|^{k+2} \cdot |w|^4)$. 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 $\mathcal{O}(\rho \cdot |\mathcal{G}| \cdot |w|)$. We then iterate the following steps, until $\mathcal{A}$ becomes empty:

1. pop some item $Z \in \mathcal{A}$ and add Z to $\mathcal{C}$

2. mark as 'active' each occurrence of Z 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 Z_r be the consequent item of r
   (b) if $Z_r \notin \mathcal{A} \cup \mathcal{C}$, add Z_r to $\mathcal{A}$
   (c) remove r from $\mathcal{R}$.

We start by observing that each item Z 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 Z at step 1 in time $\mathcal{O}(\rho)$, that is, in time proportional to the size of Z 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 Z to $\mathcal{A}$.
- When processing item Z at step 2, we retrieve list $\mathcal{L}(Z)$ in time $\mathcal{O}(\rho)$. We then mark as active each occurrence of Z as an antecedent in $\mathcal{R}$. Furthermore, we collect rules 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 Z 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 $\mathcal{O}(\rho)$. Since $|\mathcal{R}| \in \mathcal{O}(|\mathcal{G}|^{k+4} \cdot |w|^6)$, we conclude that the running time of the main loop is in $\mathcal{O}(\rho \cdot |\mathcal{G}|^{k+4} \cdot |w|^6)$. This is also the dominating quantity in the execution of the whole algorithm. Note that ρ is very small in comparison to the other factors. This analysis leads us to the following main result. Although we only regarded pure CCG so far, the algorithm can easily be extended to CCG with rule restrictions as will be discussed in Section 6.5.2.

Theorem 6.3.1 *The universal recognition problem for k-CCG with fixed k with substitution rules and ε-entries can be solved in PTIME.*

6.3.5 Hardness for CCG with ε-entries

We have seen that the universal recognition problem for CCG of bounded rule degree can be solved in PTIME. This holds regardless of whether or not ε-entries are included. If they are allowed, it is quite easy to show that the universal recognition problem is PTIME-complete under logspace-reduction. One way to show PTIME-hardness is to use the fact that the universal recognition problem for context-free grammar with ε-productions is PTIME-hard [36, Corollary 11] and to reduce this problem in logarithmic space to the universal recognition problem for CCG of bounded rule degree. This can be done through our 1-CCG construction of Definition 4.2.4. To be used as a tree automaton, the CFG first has to be converted into the Chomsky normal-form,[5] which can be done in logarithmic space unless removal of ε-productions is performed [23, Theorem 2]. The latter presumably cannot performed in logarithmic space, since checking if the word ε is generated by a given CFG is PTIME-complete [23, Theorem 1]. Therefore, when converting into Chomsky normal-form, we treat ε as a normal symbol and accordingly add ε-entries to the constructed 1-CCG.

5: A CFG $\mathcal{G} = (N, \Sigma, \{s\}, P)$ is in *Chomsky normal-form* if each production has one of the forms $n \to n_1 n_2$ with $n, n_1, n_2 \in N$, or $n \to \alpha$ with $n \in N$ and $\alpha \in \Sigma$, or $s \to \varepsilon$, and s does not appear on any right-hand side of a production [27, page 104].

In the following, we also give a more direct proof that instead uses a reduction from the generability problem, similar to the proof that CFG parsing is PTIME-hard [36, Corollary 11]. For this, pure 0-CCG is in fact sufficient.

Lemma 6.3.2 *The universal recognition problem for pure* 0*-CCG with ε-entries is PTIME-hard.*

Proof. We use a reduction from the *generability problem*: Given a finite set W, a subset $V \subseteq W$, an element $e \in W$, and a binary operation $\circ \colon W \times W \to W$ given as a table, it asks whether the element e is contained in the closure of V under the operation $\circ$. It is known to be PTIME-complete under logspace-reduction [36, Corollary 9]. We construct a CCG $\mathcal{G} = (\emptyset, W, \mathcal{R}(W, 0), \{e\}, L)$ with an empty input alphabet, atomic categories W, the element e as the only initial atomic category, and all application rules. The lexicon is defined as $L('\varepsilon') = V \cup \{c/b/a \mid a \circ b = c\}$. It is easy to see that $\varepsilon \in \mathcal{L}(\mathcal{G})$ if and only if e is in the closure of V under $\circ$. The CCG $\mathcal{G}$ can clearly be constructed in logarithmic space with regards to the size of the instance of the generability problem. □

Corollary 6.3.3 *The universal recognition problem for k-CCG with fixed k with ε-entries is PTIME-complete.*

Whether the universal recognition problem becomes solvable in logarithmic space if ε-entries are excluded from CCG of bounded degree, is an open problem. It is presumably not possible to perform ε-removal from CCG of bounded degree in logarithmic space. This follows from the fact that the above proof also shows that deciding whether ε is generated by some given CCG of bounded degree is PTIME-complete. This also matches with the complexity of checking whether ε is generated by some given CFG [23, Theorem 1].

6.4 From Parse Tree to Derivation Tree

The algorithm presented in Section 6.1 is a recognition algorithm, that is, the algorithm decides whether a given input string 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.

6.4.1 Parse Trees and Parse Forests

We call *parse tree* any tree structure whose nodes are labeled by items produced by our deduction system when processing w, and whose edges connect each antecedent item to its consequent item. As in the case of several tabular parsing algorithms based on context-free grammars [39, Chapter 3], one can easily adapt the algorithm of Section 6.1 to construct a compact representation for the forest of all parse trees for w, as described in what follows.

Whenever an item Z 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. Since Z can be produced by several deduction rules, each item Z is associated

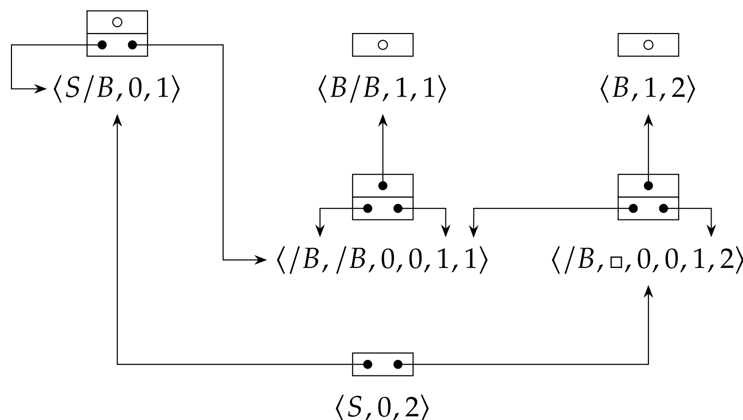


Figure 6.6: Parse forest containing cycles.

with a list $\mathcal{B}(Z)$ 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 an item $\langle a_0, 0, |w|\rangle$ with $a_0 \in I$, arbitrarily pick up a tuple t in $\mathcal{B}(\langle a_0, 0, |w|\rangle)$, and recursively apply the procedure to all of the backpointers in t.[6]

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 [46].

As a side remark, we observe that for some item Z it might happen that, in the process of following the backpointers stored in $\mathcal{B}(Z)$, we end up reaching Z itself. In other words, the constructed parse forest contains some cycles.[7] This happens because our CCG can assign categories to $'\varepsilon'$, 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.

7: Strictly speaking, it is not a forest in that case, but we use the term parse forest nonetheless due to its common usage.

Example 6.4.1 Assume we parse input ab on the basis of a CCG with a lexicon with $L(a) = \{S/B\}$, $L('\varepsilon') = \{B/B\}$, and $L(b) = \{B\}$, where S is initial. Figure 6.6 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 $\langle S/B, 0, 1\rangle$ can be combined with $\langle /B, /B, 0, 0, 1, 1\rangle$, yielding the same tree item $\langle S/B, 0, 1\rangle$ again. Second, context item $\langle /B, /B, 0, 0, 1, 1\rangle$ can be combined with itself, yielding the same context item again. Third, context item $\langle /B, /B, 0, 0, 1, 1\rangle$ can also be combined with $\langle /B, \square, 0, 0, 1, 2\rangle$, resulting in consequent item $\langle /B, \square, 0, 0, 1, 2\rangle$.

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.

6.4.2 Construction of the Derivation Tree

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 of the derivation context 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. The general approach of the construction corresponds to the soundness proof presented in Section 6.2.1.

In this section, we write the special symbol $\square$ at the position of the foot node instead of the category that is associated with it. Before giving the construction, we introduce some auxiliary functions to extract information from items, where w is the input string that was passed to the algorithm:

- $\mathrm{cat}\big(\langle c\alpha, i, j\rangle\big) = c\alpha$,
- $\mathrm{input}\big(\langle c\alpha, i, j\rangle\big) = w[i, j]$
- $\mathrm{dir}\big(\langle |b\alpha, \beta, i', i, j, j'\rangle\big) = |$
- $\mathrm{footcat}\big(\langle \alpha, \beta, i', i, j, j'\rangle,\ c\beta\big) = c\alpha$

The directionality information regarding the first bridging argument of each context item is required to attach the primary and secondary subtrees in the correct order. The category $c\alpha$ labeling the foot node of a context, given the root category $c\beta$, needs to be handed down to a recursive call, where it serves as the root category of another derivation context.

