Lexicalization and Generative Power in CCG

Marco Kuhlmann*
Linköping University

Alexander Koller**
University of Potsdam

Giorgio Satta†
University of Padua

The weak equivalence of Combinatory Categorial Grammar (CCG) and Tree-Adjoining Grammar (TAG) is a central result of the literature on mildly context-sensitive grammar formalisms. However, the categorial formalism for which this equivalence has been established differs significantly from the versions of CCG that are in use today. In particular, it allows restriction of combinatory rules on a per grammar basis, whereas modern CCG assumes a universal set of rules, isolating all cross-linguistic variation in the lexicon. In this article we investigate the formal significance of this difference. Our main result is that lexicalized versions of the classical CCG formalism are strictly less powerful than TAG.

1. Introduction

Since the late 1970s, several grammar formalisms have been proposed that extend the power of context-free grammars in restricted ways. The two most prominent members of this class of "mildly context-sensitive" formalisms (a term coined by Joshi 1985) are Tree-Adjoining Grammar (TAG; Joshi and Schabes 1997) and Combinatory Categorial Grammar (CCG; Steedman 2000; Steedman and Baldridge 2011). Both formalisms have been applied to a broad range of linguistic phenomena, and are being widely used in computational linguistics and natural language processing.

In a seminal paper, Vijay-Shanker and Weir (1994) showed that TAG, CCG, and two other mildly context-sensitive formalisms—Head Grammar (Pollard 1984) and Linear Indexed Grammar (Gazdar 1987)—all characterize the same class of string languages. However, when citing this result it is sometimes overlooked that the result applies to a version of CCG that is quite different from the versions that are in practical use today.

* Department of Computer and Information Science, Linköping University, 581 83 Linköping, Sweden. E-mail: marco.kuhlmann@liu.se.
** Department of Linguistics, Karl-Liebknecht-Str. 24–25, University of Potsdam, 14476 Potsdam, Germany. E-mail: koller@ling.uni-potsdam.de.
† Department of Information Engineering, University of Padua, via Gradenigo 6/A, 35131 Padova, Italy. E-mail: satta@dei.unipd.it.

Submission received: 4 December 2013; revised submission received: 26 July 2014; accepted for publication: 25 November 2014.

doi:10.1162/COLI_a_00219

The goal of this article is to contribute to a better understanding of the significance of this difference.

The difference between "classical" CCG as formalized by Vijay-Shanker and Weir (1994) and the modern perspective may be illustrated with the combinatory rule of backward-crossed composition. The general form of this rule looks as follows:

Backward-crossed composition, general form:

$$Y/Z \quad X\backslash Y \quad \Rightarrow \quad X/Z \qquad (<\mathbf{B}_\times)$$

This rule is frequently used in the analysis of heavy NP shift, as in the sentence *Kahn blocked skillfully a powerful shot by Rivaldo* (example from Baldridge 2002). However, backward crossed composition cannot be universally active in English as this would cause the grammar to also accept strings such as **a powerful by Rivaldo shot*, which is witnessed by the derivation in Figure 1. To solve this problem, in the CCG formalism of Vijay-Shanker and Weir (1994), one may restrict backward-crossed composition to instances where X and Y are both verbal categories—that is, functions into the category S of sentences (cf. Steedman 2000, Section 4.2.2). With this restriction the unwanted derivation in Figure 1 can be blocked, and *a powerful shot by Rivaldo* is still accepted as grammatical. Other syntactic phenomena require other grammar-specific restrictions, including the complete ban of certain combinatory rules (cf. Steedman 2000, Section 4.2.1).

Over the past 20 years, CCG has evolved to put more emphasis on supporting *fully lexicalized* grammars (Baldridge and Kruijff 2003; Steedman and Baldridge 2011), in which as much grammatical information as possible is pushed into the lexicon. This follows the tradition of other frameworks such as Lexicalized Tree-Adjoining Grammar (LTAG) and Head-Driven Phrase Structure Grammar (HPSG). Grammar-specific rule restrictions are not connected to individual lexicon entries, and are therefore avoided. Instead, recent versions of CCG have introduced a new, lexicalized control mechanism in the form of *modalities* or *slash types*. The basic idea here is that combinatory rules only apply if the slashes in their input categories have the right types. For instance, the modern version of backward-crossed composition (Steedman and Baldridge 2011) takes the following form, where $\times$ is a slash type that allows crossed but not harmonic composition:

Backward-crossed composition, modalized form:

$$Y/_\times Z \quad X\backslash_\times Y \quad \Rightarrow \quad X/_\times Z \qquad (<\mathbf{B}_\times)$$

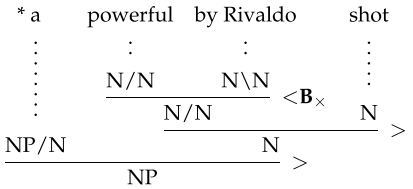


Figure 1
Overgeneration caused by unrestricted backward crossed composition.

The derivation in Figure 1 can now be blocked by assigning the slash in the category of *powerful* a type that is incompatible with $\times$, and thus cannot feed into type-aware backward-crossed composition as an input category. This "modalization" of grammatical composition, which imports some of the central ideas from the type-logical tradition of categorial grammar (Moortgat 2011) into the CCG framework, is attractive because it isolates the control over the applicability of combinatory rules in the lexicon. Quoting Steedman and Baldridge (2011, p. 186):

> Without typed slashes, language-specific restrictions or even bans on some combinatory rules are necessary in order to block certain ungrammatical word orders. With them, the combinatory rules are truly universal: the grammar of every language utilizes exactly the same set of rules, without modification, thereby leaving all cross-linguistic variation in the lexicon. As such, CCG is a *fully* lexicalized grammar formalism.

The stated goal is thus to express all linguistically relevant restrictions on the use of combinatory rules in terms of lexically specified, typed slashes. But to what extent can this goal actually be achieved? So far, there have been only partial answers—even when, as in the formalism of Vijay-Shanker and Weir (1994), we restrict ourselves to the rules of composition, but exclude other rules such as type-raising and substitution. Baldridge and Kruijff (2003) note that the machinery of their multi-modal formalism can be simulated by rule restrictions, which shows that this version of lexicalized CCG is *at most as expressive* as the classical formalism. At the other end of the spectrum, we have shown in previous work that a "pure" form of lexicalized CCG with neither rule restrictions nor slash types is *strictly less expressive* (Kuhlmann, Koller, and Satta 2010). The general question of whether "classical" CCG (rule restrictions) and "modern" CCG (slash types instead of rule restrictions) are weakly equivalent has remained open.

