Strong Equivalence of TAG and CCG

Lena Katharina Schiffer and Andreas Maletti

Faculty of Mathematics and Computer Science, Universität Leipzig, Germany
P.O. Box 100 920, D-04009 Leipzig, Germany
{schiffer,maletti}@informatik.uni-leipzig.de

Abstract

Tree-adjoining grammar (TAG) and combinatory categorial grammar (CCG) are two well-established mildly context-sensitive grammar formalisms that are known to have the same expressive power on strings (i.e., generate the same class of string languages). It is demonstrated that their expressive power on trees also essentially coincides. In fact, CCG without lexicon entries for the empty string and only first-order rules of degree at most 2 are sufficient for its full expressive power.

1 Introduction

Combinatory categorial grammar (CCG) (Steedman, 2000; Steedman and Baldridge, 2011) is one of several grammar formalisms that were introduced as an extension of context-free grammars. In particular, CCG extends the classical categorial grammar (Bar-Hillel et al., 1960), which has the same expressivity as context-free grammar, by rules that are inspired by combinatory logic (Curry et al., 1958). CCG is a mildly context-sensitive grammar formalism (Joshi, 1985). Context-sensitive grammar formalisms are formalisms that are efficiently parsable (i.e., in polynomial time) and have expressivity beyond the context-free languages. They are able to express a limited amount of cross-serial dependencies and have the constant growth property. Because of these features and its notion of syntactic categories, which is quite intuitive for natural languages, CCG has become widely applied in compuational linguistics (Steedman, 2000). Further, it can be enhanced by semantics through the lambda calculus.

CCG is based on a lexicon and a rule system. The lexicon assigns syntactic categories to the symbols of an input string and the rule system describes how neighboring categories can be combined to new categories. Each category has a target, which is similar to the return type of a function, and optionally, a number of arguments. Different from functions, each argument has a directionality that indicates if it is expected on the left or the right side. If repeated combination of categories leads to a (binary) derivation tree that comprises all input symbols and is rooted in an initial category, then the input string is accepted.

When defining CCG, there are many degrees of freedom yielding a number of different variants (Steedman, 2000; Baldridge, 2002; Steedman and Baldridge, 2011; Kuhlmann et al., 2015). This is a consequence of the linguistically motivated need to easily express specific structures that have been identified in a particular theory of syntax for a given natural language. However, we and others (Kuhlmann et al., 2015) are interested in the expressive power of CCGs as generators of formal languages, since this allows us to disentangle the confusion of subtly different formalisms and identify the principal structures expressible by a common core of the formalisms. As linguistic structure calls for a representation that goes beyond strings, we aim for a characterization of expressive power in terms of the generated trees.

The most famous result on the expressive power of CCG is by Vijay-Shanker and Weir (1994), showing that tree-adjoining grammar (TAG), linear-indexed grammar (LIG), head grammar (HG), and CCG generate the same string languages. An equivalent automaton model is the embedded push-down automaton (Vijay-Shanker, 1988). In the definition of CCG used by Vijay-Shanker and Weir (1994), the lexicon allows ε-entries, which assign syntactic categories to the empty string ε. Their rule system restricts rules to specific categories and limits the rule degree. CCG with unbounded rule degree are Turing-complete (Kuhlmann et al., 2018). Prefix-closed CCG without target restrictions, in which the rules obey special closure properties, are less powerful.

Transactions of the Association for Computational Linguistics, vol. 9, pp. 707–720, 2021. https://doi.org/10.1162/tacl_a_00393
Action Editor: Mark Steedman. Submission batch: 9/2020; Revision batch: 12/2020; Published 8/2021.

This even holds for multimodal CCGs (Kuhlmann et al., 2010, 2015), which allow many types of directionality indicators (i.e., slashes).

When going beyond the level of string languages, there exist different notions of strong generative power. We consider two formalisms as strongly equivalent if their generated derivation tree languages coincide modulo relabelings. For example, the well-known local and regular tree grammars (Gécseg and Steinby, 1997) are strongly equivalent. On the other hand, Hockenmaier and Young (2008) regard two formalisms as strongly equivalent if they capture the same sets of dependencies. Then there exist specific scrambling cases whose dependencies can be expressed by their CCG, but not by Lexicalized TAG (LTAG). Their CCG are syntactically more expressive than ours and allow type-raising, whereas the strong generative capacity (in our sense) of LTAG is strictly smaller than that of TAG (Kuhlmann and Satta, 2012). The dependencies expressed by CCG without rule restrictions and TAG are shown to be incomparable by Koller and Kuhlmann (2009).

Returning to our notion of strong generative capacity, Kuhlmann et al. (2019) investigated the tree-generative capacity of CCG without ε-entries. The generated trees are always binary. CCG with application and first-degree composition rules generate exactly the regular tree languages (Gécseg and Steinby, 1997). Without the composition rules, only a proper subset can be generated. The languages of CCG rule trees (i.e., trees labeled by applied rules instead of categories) with bounded rule degree can also be generated by simple monadic context-free tree grammar (sCFTG).

For the converse direction, we show that the tree languages generated by sCFTG can also be generated by CCG, which shows strong equivalence. This answers several open questions. Since sCFTG and TAG are strongly equivalent (Kepser and Rogers, 2011), our result also shows strong equivalence of CCG and TAG. In contrast to the construction of Vijay-Shanker and Weir (1994), which relies heavily on ε-entries, our construction avoids them and shows that they do not increase the expressive power of CCG. Additionally, we only use rules up to degree 2 and first-order categories (i.e., arguments are atomic), which shows that larger rule degree or higher-order categories do not increase the expressive power.

Our construction proceeds roughly as follows. We begin with a spine grammar, which is a variant of sCFTG that is also strongly equivalent to TAG. We encode its spines using a context-free grammar, which in turn can be represented by a special variant of push-down automata. 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 the categories.

2 Preliminaries

The nonnegative integers are $\mathbb{N}$. For every $k \in \mathbb{N}$, we let $[k] = \{i \in \mathbb{N} \mid 1 \leq i \leq k\}$. The set Σ^* contains all strings over the finite set Σ including the empty string ε. We let $\Sigma^+ = \Sigma^* \setminus \{\varepsilon\}$. The length of $w \in \Sigma^*$ is $|w|$, and concatenation is written as juxtaposition. The *prefixes* $\mathrm{Pref}(w)$ of a string $w \in \Sigma^*$ are $\{u \in \Sigma^* \mid \exists v \in \Sigma^* \colon w = uv\}$. A *string language* is a subset $L \subseteq \Sigma^*$. Given a relation $\Rightarrow \subseteq S^2$, we let $\Rightarrow^*$ be the reflexive, transitive closure of $\Rightarrow$.

2.1 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* $\Sigma = \Sigma_0 \cup \Sigma_1 \cup \Sigma_2$. 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. 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 $a \in \Sigma_0$. A *tree language* is a subset $T \subseteq T_{\Sigma_2, \emptyset}(\Sigma_0)$. 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 (Gorn, 1965) to a tree, where $\mathcal{P}_+(S)$ is the set of all nonempty subsets of S. Let

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

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 $\mathrm{leaves}(t) = \big\{w \in \mathrm{pos}(t) \mid w1 \notin \mathrm{pos}(t)\big\}$. Given a tree $t \in T$ and a position $w \in \mathrm{pos}(t)$, we write $t|_w$ and $t(w)$ to denote the subtree rooted in w and the symbol at w, respectively. Additionally, we let $t[t']_w$ be the tree obtained when replacing the subtree appearing in t at position w by $t' \in T$. Finally, let $\mathrm{yield} \colon T \to \Sigma_0^+$ be defined by $\mathrm{yield}(a) = a$ for all $a \in \Sigma_0$ and $\mathrm{yield}\big(c(t_1, \ldots, t_k)\big) = \mathrm{yield}(t_1) \cdots \mathit{yield}(t_k)$ for all $k \in [2]$, $c \in \Sigma_k$, 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 $\square$ occurs exactly once. We write $\mathrm{pos}_{\square}(C)$ to denote the unique position of $\square$ in the context $C \in C_{\Sigma_2,\Sigma_1}(\Sigma_0)$. Moreover, given $t \in T$ we simply write $C[t]$ instead of $C[t]_{\mathrm{pos}_{\square}(C)}$.