We define the recursive function dtree, which takes a parse tree rooted in a tree item to return a derivation tree. We also define the recursive function dcon, which takes a parse tree rooted in a context item and the intended root category to return a derivation context. Note that in the former case, the root node of the parse tree can be a leaf or a binary node, and in the latter case, the root node of the parse tree can be a unary or binary node.

Definition 6.4.2 *In the following, Z is a tree item, Y is a context item, c is a category, and p_1, p_2, p' are parse trees.*

$$\mathrm{dtree}(Z) = \mathrm{cat}(Z)\big(\mathrm{input}(Z)\big)$$
$$\mathrm{dtree}\big(Z(p_1, p_2)\big) = \mathrm{dcon}\big(p_2,\ \mathrm{cat}(Z)\big)[\mathrm{dtree}(p_1)]$$

$$\mathrm{dcon}\big(Y(p'),\ c\big) = \begin{cases} c\big(\square,\ \mathrm{dtree}(p')\big) & \textit{if}\ \mathrm{dir}(Y) = / \\ c\big(\mathrm{dtree}(p'),\ \square\big) & \textit{if}\ \mathrm{dir}(Y) = \backslash \end{cases}$$

$$\mathrm{dcon}\big(Y(p_1, p_2),\ c\big) = \mathrm{dcon}(p_2,\ c)[\mathrm{dcon}(p_1,\ c')]$$
$$\textit{where}\ c' = \mathrm{footcat}\big(p_2(\varepsilon),\ c\big)$$

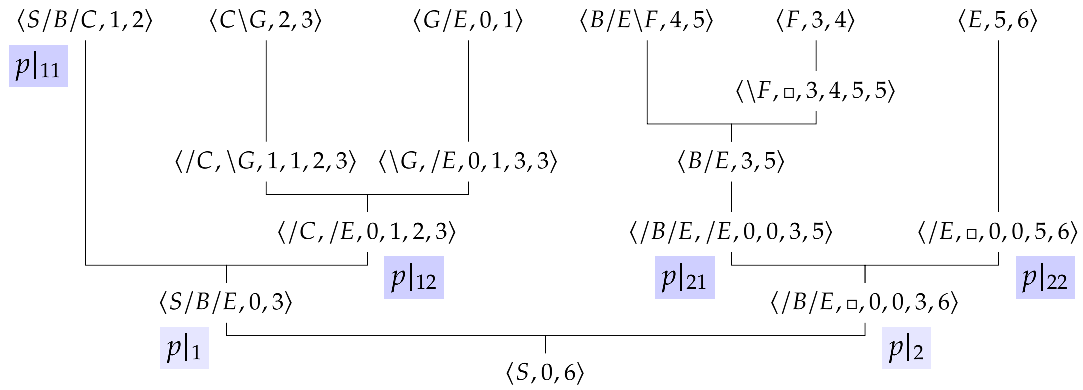


Figure 6.7: Parse tree p with the first child of each binary node labeled by the first antecedent of the respective deduction rule. As in Figure 6.4, the order of leaves does not reflect the order of input categories, which can be reconstructed from the indices stored in the leaf items. The blue labels indicate subtrees referred to in Example 6.4.3 and their shade coincides with their corresponding derivation part in Figure 6.8.

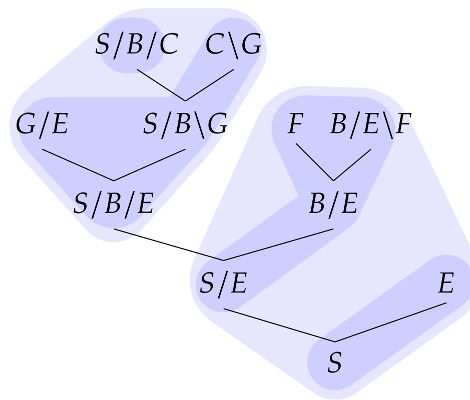


Figure 6.8: CCG derivation tree produced from parse tree p of Figure 6.7. The parts highlighted in light blue result from the first level of recursion, whereas the darker shaded parts result from the second level of recursion. Input symbols are omitted.

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 6.5.1.

Example 6.4.3 Figure 6.7 depicts a parse tree p that was extracted from the parse forest for some input string $w_1 \ldots w_6$. Figure 6.8 shows the CCG derivation tree obtained using the recursive procedure of Definition 6.4.2. The derivation trees and contexts resulting from the first two levels of recursion are highlighted in blue. The corresponding calculation steps are as follows.

We start at the root and find two subtrees $p|_1$, $p|_2$ rooted in $\langle S/B/E, 0, 3\rangle$ and $\langle /B/E, \square, 0, 0, 3, 6\rangle$, and invoke recursive calls. 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 S as an additional parameter.

$$\mathrm{dtree}(p) = \mathrm{dtree}\big(\langle S, 0, 6\rangle(p|_1, p|_2)\big) = \mathrm{dcon}(p|_2, S)[\mathrm{dtree}(p|_1)]$$

To compute the derivation tree corresponding to $p|_1$, category $S/B/E$ is handed down to the call on $p|_{12}$. Subtree $p|_{11}$ consists of a single node $\langle S/B/C, 1, 2\rangle$ and is therefore handled by the base case of the recursive function, returning $S/B/C(w_2)$.

$$\begin{aligned}\mathrm{dtree}(p|_1) &= \mathrm{dtree}\big(\langle S/B/E, 0, 3\rangle(p|_{11}, p|_{12})\big)\\ &= \mathrm{dcon}(p|_{12}, S/B/E)[\mathrm{dtree}(p|_{11})]\\ &= \mathrm{dcon}(p|_{12}, S/B/E)[S/B/C(w_2)]\end{aligned}$$

For the derivation context corresponding to $p|_2$, contexts C_{21}, C_{22}, obtained through calls on $p|_{21}$, $p|_{22}$, are combined such that C_{22} wraps around C_{21}, which is why root category S is passed on to the call on $p|_{22}$. The call on $p|_{21}$ receives as a second parameter the foot category of C_{22}, $\mathrm{footcat}(\langle /E, \square, 0, 0, 5, 6\rangle, S) = S/E$, which is used as the root category of C_{21}.

$$\begin{aligned}\mathrm{dcon}(p|_2, S) &= \mathrm{dcon}\big(\langle /B/E, \square, 0, 0, 3, 6\rangle(p|_{21}, p|_{22}), S\big)\\ &= \mathrm{dcon}(p|_{22}, S)[\mathrm{dcon}(p|_{21}, S/E)]\end{aligned}$$

These derivation parts are then finally combined to the derivation tree of Figure 6.8 by performing the tree substitutions indicated in the equations above. The results of the respective subcomputations also become apparent from Figure 6.8.

6.5 Parser Extensions and Improvements

In this section, we discuss possible extensions and some practical improvements to the parsing algorithm we have developed. We start by discussing spurious ambiguity and its removal. Then we describe the implementation of rule restrictions and of multi-modal variants of CCG. Finally, we present an adjusted algorithm with a

runtime polynomial in the grammar size if all secondary categories in the rule set are instantiated.

6.5.1 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, since it might result in flawed computations of derivation probabilities when working with generative models based on CCG [30]. Note that there exist other notions of spurious ambiguity that aim to obtain only a single derivation tree per semantic reading by arranging forward (resp. backward) chains (i.e., sequences of forward rule applications) in a canonical way [16]. In the present work we will only address the former notion of spurious ambiguity.

We first observe that the algorithm as presented in Section 6.1 has spurious ambiguity. This can easily be seen from Figure 6.9, which shows two different parse trees that correspond to the same derivation tree. On the other hand, the parse tree shown in Figure 6.10b corresponds to the derivation tree shown in Figure 6.10a, which is different from the one in Figure 6.9a. So the parse tree is not redundant with those of Figure 6.9 and not a case of spurious ambiguity, 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 6.11, depicting two parse trees corresponding to the same derivation context, where c is a placeholder for an arbitrary category.

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 spurious ambiguity. Throughout the discussion, we always assume some general but fixed derivation tree which we call the reference derivation tree. We also assume that its root is labeled by an initial atomic category.

Deduction Rule 0 This deduction rule introduces the lexical categories associated with the input symbols. Given a reference derivation tree, there is only one choice for each input symbol: the tree item composed of the associated lexical category in the derivation and the position in the input string. Therefore, deduction rule 0 is not responsible for any spurious ambiguity.