In this article, we answer this question. Our results are summarized in Figure 2. After setting the stage (Section 2), we first pinpoint the exact type of rule restrictions that make the classical CCG formalism weakly equivalent to TAG (Section 3). We do so by focusing on a class of grammars that we call *prefix-closed*. Unlike the "pure" grammars that we studied in earlier work, prefix-closed grammars do permit rule restrictions (under certain conditions that seem to be satisfied by linguistic grammars). We show that the generative power of prefix-closed CCG depends on its ability to express *target restrictions*, which are exactly the functions-into type of restrictions that are needed to block the derivation in Figure 1. We prove that the full class of prefix-closed CCGs is weakly equivalent to TAG, and the subclass that cannot express target restrictions is strictly less powerful. This result significantly sharpens the picture of the generative capacity of CCG.

In a further step (Section 4), we then prove that for at least one popular incarnation of modern, lexicalized CCG, slash types are strictly less expressive than rule

Weakly equivalent to TAG	**Strictly less powerful than TAG**
VW-CCG	
Prefix-closed VW-CCG with target restrictions (Theorem 2)	Prefix-closed VW-CCG without target restrictions (Theorem 3)
	O-CCG (Theorem 4)

Figure 2
Summary of the results in this article. VW-CCG is the formalism of Vijay-Shanker and Weir (1994).

restrictions. More specifically, we look at a variant of CCG consisting of the composition rules implemented in OpenCCG (White 2013), the most widely used development platform for CCG grammars. We show that this formalism is (almost) prefix-closed and cannot express target restrictions, which enables us to apply our generative capacity result from the first step. The same result holds for (the composition-only fragment of) the formalism of Baldridge and Kruijff (2003). Thus we find that, at least with existing means, the weak equivalence result of Vijay-Shanker and Weir cannot be obtained for lexicalized CCG. We conclude the article by discussing the implications of our results (Section 5).

2. Background

In this section we provide the technical background of the article. We introduce the basic architecture of CCG, present the formalism of Vijay-Shanker and Weir (1994), and set the points of reference for our results about generative capacity.

2.1 Basic Architecture of CCG

The two central components of CCG are a *lexicon* that associates words with *categories*, and *rules* that specify how categories can be combined. Taken together, these components give rise to *derivations*, such as the one shown in Figure 3.

Lexicon. A **category** is a syntactic type that identifies a constituent as either "complete" or "incomplete." The categories of complete constituents are taken to be primitives; the categories of incomplete constituents are modeled as (curried) functions that specify the type and the direction of their arguments. Every category is projected from the **lexicon**, which is a finite collection of word–category pairs. To give examples (taken from Steedman 2012), an English intransitive verb such as *walks* has a category that identifies it as a function seeking a (subject) noun phrase to its left (indicated by a backward slash) and returning a complete sentence; and a transitive verb such as *admires* has a category that identifies it as a function seeking an (object) noun phrase to its right (indicated by a forward slash) and returning a constituent that acts as an intransitive verb. We denote lexical assignment using the "colon equals" operator:

$$walks := \mathrm{S}\backslash\mathrm{NP} \qquad admires := (\mathrm{S}\backslash\mathrm{NP})/\mathrm{NP}$$

Formally, the set $\mathcal{C}(\mathcal{A})$ of categories is built over a finite set $\mathcal{A}$ of atomic categories, the primitive syntactic types. It is the smallest set such that

1. if $A \in \mathcal{A}$ then $A \in \mathcal{C}(\mathcal{A})$;

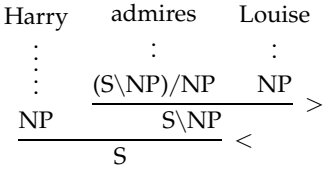


Figure 3
A sample derivation (adapted from Steedman 2012).

2. if $X, Y \in \mathcal{C}(\mathcal{A})$ then $X/Y \in \mathcal{C}(\mathcal{A})$ and $X\backslash Y \in \mathcal{C}(\mathcal{A})$.

We use the letters A, B, C to denote atomic categories, the letters X, Y, Z to denote arbitrary categories, and the symbol $|$ to denote slashes (forward or backward). We treat slashes as left-associative operators and omit unnecessary parentheses. By this convention, every category X can be written in the form

$$X \quad = \quad A|_m X_m \cdots |_1 X_1$$

where $m \geq 0$, A is an atomic category that we call the **target** of X, and the $|_i X_i$ are slash–category pairs that we call the **arguments** of X. Intuitively, the target of X is the atomic category that is obtained after X has been applied to all of its arguments. We use the Greek letters α, β, γ to denote (potentially empty) sequences of arguments. The number m is called the **arity** of the category X.

Rules. Categories can combine under a number of **rules**; this gives rise to derivations such as the one shown in Figure 3. Each rule specifies an operation that assembles two input categories into an output category. The two most basic rules of CCG are the directed versions of function application:

$$X/Y \quad Y \quad \Rightarrow \quad X \qquad \text{(forward application)} \qquad (>)$$

$$Y \quad X\backslash Y \quad \Rightarrow \quad X \qquad \text{(backward application)} \qquad (<)$$

Formally, a rule is a syntactic object in which the letters X, Y, Z act as variables for categories. A **rule instance** is obtained by substituting concrete categories for all variables in the rule. For example, the derivation in Figure 3 contains the following instances of function application. We denote rule instances by using a triple arrow instead of the double arrow in our notation for rules.

$$\text{(S\backslash NP)/NP} \quad \text{NP} \quad \Rrightarrow \quad \text{S\backslash NP} \qquad \text{and} \qquad \text{NP} \quad \text{S\backslash NP} \quad \Rrightarrow \quad \text{S}$$