A tuple (ρ_0, ρ_1, ρ_2) is called a *relabeling* if $\rho_k \colon \Sigma_k \to \Delta_k$ for all $k \in \{0,1,2\}$ and ranked set Δ. It induces the map $\rho \colon T \to T_{\Delta_2,\Delta_1}(\Delta_0)$ given by $\rho\big(c(t_1,\dots,t_k)\big) = \big(\rho_k(c)\big)\big(\rho(t_1),\dots,\rho(t_k)\big)$ for all $k \in \{0,1,2\}$, $c \in \Sigma_k$ and $t_1,\dots,t_k \in T$.

2.2 Combinatory Categorial Grammar

In the following, we give a short introduction to CCG. Given an alphabet A of *atoms* or *atomic categories* and a set of *slashes* $D = \{/, \backslash\}$ indicating directionality, the set of *categories* is defined as $\mathcal{C}(A) = T_{D,\emptyset}(A)$. We usually write the categories in infix notation and the slashes are left-associative by convention, so each category takes the form $c = a \,|_1\, c_1 \cdots |_k\, c_k$ where $a \in A$, $|_i \in D$, $c_i \in \mathcal{C}(A)$ for all $i \in \{1,\dots,k\}$. The atom a is called the *target* of c and written as $\mathrm{tar}(c)$. The slash-category pairs $|_i\, c_i$ are called *arguments* and their number k is called the *arity* of c and denoted by $\mathrm{ar}(c)$. In addition, we write $\mathrm{arg}(c,i)$ to get the i-th argument $|_i\, c_i$ of c. In the sense of trees, the sequence of arguments is a context $\square \,|_1\, c_1 \cdots |_k\, c_k$. The set of *argument contexts* is denoted by $\mathcal{A}(A) \subseteq C_{D,\emptyset}(A)$. We distinguish between two types of categories. In *first-order categories*, all arguments are atomic, whereas in *higher-order categories*, the arguments can have arguments themselves.

Next, we describe how 2 neighboring categories can be combined. Intuitively, the direction of the slash determines on which side a category matching the argument is expected. Hence there are 2 types of rules. Despite the conventions for inference systems, we put the inputs (premises) below and the output (conclusion) above to make the shape of the proof tree apparent. A *rule of degree* k with $k \in \mathbb{N}$ has one of the following forms:

$$\frac{ax \,|_1\, c_1 \cdots |_k\, c_k}{ax/c \qquad c \,|_1\, c_1 \cdots |_k\, c_k} \quad \text{(forward rule)}$$

$$\frac{ax \,|_1\, c_1 \cdots |_k\, c_k}{c \,|_1\, c_1 \cdots |_k\, c_k \qquad ax\backslash c} \quad \text{(backward rule)}$$

where $a \in A$, $c \in \mathcal{C}(A) \cup \{y\}$, $|_i \in D$, and $c_i \in \mathcal{C}(A) \cup \{y_i\}$ for all $i \in [k]$. Here, $y, y_1, \dots, y_k$ are category variables that can match any category in $\mathcal{C}(A)$ and x is an argument context variable that can match any argument context in $\mathcal{A}(A)$. The category taking the argument $(ax \,|\, c$ with $| \in D)$ is called *primary category*, the one providing it $(c \,|_1\, c_1 \cdots |_k\, c_k)$ is called *secondary category*, and they are combined to an *output category* $(ax \,|_1\, c_1 \cdots |_k\, c_k)$. Given rule r, we write $\mathrm{sec}(r)$ to refer to the secondary category. Rules of degree 0 will be referred to as *application rules*, while rules of higher degree are *composition rules*. We write $\mathcal{R}(A)$ for the set of all rules over A. A *rule system* is a pair $\Pi = (A, R)$, where A is an alphabet and $R \subseteq \mathcal{R}(A)$ is a finite set of rules over A. Given a rule $r \in R$, we obtain a *ground instance* of it by replacing the variables $\{y, y_1, \dots\}$ by concrete categories and the variable x by a concrete argument context. The ground instances of Π induce a relation $\to_\Pi \subseteq \mathcal{C}(A)^2 \times \mathcal{C}(A)$ and we write $\frac{c''}{c\ c'}\Pi$ instead of $(c, c') \to_\Pi c''$. The relation $\to_\Pi$ extends to a relation $\Rightarrow_\Pi \subset (\mathcal{C}(A)^*)^2$ on sequences of categories. It is given by

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

A *combinatory categorial grammar* (CCG) is a tuple $\mathcal{G} = (\Sigma, A, R, I, \mathcal{L})$ that consists of an alphabet Σ of *input symbols*, a rule system (A, R), a set $I \subseteq A$ of *initial categories*, and a finite relation $\mathcal{L} \subseteq \Sigma \times \mathcal{C}(A)$ called *lexicon*. It is called *k-CCG* if each rule $r \in R$ has degree at most k, where $k \in \mathbb{N}$.

The CCG $\mathcal{G}$ *generates* the category sequences $\mathcal{C}_\mathcal{G} \subseteq \mathcal{C}(A)*$ and the string language $L(\mathcal{G}) \subseteq \Sigma^*$ given by

$$\mathcal{C}_\mathcal{G} = \bigcup_{a_0 \in I} \left\{ \varphi \in \mathcal{C}(A)^* \;\middle|\; \varphi \Rightarrow^*_{(A,R)} a_0 \right\}$$

and $L(\mathcal{G}) = \mathcal{L}^{-1}(\mathcal{C}_\mathcal{G})$, where the string language $L(\mathcal{G})$ contains all strings that can be relabeled via the lexicon to a category sequence in $\mathcal{C}_\mathcal{G}$. A tree $t \in T_{\mathcal{C}(A),\emptyset}(\mathcal{L}(\Sigma))$ is called *derivation tree of* $\mathcal{G}$ if $\frac{t(w)}{t(w\cdot 1)\quad t(w\cdot 2)}(A, R)$ for every $w \in \mathrm{pos}(t) \setminus \mathrm{leaves}(t)$. We denote the set of all derivation trees of $\mathcal{G}$ by $\mathcal{D}(\mathcal{G})$.