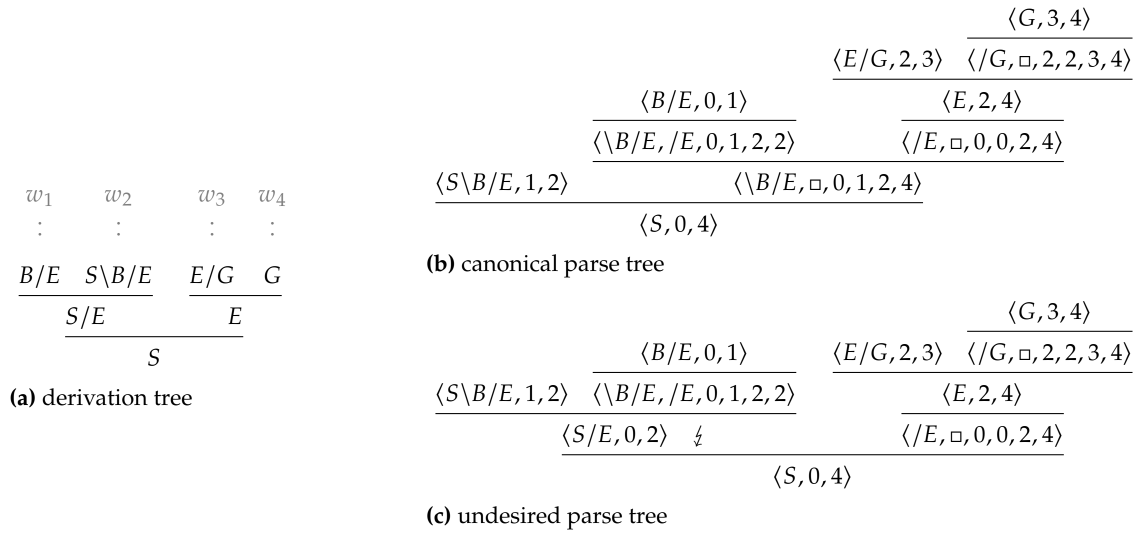


Figure 6.9: Spurious ambiguity caused by deduction rule 2.

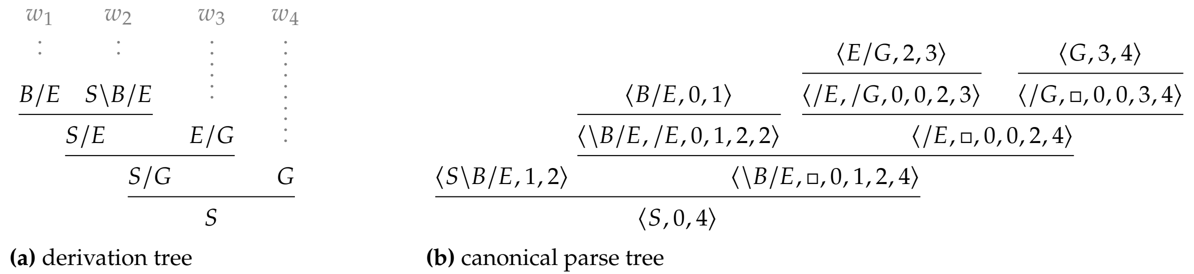


Figure 6.10: Other derivation tree with the same lexical categories as in Figure 6.9.

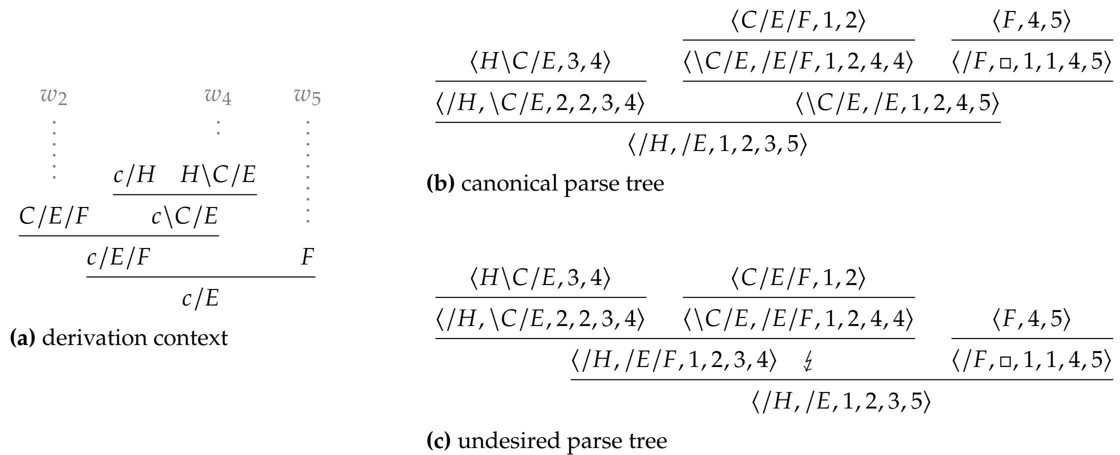


Figure 6.11: Spurious ambiguity caused by deduction rule 3. In the derivation context, c can be an arbitrary category.

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 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 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 category and thus determined by the reference derivation tree as well. Third, the guessed indices of the consequent context item are the left (resp. right) fencepost position of the reference derivation subtree that is rooted in the sibling of the secondary 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 and their consequent items. Any change of this set would result in a different derivation tree. This is the case for the parse trees of Figures 6.9b and 6.10b.

Deduction Rules 2 and 3 Consider the spine of some subtree of our reference derivation tree, such that the root is labeled by a secondary or initial category. The lexical anchor of the spine is introduced as a tree item by deduction rule 0, and when the root is reached, this is either the goal item or else a tree item storing a secondary 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 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 as 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 thus constitute different ways to group the combinatory rule applications along the spine into valid contexts without changing their order.

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 algorithm is still complete.

When two items are combined via rule 2 or 3, we will 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 spinal 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 Assume that the item Z was obtained by adding context item Y_1 to some other item. Then we want to ensure that another context item Y_2 can only be added to Z if it is not possible to add Y_2 to Y_1 first. 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. Else, 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.

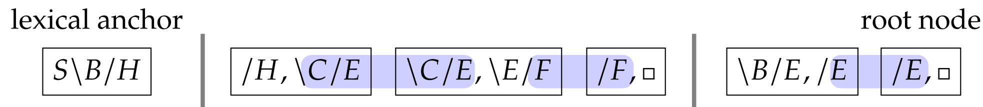


Figure 6.12: Grouping of combinatory rule applications along the spine of the derivation tree of Figure 6.3 into maximally extended contexts.

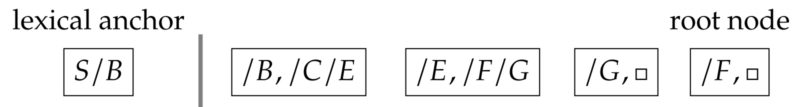


Figure 6.13: Group starting with two combinatory rule applications that increase the arity.

Example 6.5.1 In Figure 6.9c, tree item $\langle S\backslash B/E, 1, 2\rangle$ is combined with context item $Y_1 = \langle \backslash B/E, /E, 0, 1, 2, 2\rangle$ with excess length 1. Subsequently, context item $Y_2 = \langle /E, \square, 0, 0, 2, 4\rangle$ with one bridging argument is added to the consequent item $Z = \langle S/E, 0, 2\rangle$. Because excess length 1 is stored in Z, context item Y_2 could also have been added to Y_1, so this application of deduction rule 2 is not allowed (marked by $\lightning$). Consequently, the canonical parse tree of Figure 6.9b is enforced.

Example 6.5.2 Similarly, in Figure 6.11c, after combining context item $Y = \langle /H, \backslash C/E, 2, 2, 3, 4\rangle$ with $Y_1 = \langle \backslash C/E, /E/F, 1, 2, 4, 4\rangle$ with excess length 2, another context item $Y_2 = \langle /F, \square, 1, 1, 4, 5\rangle$ with one bridging argument is added to the consequent item. Again, this is forbidden because Y_1 and Y_2 could have been combined first and then added to Y in one step. Thus, the canonical parse tree of Figure 6.11b is enforced.

Explanation We claim that this approach suffices to remove all spurious ambiguity from the parsing algorithm. This is supported by the following argument.

We first focus on the splitting of trees via 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. 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 last two 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.

Example 6.5.3 Figure 6.12 shows the items (without indices) corresponding to the combinatory rule applications along the spine of the derivation tree of Figure 6.3 and visualizes how these items are grouped into maximally extended contexts. Their order is fixed by the reference derivation tree, so each item can only be combined with the neighboring items or their respective consequents. After combining $\langle \backslash C/E, \backslash E/F \rangle$ with $\langle /F, \square \rangle$, we can still combine the consequent with $\langle /H, \backslash C/E \rangle$, since the bridging arguments $\backslash C/E$ of the first antecedent are preserved. In the same manner, when combining $\langle \backslash C/E, \backslash E/F \rangle$ with $\langle /H, \backslash C/E \rangle$ first, the consequent can still be extended in the direction of the root by adding $\langle /F, \square \rangle$ to it, since the excess $\backslash E/F$ of the second antecedent is transferred to the consequent.