Application rules give rise to derivations equivalent to those of context-free grammar. Indeed, versions of categorial grammar where application is the *only* mode of combination, such as AB-grammar (Ajdukiewicz 1935; Bar-Hillel, Gaifman, and Shamir 1960), can only generate context-free languages. CCG can be more powerful because it also includes other rules, derived from the combinators of combinatory logic (Curry, Feys, and Craig 1958). In this article, as in most of the formal work on CCG, we restrict our attention to the rules of (generalized) composition, which are based on the **B** combinator.[1]

The general form of composition rules is shown in Figure 4. In each rule, the two input categories are distinguished into one **primary** (shaded) and one **secondary** input category. The number n of outermost arguments of the secondary input category is called the **degree** of the rule.[2] In particular, for $n = 0$ we obtain the rules of function

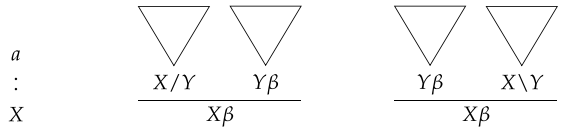
1 This means that we ignore other rules required for linguistic analysis, in particular type-raising (from the **T** combinator), substitution (from the **S** combinator), and coordination.

2 The literature on CCG assumes a bound on n; for English, Steedman (2000, p. 42) puts $n \leq 3$. Adding rules of unbounded degree increases the generative capacity of the formalism (Weir and Joshi 1988).

$$X/Y \quad Y|_n Y_n \cdots |_1 Y_1 \quad \Rightarrow \quad X|_n Y_n \cdots |_1 Y_1 \qquad \text{(forward)} \qquad (>^n)$$

$$Y\backslash_n Y_n \cdots |_1 Y_1 \quad X\backslash Y \quad \Rightarrow \quad X|_n Y_n \cdots |_1 Y_1 \qquad \text{(backward)} \qquad (<^n)$$

Figure 4
General form of composition rules (primary input category shaded), where $n \geq 0$ and $|_i \in \{/, \backslash\}$.



iff $a := X$ iff $X/Y \ Y\beta \Rightarrow X\beta$ iff $Y\beta \ X\backslash Y \Rightarrow X\beta$

Figure 5
Schematic definition of the set of derivation trees of a grammar G.

application. In contexts where we refer to both application and composition, we use the latter term for composition rules with degree $n > 0$.

Derivation Trees. Derivation trees can now be schematically defined as in Figure 5. They contain two types of branchings: unary branchings correspond to lexicon entries; binary branchings correspond to rule instances. The **yield** of a derivation tree is the left-to-right concatenation of its leaves.

2.2 Classical CCG

We now define the classical CCG formalism that was studied by Vijay-Shanker and Weir (1994) and originally introduced by Weir and Joshi (1988). As mentioned in Section 1, the central feature of this formalism is its ability to impose restrictions on the applicability of combinatory rules. Specifically, a **restricted rule** is a rule annotated with constraints that

(a) restrict the target of the primary input category; and/or

(b) restrict the secondary input category, either in parts or in its entirety.

Every grammar lists a finite number of restricted rules (where one and the same base rule may occur with several different restrictions). A **valid rule instance** is an instance that is compatible with at least one of the restricted rules.

Example 1
Linguistic grammars make frequent use of rule restrictions. To exclude the undesired derivation in Figure 1 we restricted backward crossed composition to instances where both the primary and the secondary input category are functions into the category of

sentences, S. Writing *target* for the function that returns the target of a category, the restricted rule can be written as

$$Y/Z \quad X\backslash Y \quad \Rightarrow \quad X/Z \qquad \text{(backward crossed composition)} \qquad (<\mathbf{B}_\times)$$
$$\text{where } \mathit{target}(X) = \mathrm{S} \text{ and } \mathit{target}(Y) = \mathrm{S}$$

In the following definition we write ε for the empty string.

Definition 1
A **combinatory categorial grammar** in the sense of Vijay-Shanker and Weir (1994), or VW-CCG for short, is a structure

$$G \quad = \quad (\Sigma, \mathcal{A}, :=, R, S)$$

where Σ is a finite vocabulary, $\mathcal{A}$ is a finite set of atomic categories, the lexicon $:=$ is a finite relation between the sets $\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$ and $\mathcal{C}(\mathcal{A})$, R is a finite set of (possibly) restricted rules, and $S \in \mathcal{A}$ is a distinguished atomic category.

A derivation tree of G consists of lexicon entries and valid instances of rules from R. The grammar G **generates** a string w if there exists a derivation tree whose yield is w and whose root node is labeled with the distinguished atomic category S. The **language** $L(G)$ generated by G is the set of all generated strings.

Example 2
We specify a VW-CCG $G_1 = (\Sigma_1, \mathcal{A}_1, :=_1, R_1, S_1)$ that generates the language

$$L_1 = \{a^n b^n c^n \mid n \geq 1\}\,.$$

Let $\Sigma_1 = \{a, b, c\}$, $\mathcal{A}_1 = \{A, B, C, S\}$, $S_1 = S$, and let $:=_1$ consist of the entries

$$a :=_1 A\ ,\ b :=_1 S/C\ ,\ b :=_1 S/C/B\ ,\ b :=_1 B/C\ ,\ b :=_1 B/C/B\ ,\ c :=_1 C\backslash A$$

Finally, let R_1 consist of the following rules:

1. Forward and backward application.
2. Forward and backward composition of degree 1.
3. Forward and backward composition of degree 2.

Each of these rules is restricted to instances where the target of the primary input category is S. A sample derivation of G_1 is shown in Figure 6. By contrast, Figure 7 shows a derivation that is not valid in G_1 because it uses application rules in which the target of the primary input category is B or C (rather than S).

Example 3
AB-grammar is a categorial grammar formalism in which forward and backward application are the only combinatory rules that are allowed. Furthermore, it does not support

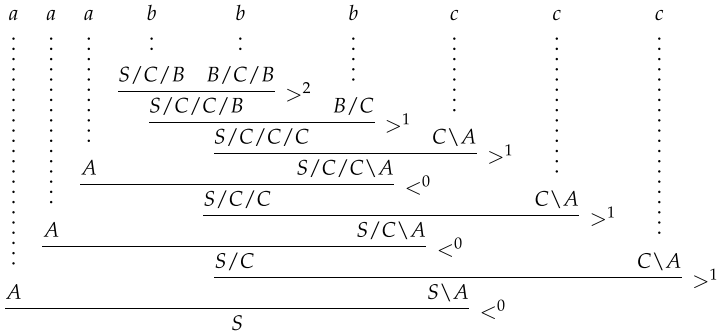


Figure 6
A sample derivation of the grammar G_1 from Example 2.