A *category relabeling* $\rho \colon \mathcal{C}(A) \to \Delta$ is a relabeling such that $\rho(c) = \rho(c')$ for all categories $c, c' \in \mathcal{C}(A)$ with $\mathrm{tar}(c) = \mathrm{tar}(c')$

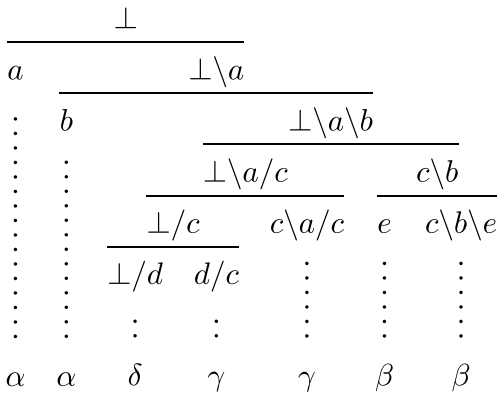


Figure 1: CCG derivation tree (see Example 2.2).

and $\arg\big(c,\, \mathrm{ar}(c)\big) = \arg\big(c',\, \mathrm{ar}(c')\big)$. The relabeled derivation trees $T_\rho(\mathcal{G}) \subseteq T_{\Delta_2,\emptyset}(\Delta_0)$ are given by

$$T_\rho(\mathcal{G}) = \big\{ \rho(t) \,\big|\, t \in \mathcal{D}(\mathcal{G}),\, t(\varepsilon) \in I \big\}$$

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

Example 1. Let $\mathcal{G} = (\Sigma, A, \mathcal{R}(A,2), \{\bot\}, \mathcal{L})$ with $\Sigma = \{\alpha, \beta, \gamma, \delta\}$ and $A = \{\bot, a, b, c, d, e\}$ be a CCG with the lexicon $\mathcal{L}$ given below, where $\mathcal{R}(A,2)$ is the set of all rules over A up to degree 2. Thus, it is a 2-CCG.

$$\begin{aligned}
\mathcal{L}(\alpha) &= \{a,\, b\} \\
\mathcal{L}(\beta) &= \{c\backslash b,\, c\backslash b\backslash e,\, e,\, e/e\} \\
\mathcal{L}(\gamma) &= \{d/c,\, c\backslash a/c\} \\
\mathcal{L}(\delta) &= \{\bot/d\}
\end{aligned}$$

$\frac{\bot x\backslash a/c}{\bot x/c \quad c\backslash a/c}$ is a forward rule of degree 2 in $\mathcal{R}(A,2)$, where x is an argument context and can thus be replaced by an arbitrary sequence of arguments. Utilizing $x = \Box\backslash a$ yields the ground instance $\frac{\bot\backslash a/a/c}{\bot\backslash a/c \quad c\backslash a/c}$, which has primary category $c_1 = \bot\backslash a/c$ and secondary category $c_2 = c\backslash a/c$. The latter has target $\mathrm{tar}(c_2) = c$ and the two arguments $\backslash a$ and $/c$, so its arity is $\mathrm{ar}(c_2) = 2$.

A derivation tree of $\mathcal{G}$ is depicted in Figure 1. We start at the bottom with categories taken from the lexicon in accordance with the input symbols. Then neighboring categories are combined until we arrive at the root with initial category $\bot$, so the input word is accepted. ■

3 Push-down Automata

We start by introducing a Moore variant of push-down automata (Autebert et al., 1997) 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 et al. (2013). Instead of processing input symbols as part of transitions (as in Mealy machines), Moore machines output a unique input symbol in each state (Fleischner, 1977). For every set Γ, we let $\Gamma^{\leq 1} = \{\varepsilon\} \cup \Gamma$ and $\Gamma^{\geq 2} = \{w \in \Gamma^* \mid 2 \leq |w|\}$ be the sets of strings over Γ of length at most 1 and at least 2, respectively.

Definition 2. A *Moore push-down automaton* (MPDA) $\mathcal{A} = (Q, \Sigma, \Gamma, \delta, \tau, I, F)$ 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, in a single step we can either push or pop a single stack symbol or ignore the stack. Note that we explicitly exclude the case where a symbol is popped and another symbol is pushed at the same time. In the following, let $\mathcal{A} = (Q, \Sigma, \Gamma, \delta, \tau, I, F)$ be an MPDA. On the set $\mathrm{Conf}_\mathcal{A} = Q \times \Gamma^*$ of configurations of $\mathcal{A}$ the *move relation* $\vdash_\mathcal{A} \subseteq \mathrm{Conf}_\mathcal{A}^2$ is

$$\vdash_\mathcal{A} = \bigcup_{(q,\gamma,\gamma',q') \in \delta} \Big\{ \big(\langle q, \gamma\alpha \rangle,\, \langle q', \gamma'\alpha \rangle \big) \,\Big|\, \alpha \in \Gamma^+ \Big\}$$

and a configuration $\langle q, \alpha \rangle \in \mathrm{Conf}_\mathcal{A}$ is *initial* (respectively, *final*) if $q \in I$ and $\alpha \in \Gamma$ (respectively, $q \in F$ and $\alpha = \varepsilon$). An *accepting run* is a sequence $\xi_0, \ldots, \xi_n \in \mathrm{Conf}_\mathcal{A}$ of configurations that are successively related by moves (i.e., $\xi_{i-1} \vdash_\mathcal{A} \xi_i$ for all $i \in [n]$), starts with an initial configuration ξ_0, and finishes in a final configuration ξ_n. In other words, we can start in an initial state with an arbitrary symbol on the stack and finish in a final state with the empty stack, and for each intermediate step there has to exist a transition. The language *accepted* by $\mathcal{A}$ contains exactly those strings $w \in \Sigma^*$, for which there exists an accepting run $\langle q_0, \alpha_0 \rangle, \ldots, \langle q_n, \alpha_n \rangle$ 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 MPDA are *equivalent* if they accept the same language. Since 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 3. *MPDA accept the context-free languages of strings of length at least 2.* ■

The MPDA $\mathcal{A}$ is *pop-normalized* if there exists a map $\text{pop}\colon \Gamma \to Q$ such that $q' = \text{pop}(\gamma)$ for every transition $(q, \gamma, \varepsilon, q') \in \delta$. In other words, for each stack symbol $\gamma \in \Gamma$ there is a unique state $\text{pop}(\gamma)$ 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. Popping 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 according pushing 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 7.

We can easily establish this property by storing a state in each stack symbol. Each pushing 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 4. *For every MPDA we can construct an equivalent pop-normalized MPDA.* ■

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

Lemma 5. *For every context-free language $L \subseteq \Sigma^*$ and $\triangleleft \notin \Sigma$, the language* $\text{Next}(L)$ *is context-free, where*

$$\text{Next}(L) = \bigcup_{\substack{n \in \mathbb{N},\, \sigma_1, \dots, \sigma_n \in \Sigma \\ \sigma_1, \dots, \sigma_n \in L}} \big\{ \langle \sigma_2, \sigma_1 \rangle \cdots \langle \sigma_n, \sigma_{n-1} \rangle \langle \triangleleft, \sigma_n \rangle \big\}$$

■

Corollary 6. *For every context-free language $L \subseteq \Sigma^{\geq 2}$ there exists a pop-normalized* MPDA $\mathcal{A}$ *such that* $L(\mathcal{A}) = \text{Next}(L)$. ■

4 Spine Grammars

Now we move on to representations of tree languages. We first recall context-free tree grammars (Rounds, 1969), but only the monadic simple variant (Kepser and Rogers, 2011).

Definition 7. 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 nullary *start nonterminal* $S \in N_0$, and (iii) a finite set $P \subseteq P_0 \cup P_1$ of *productions*, where $P_0 = N_0 \times T_{\Sigma_2, N_1}(N_0 \cup \Sigma_0)$ and $P_1 = N_1 \times C_{\Sigma_2, N_1}(N_0 \cup \Sigma_0)$. ■

In the following let $\mathcal{G} = (N, \Sigma, S, P)$ be an sCFTG. We write $(n, r) \in P$ simply as $n \to r$. Given $t, u \in T_{\Sigma_2, N_1}(\Sigma_0 \cup N_0)$ we let $t \Rightarrow_{\mathcal{G}} u$ if there exist $(n \to r) \in P$ and a position $w \in \text{pos}(t)$ 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 $t' \in T_{\Sigma_2, N_1}(\Sigma_0 \cup N_0)$. The *tree language $T(\mathcal{G})$ generated by $\mathcal{G}$* is

$$T(\mathcal{G}) = \{ t \in T_{\Sigma_2, \emptyset}(\Sigma_0) \mid S \Rightarrow_{\mathcal{G}}^* t \}$$

The sCFTG $\mathcal{G}'$ is *strongly equivalent* to $\mathcal{G}$ if $T(\mathcal{G}) = T(\mathcal{G}')$, and it is *weakly equivalent* to $\mathcal{G}$ if $\text{yield}\big(T(\mathcal{G})\big) = \text{yield}\big(T(\mathcal{G}')\big)$.