Example 6.5.4 Using Figure 6.13, we demonstrate what happens when we add smaller contexts than the intended maximal context to some tree item. Assume that $\langle /B, /C/E \rangle$ is added to $\langle S/B \rangle$. Provided that the resulting category is in $\mathcal{H}$, this is allowed since the context item increases the arity of the tree item. Next, we add $\langle /E, /F/G \rangle$ to the consequent. This is allowed as well since the two context items cannot be combined directly. However, it is forbidden to add $\langle /G, \square \rangle$ afterwards. This item reduces the arity and cannot be added to the combined item $\langle S/C/F/G \rangle$, 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 long as the result is a category in $\mathcal{H}$), but at some point we need to preserve or reduce the arity in order to attain the arity at 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 formal treatment and proof are necessary to rule out that the modified algorithm has any spurious ambiguity. This should be addressed in future work.

Separation of Splitting Strategies A property of the described strategy is that by following it, the two cases of the tree splitting in the completeness proof (see Theorem 6.2.5, inductive case 2) 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 $\mathcal{H}_1$ or $\mathcal{H}_2$, respectively. More precisely, closer to the root, which is labeled by an instantiation of a secondary category or an initial atomic category, the categories stored in the first antecedent and the consequent item of deduction rule 2 are in $\mathcal{H}_2$ 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 $\mathcal{H}_1$ 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 the spinal node closest to the root among those of lowest arity. At this position, the spinal category is in $\mathcal{H}_1 \cap \mathcal{H}_2$.

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 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.

6.5.2 Support for Rule Restrictions

A support for non-pure CCG and for rule restrictions in particular is quite easy to implement. We employ the same approach that was proposed by Kuhlmann and Satta [54], who pointed out that it boils down to the same solution that was already employed for the classical polynomial time parsing algorithm by Vijay-Shanker and Weir [91, 92].

There are two types of rule restrictions: target restrictions, which restrict the target of the primary category, and secondary restrictions, which restrict the secondary category of a rule. When a category is used as a secondary 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 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 restrictions as well as the support for non-pure CCG. Additionally, if also target restrictions are 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 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 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. Since 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 $\mathcal{G}$. 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.

6.5.3 Support for Multi-Modal CCG

While rule restrictions provide derivational control by allowing specific rules only for a subset of categories, there is a shift towards multi-modal 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 [5, 6, 50, 83, 88]. For example, $/_\diamond$ makes a category accessible to a forward harmonic rule, thus the combinatory rule $\frac{X/_\diamond Y \quad Y/_\diamond Z}{X/_\diamond Z}$ is allowed, but $\frac{X/_\diamond Y \quad Y\backslash_\diamond Z}{X\backslash_\diamond Z}$ is not. In most variants, both the primary and secondary category need to be equipped with slash types that permit the respective rule. However, in the variant proposed by Stanojević and Steedman [83], only the outermost slash type of the primary category restricts the allowed rules, and the slash types in the secondary category are simply copied into the output category, thus permitting $\frac{X/_\diamond Y \quad Y/_\times Z}{X/_\times Z}$. Note that the formal properties of multi-modal CCG depend on the precise specification of operators; the generative capacity can be lower than that of CCG with rule restrictions [49, 50].

Our parsing algorithm can easily be adapted to multi-modal 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 multi-modal 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 category. For instance, from items of the form $\langle Y/_\diamond Z, j, h\rangle$, we may infer $\langle /_\diamond Y, /_\diamond Z, i, i, j, h\rangle$, but not $\langle \backslash_\times Y, /_\diamond Z, j, h, \ell, \ell\rangle$, 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 are consistent with the slash types of the bridging arguments stored in the second antecedent.

Alternatively, some versions of multi-modal CCG can be converted into an equivalent CCG with rule restrictions using the construction by Baldridge and Kruijff [6] before employing the extension described in the previous section.

6.5.4 Instantiated Secondary Categories

We have seen that the runtime complexity of the algorithm is exponential in the maximum rule degree k of the grammar. This holds true for a CCG whose combinatory rules may contain variables in their secondary categories, which thus require proper instantiation with all possible lexical arguments. However, when considering a grammar where the secondary categories in the combinatory rules do not contain any variables, we can modify the deduction system such that the 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 $\mathcal{H}$ (in combination with all spans of the input, but we are only concerned with the grammar size here). $\mathcal{H}_1$ has size polynomial in $|\mathcal{G}|$ already when secondary categories may contain variables. The size of $\mathcal{H}_2$ becomes also polynomial if all instantiated secondary categories are part of $\mathcal{G}$, since in the same manner as for $\mathcal{H}_1$, 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 context of length k or smaller in the excess, they have to be restricted such that the bridging arguments together with the excess follow the same pattern as $\mathcal{H}_2$ 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 $\mathcal{H}_2$, 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 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 category. If a substitution rule is simulated, one of the arguments of the secondary 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 last arguments of the excess, either by exchanging or by removing them. Consequently, we start with a (segmented) category in $\mathcal{H}_2$ when the context item is introduced, and if the first antecedent of deduction rule 3 contained a category in $\mathcal{H}_2$ 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 $|\mathcal{G}|$. We can conclude that the number of context items is polynomial in $|\mathcal{G}|$. As the number of items is polynomial in the grammar size, the same holds for the number of instantiations of deduction rules.

Conclusion 7

In this chapter, we summarize the results presented in this thesis and discuss them with regard to their connection and implications within a larger context. We close with an outlook by turning to open questions that should be addressed in future work.

7.1 Summary 137
7.2 Discussion 141
7.3 Outlook 142

7.1 Summary

The aim of this thesis was to study the generative and computational power of CCG. This allows to identify properties that are desirable in the sense that they entail an expressivity that is high enough to capture natural language or limit computational complexity to make parsing feasible. In both areas, similar techniques can be used, like decomposing derivations into smaller parts (Sections 5.1, 6.2), encoding of trees (Definition 5.1.1) [44, 91], or tree rotation (Sections 4.2.1, 5.2) [16, 49, 50, 82]. As a result, one area may benefit from progress in the respective other. In this way, a formal approach can not only deepen our understanding of CCG and related formalisms, but might also shape the design of practical applications.

7.1.1 Generative Power

The generative power of CCG was studied in Chapters 4 and 5. First, in Chapter 4, we examined CCG with low rule degrees. We gave a new proof of the characterization of classical categorial grammar [10, Theorem 1.1], or more generally, CCG with application rules (Theorem 4.1.9). It can generate the regular tree languages whose min-height is bounded.[1] This holds regardless of whether the CCG is pure. For CCG with composition of first degree we have shown that it can generate exactly the regular tree languages (Theorem 4.2.6). If it is pure, it is less powerful with regard to tree languages (Theorem 5.2.2). However, in the string case, pure CCG with composition of first degree can, like pure CCG with only application rules (Theorem 4.1.1), still generate exactly the context-free languages (Theorem 4.2.8).

1: Recall that a tree or tree language is min-height bounded if for each node, there exists a short path to a leaf, such that the length of the path does not exceed the bound (see Section 4.1).

In Chapter 5, we covered CCG with arbitrary degrees of composition. Our main result on the generative power of CCG is its strong equivalence to TAG in terms of tree languages (Corollary 5.7.2). For this, composition of degree 2 and first-order categories are already sufficient (Corollary 5.7.4). Moreover, ε-entries can be

omitted (Corollary 5.7.3). This is is an important finding since these lexicon entries are in conflict with the Principle of Adjacency [86, page 54] and are computationally demanding [55]. Our construction provides a procedure for removing ε-entries from a given CCG by first converting it into an sCFTG, then trimming the symbol ε from the productions of the grammar, and finally converting it back into a CCG without ε-entries. This also reduces the rule degree to 2. The class of tree-adjoining languages (see Section 2.4) is well-studied, and due to the equivalence result, the knowledge regarding it can be transferred to CCG. Notable strongly equivalent formalisms are sCFTG (see Section 2.3.1) and spine grammar (see Section 5.3), which we used in our proofs, but also the linear top-down push-down tree automaton [21]. While ε-entries can be avoided without affecting generative power, the option to restrict the rule set is indispensable. As we have seen, pure CCG cannot even generate all local tree languages (Theorem 5.2.2). For an overview of the generative power of various variants of CCG, we refer to Table 1.1.