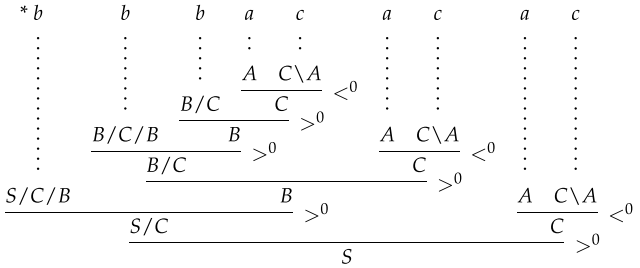


Figure 7
A derivation rendered ungrammatical by the rule restrictions in G_1.

rule restrictions.[3] Every AB-grammar can therefore be written as a VW-CCG that allows all instances of application, but no other rules.

The following lemmas establish two fundamental properties of VW-CCG that we shall use at several places in this article. Both of them are minor variants of lemmas proved by Vijay-Shanker and Weir (1994) (Lemma 3.1 and Lemma 3.2) and were previously stated by Weir and Joshi (1988).

Lemma 1
The set of arguments that occur in the derivations of a VW-CCG is finite.

Proof
No composition rule creates new arguments: Every argument that occurs in an output category already occurs in one of the input categories. Therefore, every argument must come from some word–category pair in the lexicon, of which there are only finitely many. ■

Lemma 2
The set of secondary input categories that occur in the derivations of a VW-CCG is finite.

3 Also, AB-grammar does not support lexicon entries for the empty string.

Proof
Every secondary input category is obtained by substituting concrete categories for the variables that occur in the non-shaded component of one of the rules specified in Figure 4. After the substitution, all of these categories occur as part of arguments. Then, with Lemma 1, we deduce that the substituted categories come from a finite set. At the same time, each grammar specifies a finite set of rules. This means that there are only finitely many ways to obtain a secondary input category. ■

When specifying VW-CCGs, we find it convenient sometimes to provide an explicit list of valid rule instances, rather than a textual description of rule restrictions. For this we use a special type of restricted rule that we call **templates**. A template is a restricted rule that simultaneously fixes both

(a) the target of the primary input category of the rule, and

(b) the entire secondary input category.

We illustrate the idea with an example.

Example 4
We list the templates that correspond to the rule instances in the derivation from Figure 6. (The grammar allows other instances that are not listed here.) We use the symbol $\square$ as a placeholder for that part of a primary input category that is unconstrained by rule restrictions, and therefore may consist of an arbitrary sequence of arguments.

$$A \quad S\square\backslash A \quad \Rightarrow \quad S\square \tag{1}$$

$$S\square/C \quad C\backslash A \quad \Rightarrow \quad S\square\backslash A \tag{2}$$

$$S\square/B \quad B/C \quad \Rightarrow \quad S\square/C \tag{3}$$

$$S\square/B \quad B/C/B \quad \Rightarrow \quad S\square/C/B \tag{4}$$

For example, template (1) characterizes backward application ($<^0$) where the target of the primary input category is S and the secondary input category is A, and template (4) characterizes forward composition of degree 2 ($>^2$) where the target of the primary input category is S and the secondary input category is $B/C/B$.

Note that every VW-CCG can be specified using a *finite* set of templates: It has a finite set of combinatory rules; the set of possible targets of the primary input category of each rule is finite because each target is an atomic category; and the set of possible secondary input categories is finite because of Lemma 2.

2.3 Tree-Adjoining Grammar

In this article we are interested in the generative power of CCG, in particular in its relation to that of Tree-Adjoining Grammar. We therefore provide a compact introduction to TAG. For more details, we refer to Joshi and Schabes (1997).

Elementary Trees. Tree-Adjoining Grammar is a formalism for generating trees. These trees can be characterized as rooted, ordered trees in which internal nodes are labeled with nonterminal symbols—including a distinguished **start symbol** S—and leaf nodes are labeled with nonterminals, terminals, or the empty string. Every grammar specifies a finite set of such trees; these are called **elementary trees**. There are two types: **initial trees** and **auxiliary trees**. They differ in that auxiliary trees have a distinguished nonterminal-labeled leaf node, called **foot node**; this node is conventionally marked with an asterisk. An elementary tree whose root node is labeled with a nonterminal A is called an A-tree.

Substitution and Adjunction. New trees may be derived by combining other trees using two operations called **substitution** and **adjunction**. Substitution replaces some leaf node of a given tree with an initial tree (or a tree derived from an initial tree). Adjunction replaces some internal node u of a given tree with an auxiliary tree (or a tree derived from an auxiliary tree); the subtree with root u replaces the foot node of the auxiliary tree. All replacements are subject to the condition that the node being replaced and the root of the tree that replaces it are labeled with the same nonterminal.

Generated Languages. The **tree language** generated by a TAG is the set of all trees that can be derived from its initial S-trees. Derivations are considered complete only if they satisfy additional, node-specific constraints. In particular, substitution is obligatory at every node where it is possible, and adjunction may be specified as either obligatory (OA, Obligatory Adjunction) or forbidden (NA, Null Adjunction) at a given node. In derived trees corresponding to complete derivations, all leaf nodes are labeled with terminal symbols. The left-to-right concatenation of these symbols forms the **yield** of the tree, and the yields of all trees in the tree language form the **string language** generated by the TAG.

Example 5
Figure 8 shows a TAG that generates the language $L_1 = \{a^n b^n c^n \mid n \geq 1\}$ defined in Example 2. Derivations start with adjoining the auxiliary tree t_2 at the root of the initial tree t_1. New trees can be derived by repeatedly adjoining t_2 at an S-node.

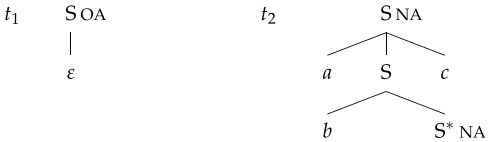


Figure 8
A TAG that generates the language $L_1 = \{a^n b^n c^n \mid n \geq 1\}$.