Spine grammars (Fujiyoshi and Kasai, 2000) are a restriction on simple monadic context-free tree grammars that remain equally expressive by Lemma 5.4 of Fujiyoshi and Kasai (2000) modulo relabelings. Let us clarify this result. Clearly, each spine grammar is itself an sCFTG and for each sCFTG $\mathcal{G}$ there exists a spine grammar $\mathcal{G}'$ and a relabeling ρ such that $T(\mathcal{G}) = \{\rho(t) \mid t \in T(\mathcal{G}')\}$. Although sCFTGs are more established, we elect to utilize spine grammars because of their essential notion of spines.

Definition 8. The sCFTG $\mathcal{G}$ is a *spine grammar* if there exists a map $d\colon \Sigma_2 \to \{1, 2\}$ such that $wi \in \text{Pref}(\text{pos}_{\square}(C))$ with $i = d(C(w))$ for every production $(n \to C) \in P$ with $n \in N_1$ and $w \in \text{Pref}(\text{pos}_{\square}(C))$ with $C(w) \in \Sigma_2$. ■

Henceforth let $\mathcal{G}$ be a spine grammar with map $d\colon \Sigma_2 \to \{1, 2\}$. Consider a production

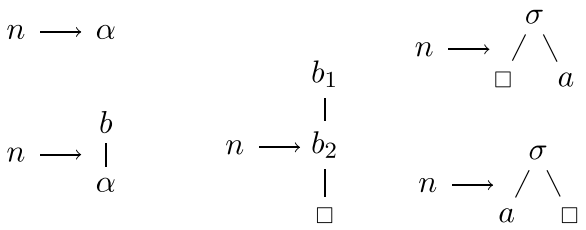
$(n \to C) \in P$ with $n \in N_1$. The *spine* of C is simply the path from the root of C to the unique occurrence $\mathrm{pos}_{\square}(C)$ of $\square$. The special feature of a spine grammar is that the symbols along the spine indicate exactly in which direction the spine continues. Since only the binary terminal symbols offer branching, the special feature of spine grammars is the existence of a map d 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 σ in all productions with spines. In the original definition of spine grammars (Fujiyoshi and Kasai, 2000, Definition 3.2), only nonterminal symbols have a spine direction. By creating copies of binary terminal symbols we can show that both variants are equivalent modulo relabelings.

Definition 9. Spine grammar $\mathcal{G}$ is in *normal form* if each $(n \to r) \in P$ is of the form (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 $a \in N_0 \setminus S$. ■

In spine grammars in normal form, the initial nonterminals are isolated and cannot occur on the right-hand sides. The 3 production types of the normal form are illustrated in Figure 2. 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

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

where $\mathcal{G}'$ is the spine grammar $\mathcal{G}$ without start productions; that is, $\mathcal{G}' = (N, \Sigma, S, P')$ with productions $P' = \{(n \to r) \in P \mid n \in N_1\}$. 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. The spine grammar $\mathcal{G}$ is *normalized* if it is in normal form and $I_{\mathcal{G}}(n) \subseteq T_{\Sigma_2,\emptyset}(\Sigma_0 \cup (N_0 \setminus \{n\}))$ for every nullary nonterminal $n \in N_0$.



(a) start (b) chain (c) terminal

Figure 2: Types of productions of spine grammars in normal form (see Definition 9).

The following result is a variant of Theorem 1 of Fujiyoshi and Kasai (2000).

Theorem 10. *For every spine grammar there is a strongly equivalent normalized spine grammar.* ■

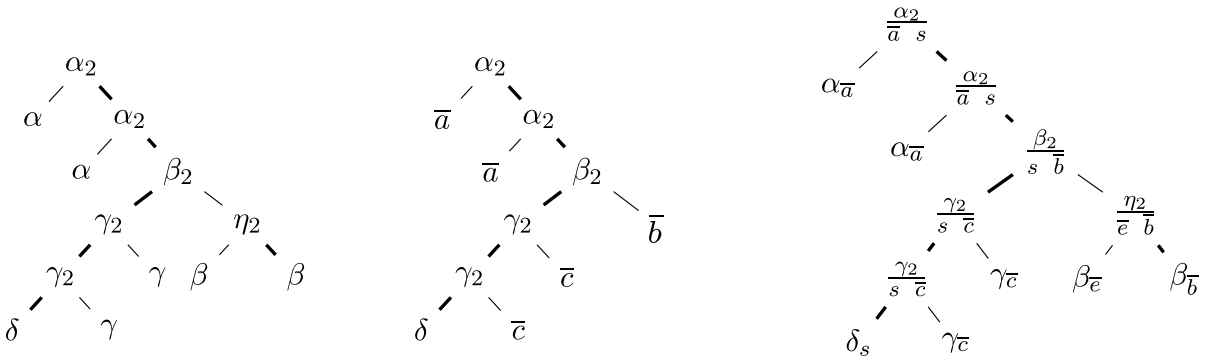
Example 11. We define the spine grammar $\mathcal{G} = (N, \Sigma, \{s\}, P)$ with $N_1 = \{t, a, b, c, b', e\}$, $N_0 = \{s, \overline{a}, \overline{b}, \overline{c}, \overline{e}\}$, $\Sigma_2 = \{\alpha_2, \beta_2, \gamma_2, \eta_2\}$, $\Sigma_0 = \{\alpha, \beta, \gamma, \delta\}$, and P as shown below.

$s \to t(\delta)$	$\overline{b} \to e(\beta)$	
$\overline{a} \to \alpha$	$t \to a\big(b'(\square)\big)$	$a \to \alpha_2(\overline{a}, \square)$
$\overline{b} \to \beta$	$b' \to b\big(c(\square)\big)$	$b \to \beta_2(\square, \overline{b})$
$\overline{c} \to \gamma$	$b \to a\big(b'(\square)\big)$	$c \to \gamma_2(\square, \overline{c})$
$\overline{e} \to \beta$	$e \to e\big(e(\square)\big)$	$e \to \eta_2(\overline{e}, \square)$

The tree in Figure 3a, in which the spines are marked by thick edges, is generated by $\mathcal{G}$. The spinal tree corresponding to the main spine of the depicted tree is shown in Figure 3b. The yield of $T(\mathcal{G})$ is $\{\alpha^n\, \delta\, \gamma^n\, \beta^m \mid n, m \geq 1\}$. ■

5 Tree-adjoining Grammars

Before we proceed we will briefly introduce TAG and sketch how a spine grammar is obtained from it. 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



(a) Generated tree (b) Spinal tree (c) Reassembled tree

Figure 3: Tree generated by spine grammar $\mathcal{G}$, a spinal tree in $I_{\mathcal{G}}(s)$ (see Example 11), and a tree in $\mathcal{F}(\mathcal{S}(\mathcal{G}))_S$ reassembled from spines (see Example 16).

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 4 shows the elementary trees of an example TAG. 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.

We briefly sketch the transformation from TAG to sCFTG by Kepser and Rogers (2011). TAG is a notational variant of footed simple CFTG, in which 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 (2000). The spine grammar of Example 11 is strongly equivalent to the TAG shown in Figure 4.