We briefly discuss the table entries that immediately follow by combining a result from the literature and a result of this thesis. For prefix-closed CCG without target restrictions, it is known that it can generate only a proper subset of tree-adjoining languages in the string case [50]. Due to the fact that CCG generates only tree-adjoining languages in the tree case as well (Corollary 5.7.2), here we also have a proper inclusion in this set. On the other hand, CCG with generalized composition of unlimited degree and ε-entries is strictly more expressive than TAG [97]. This is shown by specifying a CCG that generates a non-tree-adjoining language, which, however, does not rely on ε-entries. Since CCG without ε-entries can generate exactly the tree-adjoining (tree) languages (Corollary 5.7.2), we can transfer the result to this variant and the tree case.

Finally, we would like to draw the attention to the major role that the notion of spines played in this thesis, occurring in most of the proofs. It is noteworthy that the sets of primary spines of the (relabeled) derivation trees of 0-CCG, 1-CCG, and k-CCG form finite, regular, and context-free languages, respectively. This reminds of the language hierarchy proposed by Weir [96], which aims to generalize the step from context-free languages to tree-adjoining languages by defining language classes based on trees consisting of independent paths of increasing expressivity.

Grammar Size

The size of the sCFTG constructed in Definition 5.1.2 is exponential in a grammar-specific constant that depends on the maximum

arity of secondary categories and of lexicon entries, in the worst case leading to an exponential blow-up of the grammar size. Note that the maximum arity of a secondary category is not only determined by the rule degree, but also by the maximum arity of categories occurring in lexical arguments. Without any specific constraints regarding the CCG, due to the different computational complexities of the universal recognition problem for TAG and CCG (see Table 1.2), a translation from CCG to sCFTG in polynomial time is not possible unless PTIME = NP. Note that sCFTG with bounded node rank can be converted to TAG in polynomial time [45].[2,3] However, using the ideas that were used in the parsing algorithm presented in Section 6.1, it is expected that the construction can be improved such that its size is exponential only in the maximum rule degree of the CCG. More specifically, the nonterminals of the sCFTG could be restricted to represent only a certain subset of categories, like the categories occurring in tree items. Moreover, when all secondary categories in the rule set are instantiated,[4] by using the method described in Section 6.5.4, the construction could become polynomial in the grammar size through a restriction of the nonterminals that are similar to context items.

2: For this, the sCFTG is made *collapse-free* [45, Proposition 2] (i.e., not containing productions of the form $n \to \square$ for $n \in N$) and footed [45, Proposition 3] before it is converted into a TAG [45, Proposition 6]. These two steps in the worst case require time exponential in the maximum size of a production. However, it is possible to modify the grammar beforehand such that the size of each production is bounded by a constant. This can be achieved as we only consider trees with node rank at most 2 and takes polynomial time.

3: Also interesting in this context is the fact that the universal recognition problem for well-nested LCFRS with fan-out 2 can be solved in polynomial time [24]. This formalism has the same expressive power as TAG as well [41, Theorem 5.2].

4: This was actually an assumption in Chapter 5, albeit only for convenience.

For the conversion from TAG to CCG, the size of the constructed CCG is polynomial in the size of the given TAG. This can be verified for each individual step of the construction, including that from TAG to spine grammar [21, 45],[5] all intermediate steps, and finally the construction of Definition 5.6.1.

5: For this, the TAG is first converted into a footed simple CFTG [45, Proposition 7] that is then brought into spine grammar normal form [21, Theorem 1]. Due to the maximum node rank 2, establishing the normal form can be achieved in polynomial time. In the normalization procedure of Theorem 5.3.4, the removal of ε-productions takes polynomial time since the maximum size of a production is bounded.

The removal of ε-entries from a CCG through successive execution of the two constructions, with an ε-removal from the sCFTG in between (Corollary 5.7.3), thus results in an exponential increase of the grammar size in the worst case. Again, considering the different complexities of the universal recognition problem for CCG with and without ε-entries (see Table 1.2), the removal of ε-entries cannot be performed in polynomial time unless NP = EXPTIME [55, page 476]. As outlined above, the situation changes if we impose certain restrictions on the CCG.

Closure Properties

Due to the characterizations shown for 0-CCG, 1-CCG and k-CCG, we can derive a number of closure properties, of which the most important ones are collected in Table 7.1. For more details on the closure properties of TAL, see Section 2.4.2. For the closure properties of RTL, see [22, Sections 1.5, 2.4]. From these, several closure results for the tree languages generatable by 0-CCG can be derived.

Table 7.1: Closure properties of the tree languages generatable by 0-CCG, 1-CCG, and k-CCG.

closure	0-CCG	1-CCG (RTL)	k-CCG (TAL)
union	✓	✓	✓
intersection	✓	✓	✗
intersection with RTL	✓	✓	✓
complement	✗	✓	✗
relabeling	✓	✓	✓
α-concatenation	✓	✓	✓
α-iteration	✗	✓	✓

7.1.2 Computational Power

In Chapter 6, a new parsing algorithm for CCG is proposed. This contributes to the understanding of the computational power of CCG. When considering the influence of the input grammar size on the runtime, it is found to be exponential only in the maximum rule degree of the grammar (see Section 6.3). As a consequence, bounding the rule degree by a constant leads to a parsing algorithm that is polynomial in the grammar size (Theorem 6.3.1). If ε-entries are included in the grammar, the universal recognition problem for CCG with bounded rule degree is PTIME-complete under logspace-reduction (Corollary 6.3.3). This refines the results of Kuhlmann, Satta, and Jonsson [55], who showed that the universal recognition problem is NP-complete for CCG without ε-entries and EXPTIME-complete if ε-entries are included. Table 1.2 summarizes the results on the computational power of CCG. The fact that parsing is exponential only in the maximum rule degree of the grammar is particularly interesting since, as we have seen, rule degree 2 is sufficient for the generative power in terms of string [93] and tree languages (Corollary 5.7.4). Another contribution is that our algorithm incorporates substitution rules, which, despite their practical relevance, have hardly been addressed in a theoretical setting so far. We propose several extensions of our algorithm. One of them is a modification such that, if all secondary categories of the grammar are instantiated, i.e., if they do not contain any variables, the algorithm's runtime is polynomial in the grammar size as well. This emphasizes the importance of category variables in CCG rules for the complexity results of Kuhlmann, Satta, and Jonsson [55]. Moreover, we discuss the retrieval of a CCG derivation tree from a parse tree, the implementation of rule restrictions and multi-modal CCG, and the removal of spurious ambiguity.

Runtime Complexity

Let w be the input string and let $\mathcal{G}$ be the input grammar with rules of degree at most k. Then the algorithm runs in $\mathcal{O}(\rho \cdot |\mathcal{G}|^{k+4} \cdot |w|^6)$,

where $\mathcal{O}(\rho)$ is the space required for the representation of an item. The basic algorithm does not take into account rule restrictions. If target restrictions are included, a multiplicative factor $|A|$ (i.e., the number of atomic categories) needs to be added. The equally expressive TAG is known to be parsable in $\mathcal{O}(|\mathcal{G}|^2 \cdot |w|^6)$ [74].

7.2 Discussion

We now broaden the view and discuss our results in a wider context. We first take a look at what properties a suitable CCG formalism should have. From a tree language perspective, rule degree 2 and only composition rules are sufficient to achieve the same expressivity as TAG, which as a formalism has gained widespread recognition in natural language processing. While ε-entries can safely be omitted, rule restrictions are necessary in order to maintain the generative power. Bounding the maximum rule degree by a low constant is highly desirable to keep the computational complexity low. A low rule degree is also supported by linguistic work, that proposes for example a maximum rule degree of 3 as sufficient for English [86, page 43].

Kuhlmann, Satta, and Jonsson [55] show that CCG is more *succinct* (cf. [28]) than TAG. This means that for generating the same language class fewer resources in terms of grammar size are necessary. This is a consequence of the different complexities of the universal recognition problem together with effective transformation procedures. More precisely, unless PTIME = NP, it is clear that there exist CCGs such that the constructed weakly equivalent TAG has to be at least exponentially larger. If ε-entries are allowed, no assumption is needed, since PTIME ≠ EXPTIME always holds. On the other hand, TAG can be converted into an equivalent CCG with only polynomial increase of the grammar size. Our constructions can not only convert a TAG into a CCG and vice versa, but they also provide a way to remove ε-entries and reduce the maximum rule degree to 2. We can therefore, due to the complexity results for CCG with and without ε-entries and for CCG of bounded rule degree, conclude that CCG with ε-entries is more succinct than CCG without ε-entries, and that CCG of unbounded rule degree is more succint than CCG with rule degree 2. This also means that in general it is not possible to reduce computational effort by converting a CCG with ε-entries and a higher rule degree into a CCG without ε-entries and with rule degree 2, as this could also result in a larger grammar, balancing out the more efficient parsing with respect to grammar size. Therefore, together with the Principle of Adjacency [86, page 54], which rejects ε-entries, this should rather be understood as a directive for the design of grammars.