2.4 Generative Capacity of Classical CCG

Vijay-Shanker and Weir (1994) proved the following:

Theorem 1
VW-CCG and TAG are weakly equivalent.

The inclusion of the VW-CCG languages in the TAG languages follows from a chain of inclusions that connects VW-CCG and TAG via Linear Indexed Grammar (LIG; Gazdar 1987) and Head Grammar (HG; Pollard 1984). All of these inclusions were proved by Vijay-Shanker and Weir (1994). Here we sketch a proof of the inclusion of the TAG languages in the VW-CCG languages. Our proof closely follows that of Weir (1988, Section 5.2.2), whose construction we shall return to when establishing our own results.

Lemma 3
The TAG languages are included in the VW-CCG languages.

Proof (Sketch)
We are given a TAG G and construct a weakly equivalent VW-CCG G'. The basic idea is to make the lexical categories of G' correspond to the elementary trees of G, and to set up the combinatory rules and their restrictions in such a way that the derivations of G' correspond to derivations of G.

Vocabulary, Atomic Categories. The vocabulary of G' is the set of all terminal symbols of G; the set of atomic categories consists of all symbols of the form A^t, where either A is a nonterminal symbol of G and $t \in \{a, c\}$, or A is a terminal symbol of G and $t = a$. The distinguished atomic category of G' is S^a, where S is the start symbol of G.

Lexicon. One may assume (cf. Vijay-Shanker, Weir, and Joshi 1986) that G is in the normal form shown in Figure 9. In this normal form there is a single initial S-tree, and all remaining elementary trees are auxiliary trees of one of five possible types. For each such tree, one constructs two lexicon entries for the empty string ε as specified in Figure 9. Additionally, for each terminal symbol x of G, one constructs a lexicon entry $x := x^a$.

Rules. The rules of G' are forward and backward application and forward and backward composition of degree at most 2. They are used to simulate adjunction operations in derivations of G: Application simulates adjunction into nodes to the left or right of the foot node; composition simulates adjunction into nodes above the foot node. Without restrictions, these rules would allow derivations that do not correspond to derivations of G. Therefore, rules are restricted such that an argument of the form $|A^t$ can be eliminated by means of an application rule only if $t = a$, and by means of a composition rule only if $t = c$. This enforces two properties that are central for the correctness of the construction (Weir 1988, p. 119): First, the secondary input category in every instance of composition is a category that has just been introduced from the

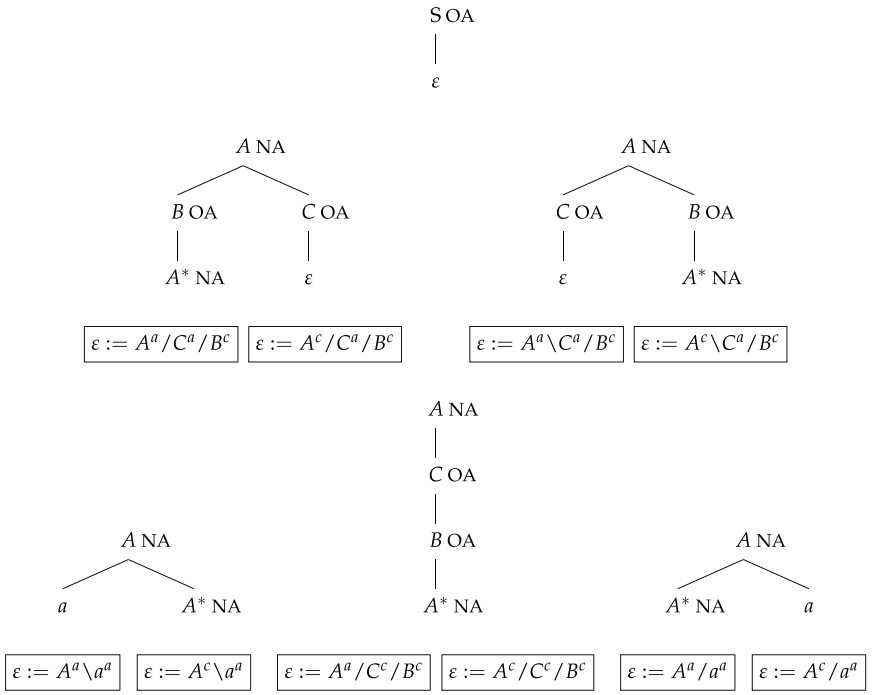


Figure 9
Correspondence between elementary trees and lexical categories in the proof of Lemma 3.

lexicon. Second, categories cannot be combined in arbitrary orders. The rule restrictions are:

1. Forward and backward application are restricted to instances where both the target of the primary input category and the entire secondary input category take the form A^a.
2. Forward and backward composition are restricted to instances where the target of the primary input category takes the form A^a and the target of the secondary input category takes the form A^c.

Using our template notation, the restricted rules can be written as in Figure 10. ■

As an aside we note that the proof of Lemma 3 makes heavy use of the ability of VW-CCG to assign lexicon entries to the empty string. Such lexicon entries violate one of the central linguistic principles of CCG, the Principle of Adjacency, according to which combinatory rules may only apply to phonologically realized entities (Steedman 2000, p. 54). It is an interesting question for future research whether a version of VW-CCG without lexicon entries for the empty string remains weakly equivalent to TAG.

$$
\begin{array}{rcll}
A^a \,\square\, / B^a \quad B^a & \Rrightarrow & A^a \,\square & \text{where} \quad A, B \in \mathcal{V} \\
B^a \quad A^a \,\square\, \backslash B^a & \Rrightarrow & A^a \,\square & \text{where} \quad A, B \in \mathcal{V} \\
A^a \,\square\, / B^c \quad B^c|_1 X_1 & \Rrightarrow & A^a \,\square\, |_1 X_1 & \text{where} \quad A, B \in \mathcal{V},\ X_1 \in \mathcal{V}^a \cup \mathcal{V}^c \\
B^c|_1 X_1 \quad A^a \,\square\, \backslash B^c & \Rrightarrow & A^a \,\square\, |_1 X_1 & \text{where} \quad A, B \in \mathcal{V},\ X_1 \in \mathcal{V}^a \cup \mathcal{V}^c \\
A^a \,\square\, / B^c \quad B^c|_2 X_2|_1 X_1 & \Rrightarrow & A^a \,\square\, |_2 X_2|_1 X_1 & \text{where} \quad A, B \in \mathcal{V},\ X_1, X_2 \in \mathcal{V}^a \cup \mathcal{V}^c \\
B^c|_2 X_2|_1 X_1 \quad A^a \,\square\, \backslash B^c & \Rrightarrow & A^a \,\square\, |_2 X_2|_1 X_1 & \text{where} \quad A, B \in \mathcal{V},\ X_1, X_2 \in \mathcal{V}^a \cup \mathcal{V}^c
\end{array}
$$

Figure 10
Restricted rules used in the proof of Lemma 3. We write $\mathcal{V}$ for the union of the nonterminal symbols and terminal symbols of the TAG G, and let $\mathcal{V}^t = \{A^t \mid A \in \mathcal{V}\}$, $t \in \{a, c\}$.