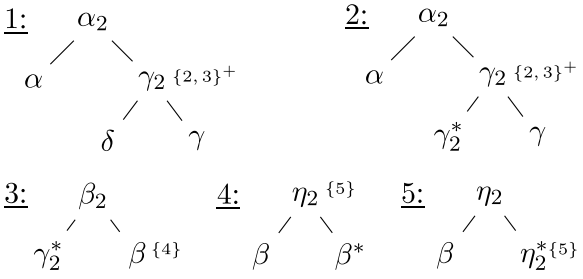


Figure 4: Tree-adjoining grammar.

6 Decomposition into Spines

We proceed with the construction starting from the normalized spine grammar $\mathcal{G}$. First, we will construct a context-free grammar (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 a 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 12. Let $\mathcal{G}$ be normalized and $\top \notin N$. The spines $\mathcal{S}(\mathcal{G}) = L(\mathcal{G}')$ 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

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

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

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

■

Example 13. We list some corresponding productions of the spine grammar $\mathcal{G}$ (left) of Example 11 and the CFG $\mathcal{G}'$ (right) for its spines $\mathcal{S}(\mathcal{G})$.

$$\begin{aligned}
&\overline{a} \to \alpha && : \quad \top \to \alpha_{\overline{a}} \\
&s \to t(\delta) && : \quad \top \to \delta_s t_s \\
&t \to a\big(b'(\square)\big) && : \quad t_s \to b'_s a_s \qquad t_{\overline{b}} \to b'_{\overline{b}}\, a_{\overline{b}} \qquad \dots \\
&a \to \alpha_2(\overline{a}, \square) && : \quad a_s \to \frac{\alpha_2}{\overline{a} \;\; s} \qquad a_{\overline{b}} \to \frac{\alpha_2}{\overline{a} \;\; \overline{b}} \qquad \dots
\end{aligned}$$

Note that for each start production we obtain a single production since the nonterminal on the left 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 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 language generated by $\mathcal{G}'$ is

$$S(\mathcal{G}) = \left\{ \delta_s \, \frac{\gamma_2}{s \;\; \overline{c}}^{\,n} \, \frac{\beta_2}{s \;\; \overline{b}} \, \frac{\alpha_2}{\overline{a} \;\; s}^{\,n} \;\middle|\; n \geq 1 \right\} \cup \left\{ \beta_{\overline{b}} \, \frac{\eta_2}{\overline{e} \;\; \overline{b}}^{\,m} \;\middle|\; m \geq 0 \right\} \cup \left\{ \alpha_a, \beta_e, \gamma_c \right\}$$

■

Note that each string generated by the CFG belongs to $(\Sigma_0 \times N)(\Sigma_2 \times N^2)^*$. 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 14. Let $T \subseteq T_{\Sigma_2 \times N, \emptyset}(\Sigma_0 \times N)$ and $w \in A$ with $A = (\Sigma_0 \times N)(\Sigma_2 \times N^2)^*$. 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

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

For $n \in N$, let $T_n = \big\{ t \in T \;\big|\; \mathrm{gen}\big(t(\varepsilon)\big) = n \big\}$ be those trees of T whose root label has n annotated in spinal direction. We define the tree language $\mathrm{att}_T(w) \subseteq T_{\Sigma_2 \times N, \emptyset}(\Sigma_0 \times N)$ recursively by $\mathrm{att}_T(\alpha_n) = \{\alpha_n\}$ for all $\alpha_n \in \Sigma_0 \times N$, and

$$\mathrm{att}_T\big(w \, \frac{\sigma}{n_1 \;\; n_2}\big) = \left\{ \frac{\sigma}{n_1 \;\; n_2}(t_1, t_2) \;\middle|\; \begin{array}{l} t_{d(\sigma)} \in \mathrm{att}_T(w) \\ t_{3-d(\sigma)} \in T_{n_{3-d(\sigma)}} \end{array} \right\}$$

for all $w \in A$ 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 15. Let $L \subseteq (\Sigma_0 \times N)(\Sigma_2 \times N^2)^*$. We inductively define the tree language $\mathcal{F}(L)$ generated by L to be the smallest tree language $\mathcal{F}$ such that $att_{\mathcal{F}}(w) \subseteq \mathcal{F}$ for every $w \in L$. ■

Example 16. The CFG $\mathcal{G}'$ of Example 13 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 3c shows a tree of $\mathcal{F}(\mathcal{S}(\mathcal{G}))_S$ since the generator of the main spine is $S = s$, which is stored in spinal direction in the root label $\frac{\alpha_2}{\overline{a} \;\; s}$. 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 $T(\mathcal{G})$ coincide modulo relabeling. This shows that the context-free language $\mathcal{S}(\mathcal{G})$ of spines completely describes the tree language $T(\mathcal{G})$ generated by $\mathcal{G}$.

Theorem 17. *Let $\mathcal{G}$ be normalized. Then $\pi\big(\mathcal{F}(\mathcal{S}(\mathcal{G}))_S\big) = T(\mathcal{G})$, where the relabeling $\pi\colon (\Sigma_0 \times N) \cup (\Sigma_2 \times N^2) \to \Sigma_0 \cup \Sigma_2$ is given by $\pi(\alpha_n) = \alpha$ and $\pi(\frac{\sigma}{n_1 \;\; n_2}) = \sigma$ for all $\alpha \in \Sigma_0$, $\sigma \in \Sigma_2$, and $n, n_1, n_2 \in N$.* ■

Corollary 18. *There exists a pop-normalized MPDA $\mathcal{A}$ such that $L(\mathcal{A}) \cup L_1 = \mathrm{Next}\big(\mathcal{S}(\mathcal{G})\big)$, where $L_1 = \big\{ w \in \mathrm{Next}\big(\mathcal{S}(\mathcal{G})\big) \mid |w| = 1 \big\}$. Moreover, $\mathcal{F}\big(L(\mathcal{A}) \cup L_1\big)_S$ and $T(\mathcal{G})$ coincide modulo relabeling.* ■

Example 19. The MPDA constructed in Corollary 18 for the spine grammar $\mathcal{G}$ of Example 11 is depicted in Figure 5. Initial states are indicated using a start marker and final states are marked by a double circle. Pushing and popping stack operations are written with downwards and upwards arrows, respectively. The MPDA consists of two components. The bigger one describes the main spine, and the smaller one

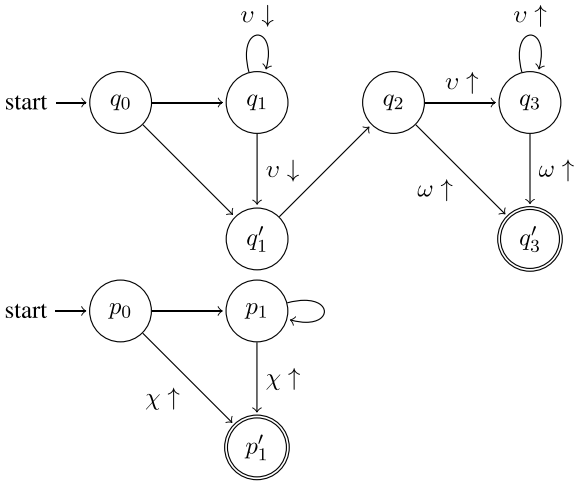


Figure 5: Sample MPDA (see Example 6).