Our proposed approach for parsing CCG with instantiated secondary categories in polynomial time also demonstrates that such a CCG is less succinct than one that uses variables in secondary categories. It intuitively makes sense that the high complexity of the universal recognition problem of CCG in comparison to TAG is a consequence of its ability to define a grammar in a concise way through the use of variables. Although it may be responsible for the high parsing complexity with regard to grammar size, precisely the capacity to specify languages in a short way is an appealing feature of CCG, and restricting it might therefore be undesirable.

7.3 Outlook

It has become apparent that rule restrictions are necessary, and it is true that *some* kind of rule restrictions are necessary. We regarded CCG with target and secondary restrictions that are implemented in the rule system. Multi-modal CCG (see Section 6.5.3) offers a certain degree of control over which rules can be applied, entirely through the lexicon and without resorting to rule restrictions. However, Kuhlmann, Koller, and Satta [49, 50] showed that without target restrictions, the expressive power is reduced even for a variant of multi-modal CCG. Another variant has been proposed that possibly avoids this shortcoming since it is not prefix-closed [83]. Since there are different ways to define multi-modal CCG, future work should study its generative capacity under different conditions.

Other interesting questions arise from the inclusion of additional rule types. For substitution, there exists a proof sketch for the string case [86, page 210] that demonstrates that it at least does not affect the weak generative capacity. There is an ongoing discussion regarding the question how type-raising should be implemented in CCG, and under what circumstances it can be simulated by standard CCG. In particular, this concerns the question whether type-raising should be treated as a *lexical* or *grammatical* operation [29, 34]. Steedman [86, page 211] gives an intuition of how CCG with type-raising could be simultated by indexed grammar. He argues that under the condition that type-raising is merely a finite schema, i.e., each of the category variables y, z in $\frac{z}{y/(y\backslash z)}$ and $\frac{z}{y\backslash(y/z)}$ ranges over a finite set of categories, only linear productions are needed and it can thus be simulated by LIG. Since LIG is weakly equivalent to CCG [93], this would also mean that type-raising can be compiled into the lexicon. Komagata [48] suggests several other restrictions on type-raising rules to keep the generative power equivalent to the standard CCG formalism. Hoffman [33] shows that the use of variables in the lexicon that allow instantiation with arbitrary categories can raise the generative power of CCG over

that of TAG. She uses categories of the form $y\backslash B/(y\backslash C)$, where y is a category variable and B, C are atoms. These categories are different from what we see in type-raising, but this indicates that the use of type-raised categories with variables in the lexicon might increase the generative power as well. It still needs to be determined if this is indeed the case, and if true, how much the generative capacity increases.

This thesis considers strong generative capacity in terms of tree languages consisting of constituency trees. From another perspective, it can be viewed in terms of dependency. Interestingly, differences between TAG and CCG have been exposed in this regard, where dependency is expressed either as generated dependency tree sets [47] or as permutations [83]. A characterization of the class of dependency tree sets generatable by CCG would help to better understand the differences between these formalisms.

In the area of computational complexity, it is an open problem if the universal recognition problem for pure CCG is solvable in polynomial time. We have seen that the universal recognition problem for CCG of bounded rule degree with ε-entries is presumably not solvable in small (i.e., logarithmic) space, since it is PTIME-complete under logspace-reduction (Corollary 6.3.3). However, it is unclear whether this might be possible if ε-entries are omitted.

Bibliography

[1] Vito Michele Abrusci, Christophe Fouqueré, and Jacqueline Vauzeilles. 'Tree adjoining grammars in noncommutative linear logic'. In: *International Conference on Logical Aspects of Computational Linguistics (LACL)*. Springer. 1996, pp. 96–117 (cited on page 22).

[2] Kazimierz Ajdukiewicz. 'Die syntaktische Konnexität'. In: *Studia Philosophica* 1 (1935), pp. 1–27 (cited on page 1).

[3] Jean-Michel Autebert, Jean Berstel, and Luc Boasson. 'Context-Free Languages and Pushdown Automata'. In: *Handbook of Formal Languages*. Ed. by Grzegorz Rozenberg and Arto Salomaa. Vol. 1. Springer, 1997. Chap. 3, pp. 111–174 (cited on pages 15, 74, 75).

[4] Franz Baader and Tobias Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998 (cited on page 18).

[5] Jason Baldridge. 'Lexically Specified Derivational Control in Combinatory Categorial Grammar'. PhD thesis. University of Edinburgh, 2002 (cited on pages 1, 134).

[6] Jason Baldridge and Geert-Jan M. Kruijff. 'Multi-Modal Combinatory Categorial Grammar'. In: *Proceedings of the Tenth Conference of the European Chapter of the Association for Computational Linguistics (EACL)*. 2003, pp. 211–218 (cited on page 134).

[7] Yehoshua Bar-Hillel. 'A quasi-arithmetical notation for syntactic description'. In: *Language* 29.1 (1953), pp. 47–58 (cited on page 1).

[8] Yehoshua Bar-Hillel, Haim Gaifman, and Eli Shamir. 'On categorial and phrase-structure grammars'. In: *Bulletin of the Research Council of Israel* 9F.1 (1960), pp. 1–16 (cited on pages 1, 4, 6, 30, 35, 36, 50).

[9] Jean Berstel. *Transductions and Context-Free Languages*. Vol. 38. Leitfäden der angewandten Mathematik und Mechanik. B. G. Teubner, 1979 (cited on pages 14, 78, 79).

[10] Wojciech Buszkowski. 'Generative Power of Categorial Grammars'. In: *Categorial Grammars and Natural Language Structures*. Ed. by Richard T. Oehrle, E. Bach, and Deirdre Wheeler. Vol. 32. Studies in Linguistics and Philosophy. Springer, 1988. Chap. 4, pp. 69–94 (cited on pages 6–8, 36, 137).

[11] Wojciech Buszkowski. 'Mathematical linguistics and proof theory'. In: *Handbook of logic and language*. Elsevier, 1997, pp. 683–736 (cited on page 7).

[12] Noam Chomsky. 'Three models for the description of language'. In: *IRE Transactions on Information Theory* 2.3 (1956), pp. 113–124 (cited on page 1).

[13] Haskell B. Curry, Robert Feys, and William Craig. *Combinatory Logic*. Vol. 22. Studies in Logic and the Foundations of Mathematics. North-Holland, 1958 (cited on page 1).

[14] Normann Decker, Martin Leucker, and Daniel Thoma. 'Impartiality and Anticipation for Monitoring of Visibly Context-Free Properties'. In: *Runtime Verification: 4th International Conference*. Ed. by Axel Legay and Saddek Bensalem. Vol. 8174. LNCS. Springer, 2013, pp. 183–200 (cited on page 74).

[15] Manfred Droste, Sven Dziadek, and Werner Kuich. 'Weighted simple reset pushdown automata'. In: *Theoretical Computer Science* 777 (2019), pp. 252–259 (cited on page 16).

[16] Jason Eisner. ‘Efficient Normal-Form Parsing for Combinatory Categorial Grammar’. In: *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics (ACL)*. 1996, pp. 79–86 (cited on pages 126, 137).

[17] Elisabet Engdahl. ‘Parasitic gaps’. In: *Linguistics and philosophy* (1983), pp. 5–34 (cited on page 3).

[18] Herbert Fleischner. ‘On the equivalence of Mealy-type and Moore-type automata and a relation between reducibility and Moore-reducibility’. In: *Journal of Computer and System Sciences* 14.1 (1977), pp. 1–16 (cited on pages 74, 75).

[19] Timothy A. D. Fowler and Gerald Penn. ‘Accurate context-free parsing with combinatory categorial grammar’. In: *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics (ACL)*. 2010, pp. 335–344 (cited on pages 4, 6, 46).

[20] Akio Fujiyoshi. ‘Restrictions on monadic context-free tree grammars’. In: *Proceedings of the 20th International Conference on Computational Linguistics (COLING)*. 2004, pp. 78–84 (cited on pages 17, 95, 96).

[21] Akio Fujiyoshi and Takumi Kasai. ‘Spinal-formed context-free tree grammars’. In: *Theory of Computing Systems* 33.1 (2000), pp. 59–83 (cited on pages 8, 22, 66, 68, 69, 94, 95, 138, 139).

[22] Ferenc Gécseg and Magnus Steinby. ‘Tree Automata’. 2015 (cited on pages 7, 18, 22, 36, 38, 42, 46, 139).

[23] Leslie M. Goldschlager. ‘ε-productions in context-free grammars’. In: *Acta Informatica* 16.3 (1981), pp. 303–308 (cited on pages 120, 121).

[24] Carlos Gómez-Rodríguez, Marco Kuhlmann, and Giorgio Satta. ‘Efficient parsing of well-nested linear context-free rewriting systems’. In: *Proceedings of the 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*. 2010, pp. 276–284 (cited on page 139).