3. Relevance of Target Restrictions in Prefix-Closed CCG

In this section we present the central technical results of this article. We study a class of VW-CCGs that we call *prefix-closed* and show that for this class, weak equivalence with TAG stands and falls with the ability to specify target restrictions.

3.1 Prefix-Closed Grammars

Rule restrictions are important tools in grammars for natural language; but not all of their potential uses have obvious linguistic motivations. For instance, one could write a grammar that permits all compositions with a functional category A/B as the secondary input category, but rules out application with the "shorter" category A. Such constraints do not seem to be used in linguistically motivated grammars; for example, none of the grammars developed by Steedman (2000) needs them. In prefix-closed grammars, this use of rule restrictions is explicitly barred.

Definition 2
A VW-CCG is **prefix-closed** if it satisfies the following implication:

$$
\begin{array}{rcll}
\text{if} & X/Y \quad Y|_n Y_n \cdots |_1 Y_1 & \Rrightarrow & X|_n Y_n \cdots |_1 Y_1 \quad \text{is a valid rule instance} \\
\text{then so is} & X/Y \quad Y|_n Y_n \cdots |_k Y_k & \Rrightarrow & X|_n Y_n \cdots |_k Y_k \quad \text{for any } k \geq 1
\end{array}
$$

and similarly for backward rules.

Note that prefix-closed grammars still allow some types of rule restrictions. The crucial property is that, if a certain combinatory rule applies at all, then it also applies to combinations where the secondary input category has already been applied to some ($k \leq n$) or even all ($k > n$) of its arguments.

Example 6
We illustrate prefix-closedness using some examples:

1. Every AB-grammar (when seen as a VW-CCG) is trivially prefix-closed; in these grammars, $n = 0$.
2. The "pure" grammars that we considered in our earlier work (Kuhlmann, Koller, and Satta 2010) are trivially prefix-closed.

3. The grammar G_1 from Example 2 is prefix-closed.
4. The grammars constructed in the proof of Lemma 3 are not prefix-closed; they do not allow the following instances of application, where the secondary input category is of the form B^c (rather than B^a):

$$A^a \Box / B^c \quad B^c \Rrightarrow A^a \Box \qquad \text{where} \qquad A, B \in \mathcal{V}$$
$$B^c \quad A^a \Box \backslash B^c \Rrightarrow A^a \Box \qquad \text{where} \qquad A, B \in \mathcal{V}$$

Example 7
The linguistic intuition underlying prefix-closed grammars is that if such a grammar allows us to delay the combination of a functor and its argument (via composition), then it also allows us to combine the functor and its argument immediately (via application). To illustrate this intuition, consider Figure 11, which shows two derivations related to the discussion of word order in Swiss German subordinate clauses (Shieber 1985):

…	mer	em Hans	es huus	hälfed	aastriche
…	we	Hans_{dat}	the house_{acc}	helped	paint

"… we helped Hans paint the house"

Derivation (5) (simplified from Steedman and Baldridge 2011, p. 201) starts by composing the tensed verb *hälfed* into the infinitive *aastriche* and then applies the resulting category to the accusative argument of the infinitive, *es huus*. Prefix-closedness implies that, if the combination of *hälfed* and *aastriche* is allowed when the latter is still waiting for *es huus*, then it must also be allowed if *es huus* has already been found.

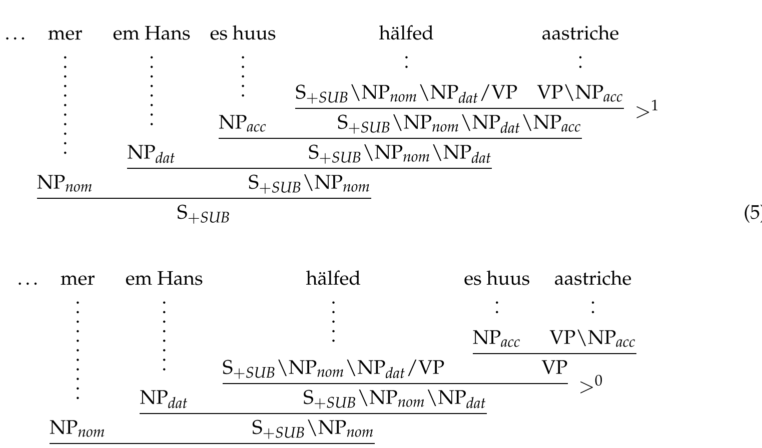


Figure 11
Prefix-closedness predicts different word orders for Swiss German subordinate clauses.

Thus prefix-closedness predicts derivation (6), and along with it the alternative word order

...	mer	em Hans	hälfed	es huus	aastriche
...	we	Hans_{dat}	helped	the house_{acc}	paint

This word order is in fact grammatical (Shieber 1985, pp. 338–339).

3.2 Generative Capacity of Prefix-Closed Grammars

We now show that the restriction to prefix-closed grammars does not change the generative capacity of VW-CCG.

Theorem 2
Prefix-closed VW-CCG and TAG are weakly equivalent.