describes the side spine. The distinction between the three stack symbols is necessary due to pop-normalization. The distinction between q_1 and q_1' (and similar states) is necessary because their previous action distinguishes their produced input symbol since we recognize $\mathrm{Next}\big(\mathcal{S}(\mathcal{G}_2)\big)$. 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 $\tau\big(p_1'\big) = (\triangleleft, z)$ where $z = \frac{\eta_2}{\bar{e}\ \ \bar{b}}$. To completely capture the behavior of $\mathcal{G}$, we additionally require the set $L_1 = \{(\triangleleft, \alpha_{\bar{a}}), (\triangleleft, \beta_{\bar{b}}), (\triangleleft, \beta_{\bar{e}}), (\triangleleft, \gamma_{\bar{c}})\}$, which contains the spines of length 1. ■

7 Constructing the CCG

In this section, let $\mathcal{G} = (N, \Sigma, S, P)$ be a normalized spine grammar with spine direction $d\colon \Sigma \to \{1, 2\}$ and $\mathcal{A} = (Q, \Delta, \Gamma, \delta, \tau, I, F)$ the pop-normalized MPDA constructed in Corollary 18 with $pop\colon \Gamma \to Q$. We note that $\Delta = \Sigma' \times \Sigma''$ with $\Sigma' = \{\triangleleft\} \cup (\Sigma_2 \times N^2)$ as well as $\Sigma'' = (\Sigma_0 \times N) \cup (\Sigma_2 \times N^2)$. 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 $\mathrm{gen}\colon Q \to N$ is given for every state $q \in Q$ by $\mathrm{gen}(q) = \mathrm{gen}(s_2)$, 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 $L_1 = \big\{w \in \mathrm{Next}\big(\mathcal{S}(\mathcal{G})\big) \mid |w| = 1\big\}$ and $\mathrm{gen}\colon L_1 \to N$ be given by $\mathrm{gen}(w) = n$ for all $w = (\triangleleft, \alpha_n) \in L_1$. We extend $\tau\colon Q \to \Delta$ to $\tau'\colon (Q \cup L_1) \to \Delta$ by $\tau'(q) = \tau(q)$ for all $q \in Q$ and $\tau'(a) = a$ for short strings $a \in L_1$.

Recall that $D = \{/, \backslash\}$. The slash type $\mathrm{slash}\colon (Q \setminus F) \to D$ and combining nonterminal $\mathrm{comb}\colon (Q \setminus F) \cup \{\bot\} \to N$ of a state $q \in Q \setminus F$ tell whether the symbol $\tau(q)$ generated by state q occurs as the first or second child of its parent symbol and with which spine generator it is combined. Let $\tau(q) = \big(\frac{\sigma}{n_1\ n_2}, s_2\big)$ 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. In addition, $\mathrm{comb}(q) = n_{3-d(\sigma)}$ and $\mathrm{comb}(\bot) = S$.

We simulate the accepting runs of $\mathcal{A}$ in the spines consisting of primary categories of the CCG. The main idea is that the primary categories on the spine store the current configuration of $\mathcal{A}$. This is achieved by adding an additional argument for transitions that push a symbol, whereas for each popping transition, an argument is removed. The rightmost argument stores the current state in the first component and the top of the stack in the second component. The previous arguments store the preceding stack symbols in their second components and the state the automaton returns to when the stack symbol stored in the next argument is popped in the first components. 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 top of a spine (which act as secondary categories unless they belong to the main 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 stores the final state corresponding to its secondary combination partner in the third component. This 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 state stored in the last 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.

Definition 20. We define the CCG $\mathcal{G}_{\mathcal{A},L_1} = (\Delta_0, A, R, I', \mathcal{L})$ as follows:

Let $A = \{(q, \gamma, f) \in A' \mid \mathrm{gen}(f) = \mathrm{comb}(q)\}$ with $A' = (Q \cup \{\bot\}) \times \Gamma \times (F \cup L_1)$. We use a_i to refer to the i-th component of an atom $a \in A$. Additionally, let $I' = \{(\bot, \varepsilon, f) \in A \mid \mathrm{gen}(f) = S\}$.

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

$$R_1^{/} = \bigcup_{\substack{a,b,c\in A,\,|\in D\\(b_1,\varepsilon,\varepsilon,c_1)\in\delta\\b_2=c_2}} \left\{ \frac{ax \,|\, c}{\underline{ax/b} \qquad b \,|\, c} \right\} \tag{1}$$

$$R_2^{/} = \bigcup_{\substack{a,b,c,e\in A,\,|,|'\in D\\(b_1,\varepsilon,e_2,e_1)\in\delta\\b_2=c_2\\c_1=\text{pop}(e_2)}} \left\{ \frac{ax \,|\, c \,|'\, e}{\underline{ax/b} \qquad b \,|\, c \,|'\, e} \right\} \tag{2}$$

$$R_3^{/} = \bigcup_{\substack{a,b\in A\\(b_1,b_2,\varepsilon,q)\in\delta}} \left\{ \frac{ax}{\underline{ax/b} \qquad b} \right\} \tag{3}$$

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 need some notions for the lexicon. A category $c \in \mathcal{C}(A)$ is *well-formed* if $| = \text{slash}(b_1)$ and $b_1 \in Q$ for every $i \in [\text{ar}(c)]$ with $|\, b = \arg(c, i)$. Let $C_{\text{wf}} = \{c \in \mathcal{C}(A) \mid c \text{ well-formed}\}$ be the set of well-formed categories. Clearly $I' \subseteq C_{\text{wf}}$. In addition, we introduce sets $\top_{L_1}$ and $\top_{\mathcal{A}}$ of top-of-spine categories derived from the short strings of L_1 and the strings accepted by $\mathcal{A}$, respectively:

$$\top_{L_1} = \{a \in I' \mid a_3 \in L_1\} \cup \bigcup_{\substack{r\in R\\ax=\text{sec}(r)}} \{ax \in C_{\text{wf}} \mid a_3 \in L_1\}$$

$$\top_{\mathcal{A}} = \{a \in I' \mid a_3 \in F\} \cup \bigcup_{\substack{r\in R\\ax=\text{sec}(r)}} \{ax \in C_{\text{wf}} \mid a_3 \in F\}$$

Note that $\top_{L_1} \cup \top_{\mathcal{A}} \subseteq C_{\text{wf}}$. Now we can define the lexicon as follows for all $\alpha \in \Delta_0 = \Sigma' \times (\Sigma_0 \times N)$:

$$\mathcal{L}(\alpha) = \left\{ ax \;\middle|\; \begin{array}{c} ax \in \top_{L_1} \\ \tau'(a_3) = \alpha \end{array} \right\} \cup \left\{ ax \,|\, b \in C_{\text{wf}} \;\middle|\; \begin{array}{c} ax \in \top_{\mathcal{A}} \\ b_1 \in I \\ \text{gen}(a_3) = \text{gen}(b_1) \\ \text{pop}(b_2) = a_3 \\ \tau'(b_1) = \alpha \end{array} \right\}$$

■

Each atom of A consists of three components. The first component stores the current state of $\mathcal{A}$ (or the special symbol $\perp$), the second component stores the current symbol at the top of the stack, and the third component stores the final state corresponding to the combining category of the attaching side spine. With this intuition, the rule system directly implements the transitions of $\mathcal{A}$.

The lexicon assigns categories to symbols that can label leaves, so these symbols are taken from the nullary terminal symbols. The assigned categories consist of a category that appears at the top of a spine and an additional argument for the initial state of an accepting run. The spines of length 1 are translated directly to secondary categories or initial categories.

Let us make a few general observations that hold for all the categories that appear in derivation trees of $\mathcal{G}_{\mathcal{A},L_1}$: (i) All categories are well-formed. This follows from the fact 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. (ii) All primary categories $ax \,|\, b$ obey $\text{gen}(a_3) = \text{gen}(b_1)$. This is directly required by the rule system.