[25] Saul Gorn. ‘Explicit definitions and linguistic dominoes’. In: *Proceedings of the Systems and Computer Science Conference*. University of Toronto Press, 1965, pp. 77–115 (cited on page 16).

[26] Sheila A. Greibach. ‘A new normal-form theorem for context-free phrase structure grammars’. In: *Journal of the ACM (JACM)* 12.1 (1965), pp. 42–52 (cited on page 36).

[27] Michael A. Harrison. *Introduction to formal language theory*. Addison-Wesley, 1978 (cited on pages 22, 69, 120).

[28] Juris Hartmanis. ‘On the Succinctness of Different Representations of Languages’. In: *SIAM Journal on Computing* 9.1 (1980), pp. 114–120 (cited on page 141).

[29] Julia Hockenmaier and Yonatan Bisk. ‘Normal-form parsing for Combinatory Categorial Grammars with generalized composition and type-raising’. In: *Proceedings of the 23rd International Conference on Computational Linguistics (COLING)*. 2010, pp. 465–473 (cited on page 142).

[30] Julia Hockenmaier and Mark Steedman. ‘Generative Models for Statistical Parsing with Combinatory Categorial Grammar’. In: *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL)*. 2002, pp. 335–342 (cited on page 126).

[31] Julia Hockenmaier and Peter Young. ‘Non-Local Scrambling: The Equivalence of TAG and CCG Revisited’. In: *Proceedings of the 9th International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+)*. 2008, pp. 41–48 (cited on page 6).

[32] Dieter Hofbauer, Maria Huber, and Gregory Kucherov. 'Some Results on Top-Context-Free Tree Languages'. In: *Proceedings of the 19th International Colloquium on Trees in Algebra and Programming*. Vol. 787. LNCS. Springer, 1994, pp. 157–171 (cited on page 22).

[33] Beryl Hoffman. 'The formal consequences of using variables in CCG categories'. In: *Proceedings of the 31st Annual Meeting of the Association for Computational Linguistics (ACL)*. 1993, pp. 298–300 (cited on page 142).

[34] Matthew Honnibal and James R. Curran. 'Fully lexicalising CCGbank with hat categories'. In: *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 2009, pp. 1212–1221 (cited on page 142).

[35] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 1979 (cited on pages 4, 14).

[36] Neil D. Jones and William T. Laaser. 'Complete problems for deterministic polynomial time'. In: *Theoretical Computer Science* 3.1 (1976), pp. 105–117 (cited on page 120).

[37] Aravind K. Joshi. 'Tree Adjoining Grammars: How Much Context-Sensitivity Is Required to Provide Reasonable Structural Descriptions?' In: *Natural Language Parsing*. Ed. by David R. Dowty, Lauri Karttunen, and Arnold M. Zwicky. Cambridge University Press, 1985, pp. 206–250 (cited on pages 1, 9, 20, 22, 23).

[38] Aravind K. Joshi and Yves Schabes. 'Tree-Adjoining Grammars'. In: *Beyond Words*. Ed. by Grzegorz Rozenberg and Arto Salomaa. Vol. 3. Handbook of Formal Languages. Springer, 1997, pp. 69–123 (cited on pages 4, 19).

[39] Laura Kallmeyer. *Parsing Beyond Context-Free Grammars*. Cognitive Technologies. Springer, 2010 (cited on pages 20–23, 121).

[40] Makoto Kanazawa. *The convergence of well-nested mildly context-sensitive grammar formalisms*. Invited talk at the 14th International Conference on Formal Grammar (FG). 2009. URL: https://makotokanazawa.ws.hosei.ac.jp/talks/fg2009_talk.pdf (cited on page 23).

[41] Makoto Kanazawa. *Formal Grammar: An Introduction. Lecture 5: Mildly Context-Sensitive Languages*. Lecture notes for the course Mathematical Linguistics. 2016. URL: https://makotokanazawa.ws.hosei.ac.jp/FormalGrammar/lecture5.pdf (cited on pages 23, 139).

[42] Makoto Kanazawa and Sylvain Salvati. 'The copying power of well-nested multiple context-free grammars'. In: *International Conference on Language and Automata Theory and Applications (LATA)*. Springer. 2010, pp. 344–355 (cited on page 23).

[43] Makoto Kanazawa and Sylvain Salvati. 'MIX Is Not a Tree-Adjoining Language'. In: *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (ACL)*. 2012, pp. 666–674 (cited on page 23).

[44] Yoshihide Kato and Shigeki Matsubara. 'A New Representation for Span-based CCG Parsing'. In: *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 2021, pp. 10579–10584 (cited on page 137).

[45] Stephan Kepser and Jim Rogers. 'The equivalence of tree adjoining grammars and monadic linear context-free tree grammars'. In: *Journal of Logic, Language and Information* 20.3 (2011), pp. 361–384 (cited on pages 8, 17, 21, 94, 139).

[46] Dan Klein and Christopher D. Manning. 'Parsing and Hypergraphs'. In: *Proceedings of the Seventh International Workshop on Parsing Technologies (IWPT)*. Tsinghua University Press, 2001, pp. 123–134 (cited on page 122).

[47] Alexander Koller and Marco Kuhlmann. 'Dependency Trees and the Strong Generative Capacity of CCG'. In: *Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics (EACL)*. 2009, pp. 460–468 (cited on pages 6, 7, 143).

[48] Nobo Komagata. 'Generative Power of CCGs with Generalized Type-Raised Categories'. In: *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics (ACL)*. 1997, pp. 513–515 (cited on page 142).

[49] Marco Kuhlmann, Alexander Koller, and Giorgio Satta. 'The Importance of Rule Restrictions in CCG'. In: *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics (ACL)*. 2010, pp. 534–543 (cited on pages 4–6, 46, 134, 137, 142).

[50] Marco Kuhlmann, Alexander Koller, and Giorgio Satta. 'Lexicalization and Generative Power in CCG'. In: *Computational Linguistics* 41.2 (2015), pp. 187–219 (cited on pages 1, 5, 6, 50, 64, 65, 94, 95, 134, 137, 138, 142).

[53] Marco Kuhlmann and Giorgio Satta. 'Tree-Adjoining Grammars are not Closed Under Strong Lexicalization'. In: *Computational Linguistics* 38.3 (2012), pp. 617–629 (cited on page 6).

[54] Marco Kuhlmann and Giorgio Satta. 'A New Parsing Algorithm for Combinatory Categorial Grammar'. In: *Transactions of the Association for Computational Linguistics (TACL)* 2 (2014), pp. 405–418 (cited on pages 9, 98, 99, 133).

[55] Marco Kuhlmann, Giorgio Satta, and Peter Jonsson. 'On the Complexity of CCG Parsing'. In: *Computational Linguistics* 44.3 (2018), pp. 447–482 (cited on pages 8–10, 138–141).

[56] Tom Kwiatkowksi, Luke S. Zettlemoyer, Sharon Goldwater, and Mark Steedman. 'Inducing probabilistic CCG grammars from logical form with higher-order unification'. In: *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 2010, pp. 1223–1233 (cited on page 1).

[57] Tom Kwiatkowski, Sharon Goldwater, Luke S. Zettlemoyer, and Mark Steedman. 'A probabilistic model of syntactic and semantic acquisition from child-directed utterances and their meanings'. In: *Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics (EACL)*. 2012, pp. 234–244 (cited on page 1).

[58] Joachim Lambek. 'The mathematics of sentence structure'. In: *American Mathematical Monthly* 65.3 (1958), pp. 154–170 (cited on page 7).

[59] Kenton Lee, Mike Lewis, and Luke S. Zettlemoyer. 'Global neural CCG parsing with optimality guarantees'. In: *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 2016, pp. 2366–2376 (cited on page 1).

[60] Mike Lewis and Mark Steedman. 'Unsupervised induction of cross-lingual semantic relations'. In: *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 2013, pp. 681–692 (cited on page 1).

[61] Andreas Maletti and Joost Engelfriet. 'Strong lexicalization of tree adjoining grammars'. In: *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (ACL)*. Ed. by Haizhou Li, Chin-Yew Lin, Miles Osborne, Gary Geunbae Lee, and Jong C. Park. 2012, pp. 506–515 (cited on page 69).

[64] Christopher Manning and Hinrich Schütze. *Foundations of statistical natural language processing*. MIT press, 1999 (cited on pages 5, 6).

[65] Shunichi Matsubara and Takumi Kasai. 'A characterization of TALs by the generalized Dyck language'. In: *IEICE Transactions on Information and Systems (Japanese Edition)* J90-D.6 (2007), pp. 1417–1427 (cited on page 22).

[66] Uwe Mönnich. 'A logical characterization of extended TAGs'. In: *Proceedings of the 11th International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+)*. 2012, pp. 37–45 (cited on page 22).

[67] Richard Moot and Christian Retoré. *The logic of categorial grammars: A deductive account of natural language syntax and semantics*. Vol. 6850. LNCS. Springer, 2012 (cited on page 36).