Proof
Every prefix-closed VW-CCG is a VW-CCG, therefore the inclusion follows from Theorem 1. To show that every TAG language can be generated by a prefix-closed VW-CCG, we recall the construction of a weakly equivalent VW-CCG for a given TAG that we sketched in the proof of Lemma 3. As already mentioned in Example 6, the grammar G' constructed there is not prefix-closed. However, we can make it prefix-closed by explicitly allowing the "missing" rule instances:

$$A^a \,\square\, /B^c \quad B^c \quad \Rightarrow \quad A^a \,\square \qquad \text{where} \qquad A, B \in \mathcal{V}$$

$$B^c \quad A^a \,\square\, \backslash B^c \quad \Rightarrow \quad A^a \,\square \qquad \text{where} \qquad A, B \in \mathcal{V}$$

We shall now argue that this modification does not actually change the language generated by G'. The only categories that qualify as secondary input categories of the new instances are atomic categories of the form B^c where B is a nonterminal of the TAG G. Now the *lexical* categories of G' either are of the form x^a (where x is a terminal symbol) or are non-atomic. Categories of the form B^c are not among the *derived* categories of G' either, as the combinatory rules only yield output categories whose targets have the form B^a. This means that the new rule instances can never be used in a complete derivation of G', and therefore do not change the generated language. Thus we have a construction that turns a TAG into a weakly equivalent prefix-closed VW-CCG. ■

3.3 Prefix-Closed Grammars Without Target Restrictions

In this section we shall see that the weak equivalence between prefix-closed VW-CCG and TAG depends on the ability to restrict the target of the primary input category in a combinatory rule. These are the restrictions that we referred to as constraints of type (a) in Section 2.2. We say that a grammar that does not make use of these constraints is **without target restrictions**. This property can be formally defined as follows.

Definition 3
A VW-CCG is **without target restrictions** if it satisfies the following implication:

$$\begin{array}{rllll} \text{if} & X/Y & Y\beta & \Rightarrow & X\beta \quad \text{is a valid rule instance} \\ \text{then so is} & \bar{X}/Y & Y\beta & \Rightarrow & \bar{X}\beta \quad \text{for any category } \bar{X} \text{ of the grammar} \end{array}$$

and similarly for backward rules.

Example 8

1. Every AB-grammar is without target restrictions; it allows forward and backward application for every primary input category.
2. The grammar G_1 from Example (2) is *not* without target restrictions, because its rules are restricted to primary input categories with target S.

Target restrictions on the primary input category are useful in CCGs for natural languages; recall our discussion of backward-crossed composition in Section 1. As we shall see, target restrictions are also relevant from a formal point of view: If we require VW-CCGs to be without target restrictions, then we lose some of their weak generative capacity. This is the main technical result of this article. For its proof we need the following standard concept from formal language theory:

Definition 4
Two languages L and L' are **Parikh-equivalent** if for every string $w \in L$ there exists a permuted version w' of w such that $w' \in L'$, and vice versa.

Theorem 3
The languages generated by prefix-closed VW-CCG without target restrictions are properly included in the TAG languages.

Proof
Every prefix-closed VW-CCG without target restrictions is a VW-CCG, so the inclusion follows from Theorem 1. To see that the inclusion is proper, consider the TAG language $L_1 = \{a^n b^n c^n \mid n \geq 1\}$ from Example 5. We are interested in sublanguages $L' \subseteq L_1$ that are Parikh-equivalent to the full language L_1. This property is trivially satisfied by L_1 itself. Moreover, it is not hard to see that L_1 is in fact the *only* sublanguage of L_1 that has this property. Now in Section 3.4 we shall prove a central lemma (Lemma 6), which asserts that, if we assume that L_1 is generated by a prefix-closed VW-CCG without target restrictions, then at least one of the Parikh-equivalent sublanguages of L_1 must be context-free. Because L_1 is the only such sublanguage, this would give us proof that L_1 is context-free; but we know it is not. Therefore we conclude that L_1 is not generated by a prefix-closed VW-CCG without target restrictions. ■

Before turning to the proof of the central lemma (Lemma 6), we establish two other results about the languages generated by grammars without target restrictions.

Lemma 4
The languages generated by prefix-closed VW-CCG without target restrictions properly include the context-free languages.

Proof
Inclusion follows from the fact that AB-grammars (which generate all context-free languages) are prefix-closed VW-CCGs without target restrictions. To see that the inclusion is proper, consider a grammar G_2 that is like G_1 but does not have any rule restrictions. This grammar is trivially prefix-closed and without target restrictions; it is actually "pure" in the sense of Kuhlmann, Koller, and Satta (2010). The language $L_2 = L(G_2)$ contains all the strings in $L_1 = \{a^n b^n c^n \mid n \geq 1\}$, together with other strings, including the string *bbbacacac*, whose derivation we showed in Figure 7. It is not hard to see that all of these additional strings have an equal number of *a*s, *b*s, and *c*s. We can therefore write L_1 as an intersection of L_2 and a regular language: $L_1 = L_2 \cap a^* b^* c^*$. To obtain a contradiction, suppose that L_2 is context-free; then because of the fact that context-free languages are closed under intersection with regular languages, the language L_1 would be context-free as well—but we know it is not. Therefore we conclude that L_2 is not context-free either. ■

Lemma 5
The class of languages generated by prefix-closed VW-CCG without target restrictions is not closed under intersection with regular languages.

Proof
If the class of languages generated by prefix-closed VW-CCG without target restrictions was closed under intersection with regular languages, then with L_2 (the language mentioned in the previous proof) it would also include the language $L_1 = L_2 \cap a^* b^* c^*$. However, from the proof of Theorem 3 we know that L_1 is not generated by any prefix-closed VW-CCG without target restrictions. ■

3.4 Proof of the Main Lemma for VW-CCG

We shall now prove the central lemma that we used in the proof of Theorem 3.

Lemma 6 (Main Lemma for VW-CCG)
For every language L that is generated by some prefix-closed VW-CCG without target restrictions, there is a sublanguage $L' \subseteq L$ such that

1. L' and L are Parikh-equivalent, and
2. L' is context-free.

Throughout this section, we let G be some arbitrary prefix-closed VW-CCG without target restrictions. The basic idea is to transform the derivations of G into a certain special form, and to prove that the transformed derivations yield a context-free language. The transformation is formalized by the rewriting system in Figure 12.[4] To see how the rules of this system work, consider rule R1; the other rules are symmetric. Rule R1 rewrites an entire derivation into another derivation. It states that, whenever we have a situation where a category of the form X/Y is combined with a category of the form $Y\beta/Z$ by means of composition, and the resulting category is combined with a category Z by means of application, then we may just as well first combine $Y\beta/Z$ with Z, and then use the resulting category as a secondary input category together with X/Y.