Finally, we will now describe how to relabel the derivation trees $\mathcal{D}(\mathcal{G}_{\mathcal{A},L_1})$ of the CCG $\mathcal{G}_{\mathcal{A},L_1}$ that uses categories built using the input symbols of the MPDA $\mathcal{A}$. Note that only well-formed categories will occur in derivation trees. Primary and non-primary categories are relabeled differently. The relabeling $\rho\colon C_{\text{wf}} \to \Delta$ is defined for every $c \in C_{\text{wf}}$ by $\rho(ax \,|\, b) = \tau'(b_1)$ for all primary categories $ax \,|\, b \in C_{\text{wf}}$; i.e., $\text{gen}(a_3) = \text{gen}(b_1)$. Otherwise $\rho(ax) = \tau'(a_3)$ for all initial and secondary categories $ax \in C_{\text{wf}}$.

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

Lemma 21. *For all secondary categories* $ax \,|\, b$ *we have* $\text{gen}(a_3) \neq \text{gen}(b_1)$. ■

We are now ready to describe the general form of primary spines of $\mathcal{G}_{\mathcal{A},L_1}$. Given a primary spine $c_0 \dots c_n$ read from lexicon entry towards the root with $n \geq 1$, we know that it starts with a lexicon entry $c_0 = ax \,|\, b \in \mathcal{L}(\Delta_0)$ 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 $ax \,|_1 b_1 \dots |_m b_m$ with $m \geq 1$. Let $b_i = (q_i, \gamma_i, f_i)$ for every $i \in [m]$. The

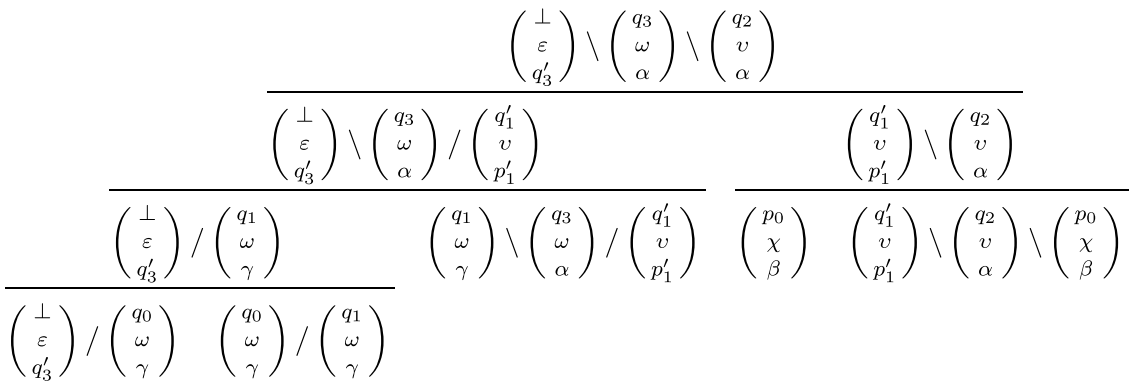


Figure 6: Part of a derivation tree of $\mathcal{G}_{\mathcal{A},L_1}$ (see Example 23).

category c_n is relabeled to $\tau'(a_3)$ and c is relabeled to $\tau'(q_m)$. Additionally, unless $a_1 = \bot$, the first components of all atoms in ax have the same spine generator $\mathrm{gen}(a_1)$ and $\mathrm{gen}(q_1) = \cdots = \mathrm{gen}(q_m)$, but $\mathrm{gen}(a_1) \neq \mathrm{gen}(q_1)$. Finally, neighboring arguments $|_{i-1}\, b_{i-1}\, |_i\, b_i$ in the suffix are coupled such that $\mathrm{pop}(\gamma_i) = q_{i-1}$ 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 $\mathcal{G}_{\mathcal{A},L_1}$ are relabeled to strings of $\mathrm{Next}\big(\mathcal{S}(\mathcal{G})\big)$ and vice versa. Additionally, spines attach in essentially the same manner in the CCG and using $\mathcal{F}$. This yields the following main theorem.

Theorem 22. *Given a spine grammar $\mathcal{G}$, we can construct a CCG $\mathcal{G}'$ that can generate $T(\mathcal{G})$.* ■

Example 23. Figure 6 shows part of the derivation tree of CCG $\mathcal{G}_{\mathcal{A},L_1}$ that corresponds to the tree of Figure 3a, which is generated by the spinal grammar $\mathcal{G}$ of Example 11. We use the following abbreviations: $\alpha = (\triangleleft, \alpha_{\overline{a}})$, $\beta = (\triangleleft, \beta_{\overline{b}})$, and $\gamma = (\triangleleft, \gamma_{\overline{c}})$. The labeling of the depicted section is $\delta\,\gamma_2\,\gamma_2\,\beta_2$ for the main spine and $\beta\,\eta_2$ for the side spine (see Figure 3a). The corresponding runs of $\mathcal{A}$ are $\big(\langle q_0, \omega\rangle, \langle q_1, \omega\rangle, \langle q_1', \upsilon\omega\rangle, \langle q_2, \upsilon\omega\rangle\big)$ and $\big(\langle p_0, \chi\rangle, \langle p_1', \varepsilon\rangle\big)$.

Let us observe how the transitions of $\mathcal{A}$ are simulated by $\mathcal{G}_{\mathcal{A},L_1}$. 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 $\backslash(q_3, \omega, \alpha)/(q_1', \upsilon, p_1')$. 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 popping transition $(p_0, \chi, \varepsilon, p_1')$ on the side spine run is realized by removing $/(p_0, \chi, \beta)$.

The third components are required to relabel the non-primary categories. At the bottom 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 top of a spine as an initial category with $q_3' \in F$ in its third component, while the appended (q_0, ω, γ) 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 L_1. Therefore, it is an element of $\top_{L_1}$, which is a subset of the lexicon.

Let us illustrate how the attachment of the side spine to the main spine is realized. The lexicon contains $(q_1', \upsilon, p_1')\backslash(q_2, \upsilon, \alpha)\backslash(p_0, \chi, \beta)$, of which 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. ■

For the converse inclusion we utilize Theorem 20 of Kuhlmann et al. (2019). It states that for every CCG $\mathcal{G}'$ there exists an sCFTG that generates the rule trees of $\mathcal{G}'$. Whereas derivation trees are labeled by categories, *rule trees* are labeled by lexicon entries at leaves and

by applied rules (instead of the output category) at inner nodes. Rule trees are a natural encoding of derivation trees using only a finite set of labels. As each rule indicates the target and last argument of its output category, rule trees can be relabeled in the same manner as derivation trees. For completeness' sake we restate Definition 16 of Kuhlmann et al. (2019).

Definition 24. Let $\mathcal{G} = (\Sigma, A, R, I, \mathcal{L})$ be a CCG and $\mathrm{T} = T_{R,\emptyset}(\mathcal{L}(\Sigma))$. A tree $t \in \mathrm{T}$ is a *rule tree* if $\mathrm{cat}(t) \in I$, where the partial map $\mathrm{cat}\colon \mathrm{T} \to C(A)$ is inductively defined by (i) $\mathrm{cat}(a) = a$ for all lexicon entries $a \in \mathcal{L}(\Sigma)$, (ii) $\mathrm{cat}\left(\dfrac{axy}{ax\backslash b \quad by}(t_1, t_2)\right) = azy$ for all trees $t_1, t_2 \in \mathrm{T}$ with $\mathrm{cat}(t_1) = az/b$ and $\mathrm{cat}(t_2) = by$, and (iii) $\mathrm{cat}\left(\dfrac{axy}{by \quad ax\backslash b}(t_1, t_2)\right) = azy$ for all $t_1, t_2 \in \mathrm{T}$ with $\mathrm{cat}(t_1) = by$ and $\mathrm{cat}(t_2) = az\backslash b$. The set of all rule trees of $\mathcal{G}$ is denoted by $\mathcal{R}(\mathcal{G})$. ■

We observe that any category relabeling can equivalently be applied to rule trees instead of derivation trees (because a category relabeling only depends on the target a and the last argument $|\, b$ of a category $ax \,|\, b$). This yields the second main theorem.