[68] Frank Morawietz and Uwe Mönnich. 'A model-theoretic description of tree adjoining grammars'. In: *Electronic Notes in Theoretical Computer Science* 53 (2004), pp. 210–232 (cited on page 22).

[69] Johannes Osterholzer. 'New Results on Context-Free Tree Languages'. PhD thesis. Dresden University of Technology, 2018 (cited on page 22).

[70] Johannes Osterholzer, Toni Dietze, and Luisa Herrmann. 'Linear context-free tree languages and inverse homomorphisms'. In: *Information and Computation* 269 (2019). Special Issue on the 10th International Conference on Language and Automata Theory and Applications (LATA), p. 104454 (cited on page 22).

[71] James Rogers. 'wMSO theories as grammar formalisms'. In: *Theoretical Computer Science* 293.2 (2003), pp. 291–320 (cited on page 22).

[72] William C. Rounds. 'Tree-Oriented Proofs of Some Theorems on Context-Free and Indexed Languages'. In: *Proceedings of the Second Annual ACM Symposium on Theory of Computing (STOC)*. Association for Computing Machinery, 1970, 109–116 (cited on pages 7, 17, 22).

[73] Sylvain Salvati. 'MIX is a 2-MCFL and the word problem in Z2 is captured by the IO and the OI hierarchies'. In: *Journal of Computer and System Sciences* 81.7 (2015), pp. 1252–1277 (cited on page 23).

[74] Yves Schabes. 'Mathematical and computational aspects of lexicalized grammars'. PhD thesis. University of Pennsylvania, 1990 (cited on pages 9, 141).

[75] Yves Schabes, Anne Abeillé, and Aravind K. Joshi. 'Parsing Strategies with 'Lexicalized' Grammars: Application to Tree Adjoining Grammars'. In: *Proceedings of the 12th International Conference on Computational Linguistics (COLING)*. 1988, pp. 578–583 (cited on page 45).

[78] Hiroyuki Seki, Takashi Matsumura, Mamoru Fujii, and Tadao Kasami. 'On multiple context-free grammars'. In: *Theoretical Computer Science* 88.2 (1991), pp. 191–229 (cited on page 23).

[79] Stuart M. Shieber. 'Evidence Against the Context-Freeness of Natural Language'. In: *Linguistics and Philosophy* 8.3 (1985), pp. 333–343 (cited on page 1).

[80] Stuart M. Shieber, Yves Schabes, and Fernando Pereira. 'Principles and Implementation of Deductive Parsing'. In: *Journal of Logic Programming* 24.1–2 (1995), pp. 3–36 (cited on page 103).

[81] Michael Sipser. *Introduction to the Theory of Computation*. Second Edition. Thomson Course Technology, 2006 (cited on page 64).

[82] Miloš Stanojević and Mark Steedman. 'CCG Parsing Algorithm with Incremental Tree Rotation'. In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*. 2019, pp. 228–239 (cited on page 137).

[83] Miloš Stanojević and Mark Steedman. 'Formal Basis of a Language Universal'. In: *Computational Linguistics* 47.1 (2021), pp. 9–42 (cited on pages 6, 7, 134, 142, 143).

[84] Mark Steedman. 'Dependency and coördination in the grammar of Dutch and English'. In: *Language* 61.3 (1985), pp. 523–568 (cited on pages 5, 20).

[85] Mark Steedman. *A very short introduction to CCG*. Unpublished manuscript. 1996. URL: https://www.inf.ed.ac.uk/teaching/courses/nlg/readings/ccgintro.pdf (cited on page 97).

[86] Mark Steedman. *The Syntactic Process*. MIT Press, 2000 (cited on pages 1–3, 5–7, 10, 29, 138, 141, 142).

[87] Mark Steedman. *Taking scope: The natural semantics of quantifiers*. MIT Press, 2011 (cited on pages 22, 97).

[88] Mark Steedman and Jason Baldridge. 'Combinatory Categorial Grammar'. In: *Non-Transformational Syntax: Formal and Explicit Models of Grammar*. Ed. by Robert D. Borsley and Kersti Börjars. Blackwell, 2011. Chap. 5, pp. 181–224 (cited on pages 1, 134).

[89] Hans-Jörg Tiede. 'Deductive Systems and Grammars: Proofs as Grammatical Structures'. PhD thesis. Bloomington, IN, USA: Indiana University, 1999 (cited on pages 7, 36, 46).

[90] Krishnamurti Vijay-Shanker. 'A study of tree adjoining grammars'. PhD thesis. University of Pennsylvania, 1988 (cited on pages 4, 22, 23).

[91] Krishnamurti Vijay-Shanker and David J. Weir. 'Polynomial time parsing of combinatory categorial grammars'. In: *Proceedings of the 28th Annual Meeting of the Association for Computational Linguistics (ACL)*. 1990, pp. 1–8 (cited on pages 1, 9, 133, 137).

[92] Krishnamurti Vijay-Shanker and David J. Weir. 'Parsing some constrained grammar formalisms'. In: *Computational Linguistics* 19.4 (1993), pp. 591–636 (cited on pages 1, 9, 133).

[93] Krishnamurti Vijay-Shanker and David J. Weir. 'The Equivalence of Four Extensions of Context-Free Grammars'. In: *Mathematical Systems Theory* 27.6 (1994), pp. 511–546 (cited on pages 1, 4–7, 10, 21, 25, 28, 30, 33–35, 94, 140, 142).

[94] Krishnamurti Vijay-Shanker, David J. Weir, and Aravind Joshi. 'Characterizing structural descriptions produced by various grammatical formalisms'. In: *25th Annual Meeting of the Association for Computational Linguistics (ACL)*. 1987, pp. 104–111 (cited on page 23).

[95] David J. Weir. 'Characterizing mildly context-sensitive grammar formalisms'. PhD thesis. University of Pennsylvania, 1988 (cited on pages 21–23, 94).

[96] David J. Weir. 'A geometric hierarchy beyond context-free languages'. In: *Theoretical Computer Science* 104.2 (1992), pp. 235–261 (cited on page 138).

[97] David J. Weir and Aravind K. Joshi. '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)*. 1988, pp. 278–285 (cited on pages 5–8, 138).

[98] Luke S. Zettlemoyer and Michael Collins. 'Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars'. In: *Proceedings of the 21st Conference on Uncertainty in Artificial Intelligence (UAI)*. 2005, pp. 658–666 (cited on page 1).

[99] Luke S. Zettlemoyer and Michael Collins. 'Online learning of relaxed CCG grammars for parsing to logical form'. In: *Proceedings of the 2007 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 2007, pp. 678–687 (cited on page 1).

Bibliographische Daten

Begutachtete Veröffentlichungen

- Marco Kuhlmann, Andreas Maletti, and Lena K. Schiffer. 'The Tree-Generative Capacity of Combinatory Categorial Grammars'. In: *Proceedings of the 39th Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*. Ed. by A. Chattopadhyay and P. Gastin. Vol. 150. LIPIcs. Schloss Dagstuhl — Leibniz-Zentrum für Informatik, 2019

- Lena K. Schiffer and Andreas Maletti. 'Strong Equivalence of TAG and CCG'. In: *Transactions of the Association for Computational Linguistics (TACL)* 9 (2021)

- Marco Kuhlmann, Andreas Maletti, and Lena K. Schiffer. 'The Tree-Generative Capacity of Combinatory Categorial Grammars'. In: *Journal of Computer and System Sciences* 124 (2022). Extended version.

- Lena K. Schiffer, Marco Kuhlmann, and Giorgio Satta. 'Tractable Parsing for CCGs of Bounded Degree'. In: *Comput. Linguist.* 48.3 (2022)

- Andreas Maletti and Lena K. Schiffer. 'Combinatory Categorial Grammars as Acceptors of Weighted Forests'. In: *Information and Computation* 294 (2023). Not part of this dissertation.

Unveröffentlichte Manuskripte

- Andreas Maletti and Lena K. Schiffer. 'Strong Equivalence of TAG and CCG'. Extended version. Unpublished manuscript. 2022. arXiv: `2205.07743 [cs.FL]`

Selbständigkeitserklärung

Hiermit erkläre ich, die vorliegende Dissertation selbständig und ohne unzulässige fremde Hilfe angefertigt zu haben. Ich habe keine anderen als die angeführten Quellen und Hilfsmittel benutzt und sämtliche Textstellen, die wörtlich oder sinngemäß aus veröffentlichten oder unveröffentlichten Schriften entnommen wurden, und alle Angaben, die auf mündlichen Auskünften beruhen, als solche kenntlich gemacht. Ebenfalls sind alle von anderen Personen bereitgestellten Materialen oder erbrachten Dienstleistungen als solche gekennzeichnet.

....
(Ort, Datum)

....
(Lena Katharina Schiffer)