Theorem 25. *CCGs and sCFTGs are strongly equivalent up to relabeling.* ■

Kepser and Rogers (2011) proved that TAGs and sCFTGs are strongly equivalent, which shows that they are also strongly equivalent (up to relabeling) to CCGs.

Corollary 26. *CCGs and TAGs are strongly equivalent up to relabeling.* ■

Clearly, from strong equivalence we can conclude weak equivalence as well (without the relabeling since the lexicon provides the relabeling). Weak equivalence was famously proven by Vijay-Shanker and Weir (1994), but Theorem 3 of Kuhlmann et al. (2015) shows that the original construction is incorrect. However, Weir (1988) provides an alternative construction and proof. 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 is a category assigned to the empty string; these interspersed categories form the main building block in the original constructions. The necessity of these ε-entries (Vijay-Shanker and Weir, 1994) is an interesting and important question that naturally arises and has been asked by Kuhlmann et al. (2015). We settle this question and demonstrate that they can be avoided.

Corollary 27. *CCGs and TAGs are weakly equivalent, and CCGs with ε-entries and CCGs generate the same (ε-free) languages.* ■

The tree expressive power of CCGs with restricted rule degrees has already been investigated by Kuhlmann et al. (2019). It has been shown that 0-CCGs accept a proper subset of the regular tree languages (Gécseg and Steinby, 1997), whereas 1-CCGs accept exactly the regular tree languages. It remained open whether there is a k such that k-CCGs and $(k+1)$-CCGs have the same expressive power. Our construction establishes that 2-CCGs are as expressive as k-CCGs for arbitrary $k \geq 2$. Another consequence of our construction is that first-order categories are sufficient.

Corollary 28. *2-CCGs with first-order categories have the same expressive power as k-*CCGs *with $k > 2$.* ■

8 Conclusion

We presented a translation from spine grammar to CCG. Due to the strong equivalence of spine grammar and TAG (Kepser and Rogers, 2011), we can also construct a strongly equivalent CCG for each TAG. Together with the translation from CCG to sCFTG (Kuhlmann et al., 2019), this proves the strong equivalence of TAG and CCG, which means that both formalisms generate the same derivation trees modulo relabelings. Our construction uses CCG rules of degree at most 2, only first-order categories, lexicon entries of arity at most 3, and no ε-entries in the lexicon. Such CCGs thus have full expressive power. Avoiding ε-entries is particularly interesting because they violate the Principle of Adjacency (Steedman, 2000, p. 54), which is a fundamental linguistic principle underlying CCG and requires that all combining categories correspond to phonologically realized counterparts in the input and are string-adjacent. Their elimination is performed by trimming them from the sCFTG obtained from a CCG with ε-entries and translating the trimmed sCFTG back to a CCG using our construction.

Translating CCG to sCFTG (Kuhlmann et al., 2019) yields sCFTGs whose size is exponential

in a CCG-specific constant, which depends on the maximal rule degree and the maximal arity of lexicon entries. The increase can be attributed to variables in CCG rules, which need to be properly instantiated. Our construction increases the grammar size only polynomially, which can be verified for each step. Overall, a k-CCG can be converted to an equivalent 2-CCG without ε-entries in time and space exponential in k (and the maximal length of lexicon entries) and polynomial in the size of the grammar.

Acknowledgments

We would like to thank Mark Steedman and the three anonymous reviewers for their valuable and detailed comments, which greatly helped in improving the comprehensibility of this paper. The work of Lena Katharina Schiffer was funded by the German Research Foundation (DFG) Research Training Group GRK 1763 'Quantitative Logics and Automata'.

References

Jean-Michel Autebert, Jean Berstel, and Luc Boasson. 1997. Context-free languages and pushdown automata, Grzegorz Rozenberg and Arto Salomaa, editors, *Handbook of Formal Languages*, volume 1, chapter 3, pages 111–174. Springer. https://doi.org/10.1007/978-3-642-59136-5_3

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

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

Haskell B. Curry, Robert Feys, and William Craig. 1958. *Combinatory Logic*. Number 1 in Studies in Logic and the Foundations of Mathematics. North-Holland.

Normann Decker, Martin Leucker, and Daniel Thoma. 2013. Impartiality and anticipation for monitoring of visibly context-free properties. In *Proc. Runtime Verification*, volume 8174 of *LNCS*, pages 183–200. Springer. https://doi.org/10.1007/978-3-642-40787-1_11

Herbert Fleischner. 1977. On the equivalence of Mealy-type and Moore-type automata and a relation between reducibility and Moore-reducibility. *Journal of Computer and System Sciences*, 14(1):1–16. https://doi.org/10.1016/S0022-0000(77)80038-X

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

Ferenc Gécseg and Magnus Steinby. 1997. Tree languages. In Grzegorz Rozenberg and Arto Salomaa, editors, *Handbook of Formal Languages*, volume 3, chapter 1, pages 1–68. Springer. https://doi.org/10.1007/978-3-642-59126-6_1

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

Julia Hockenmaier and Peter Young. 2008. Non-local scrambling: The equivalence of TAG and CCG revisited. In *Proc. 9th TAG+*. University of Tübingen.

Aravind K. Joshi. 1985. Tree adjoining grammars: How much context-sensitivity is required to provide reasonable structural descriptions? In David R. Dowty, Lauri Karttunen, and Arnold M. Zwicky, editors, *Natual Language Parsing*, chapter 6, pages 206–250. Cambridge University Press. https://doi.org/10.1017/CBO9780511597855.007

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

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

Marco Kuhlmann, Alexander Koller, and Giorgio Satta. 2010. The importance of rule restrictions in CCG. In *Proc. Association for Computational Linguistics*, pages 534–543. ACL.

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

Marco Kuhlmann, Andreas Maletti, and Lena K. Schiffer. 2019. The tree-generative capacity of combinatory categorial grammars. In *Proc. Foundations of Software Technology and Theoretical Computer Science*, volume 150 of *LIPIcs*, pages 44:1–44:14. Schloss Dagstuhl — Leibniz-Zentrum für Informatik.

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

Marco Kuhlmann and Giorgio Satta. 2012. Tree-adjoining grammars are not closed under strong lexicalization. *Computational Linguistics*, 38(3), 617–629.

William C. Rounds. 1969. Context-free grammars on trees. In *Proc. Symposium on Theory of Computing*, pages 143–148. ACM. https://doi.org/10.1145/800169.805428

Mark Steedman. 2000. *The Syntactic Process*. MIT Press. https://doi.org/10.1002/9781444395037.ch5

Mark Steedman and Jason Baldridge. 2011. Combinatory categorial grammar. In Robert D. Borsley and Kersti Börjars, editors, *Non-Transformational Syntax: Formal and Explicit Models of Grammar*, chapter 5, pages 181–224. Blackwell.

Krishnamurti Vijay-Shanker. 1988. *A Study of Tree Adjoining Grammars*. Ph.D. thesis, University of Pennsylvania.

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

David J. Weir. 1988. *Characterizing Mildly Context-sensitive Grammar Formalisms*. Ph.D. thesis, University of Pennsylvania.