4 Recall that we use the Greek letter β to denote a (possibly empty) sequence of arguments.

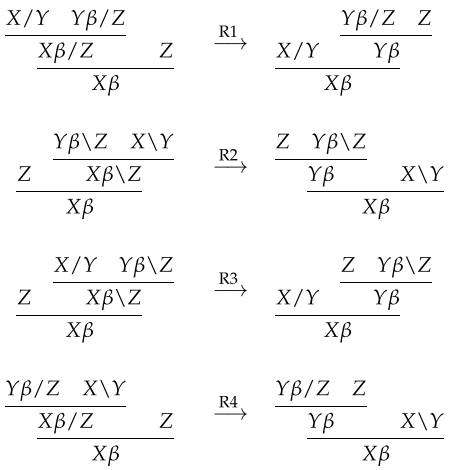
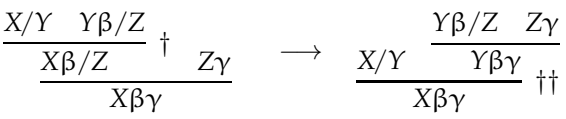


Figure 12
Rewriting rules used in the transformation.

Note that R1 and R2 produce a new derivation for the original sentence, whereas R3 and R4 produce a derivation that yields a permutation of that sentence: The order of the substrings corresponding to the categories Z and X/Y (in the case of rule R3) or $X\backslash Y$ (in the case of rule R4) is reversed. In particular, R3 captures the relation between the two derivations of Swiss German word orders shown in Figure 11: Applying R3 to derivation (5) gives derivation (6). Importantly though, while the transformation may reorder the yield of a derivation, every transformed derivation still is a derivation of G.

Example 9
If we take the derivation in Figure 6 and exhaustively apply the rewriting rules from Figure 12, then the derivation that we obtain is the one in Figure 7. Note that although the latter derivation is not grammatical with respect to the grammar G_1 from Example 2, it *is* grammatical with respect to the grammar G_2 from the proof of Lemma 4, which is without target restrictions.

It is instructive to compare the rewriting rules in Figure 12 to the rules that establish the normal form of Eisner (1996). This normal form is used in practical CCG parsers to solve the problem of "spurious ambiguity," where one and the same semantic interpretation (which in CCG takes the form of a lambda term) has multiple syntactic derivation trees. It is established by rewriting rules such as the following:



(5)

The rules in Figure 12 have much in common with the Eisner rules; yet there are two important differences. First, as already mentioned, our rules (in particular, rules R3 and R4) may reorder the yield of a derivation, whereas Eisner's normal form

preserves yields. Second, our rules *decrease* the degrees of the involved composition operations, whereas Eisner's rules may in fact *increase* them. To see this, note that the left-hand side of derivation (7) involves a composition of degree $|\beta| + 1$ (†), whereas the right-hand side involves a composition of degree $|\beta| + |\gamma|$ (††). This means that rewriting will increase the degree in situations where $|\gamma| > 1$. In contrast, our rules only fire in the case where the combination with Z happens by means of an application, that is, if $|\gamma| = 0$. Under this condition, each rewrite step is guaranteed to decrease the degree of the composition. We will use this observation in the proof of Lemma 7.

3.4.1 Properties of the Transformation. The next two lemmas show that the rewriting system in Figure 12 implements a total function on the derivations of G.

Lemma 7
The rewriting system is terminating and confluent: Rewriting a derivation ends after a finite number of steps, and different rewriting orders all result in the same output.

Proof
To argue that the system is terminating, we note that each rewriting step decreases the arity of one secondary input category in the derivation by one unit, while all other secondary input categories are left unchanged. As an example, consider rewriting under R1. The secondary input categories in the scope of that rule are $Y\beta/Z$ and Z on the left-hand side and $Y\beta$ and Z on the right-hand side. Here the arity of $Y\beta$ equals the arity of $Y\beta/Z$, minus one. Because the system is terminating, to see that it is also confluent, it suffices to note that the left-hand sides of the rewrite rules do not overlap. ■

Lemma 8
The rewriting system transforms derivations of G into derivations of G.

Proof
We prove the stronger result that every rewriting step transforms derivations of G into derivations of G. We only consider rewriting under R1; the arguments for the other rules are similar. Assume that R1 is applied to a derivation of G. The rule instances in the scope of the left-hand side of R1 take the following form:

$$X/Y \quad Y\beta/Z \quad \Rightarrow \quad X\beta/Z \tag{8}$$

$$X\beta/Z \quad Z \quad \Rightarrow \quad X\beta \tag{9}$$

Turning to the right-hand side, the rule instances in the rewritten derivation are

$$Y\beta/Z \quad Z \quad \Rightarrow \quad Y\beta \tag{10}$$

$$X/Y \quad Y\beta \quad \Rightarrow \quad X\beta \tag{11}$$

The relation between instances (8) and (11) is the characteristic relation of prefix-closed grammars (Definition 2): If instance (8) is valid, then because G is prefix-closed,

instance (11) is valid as well. Similarly, the relation between instances (9) and (10) is the characteristic relation of grammars without target restrictions (Definition 3): If instance (9) is valid, then because G is without target restrictions, instance (10) is valid as well. We conclude that if R1 is applied to a derivation of G, then the result is another derivation of G. ■

Combining Lemma 7 and Lemma 8, we see that for every derivation d of G, exhaustive application of the rewriting rules produces another uniquely determined derivation of G. We shall refer to this derivation as $R(d)$. A **transformed derivation** is any derivation d' such that $d' = R(d)$ for some derivation d.

3.4.2 Language Inclusion and Parikh-Equivalence.

Lemma 9
The yields of the transformed derivations are a subset of and Parikh-equivalent to $L(G)$.

Proof
Let $\mathcal{Y}$ be the set of yields of the transformed derivations. Every string $w' \in \mathcal{Y}$ is obtained from a string $w \in L(G)$ by choosing some derivation d of w, rewriting this derivation into the transformed derivation $R(d)$, and taking the yield. Inclusion then follows from Lemma 8. Because of the permuting rules R3 and R4, the strings w and w' will in general be different. What we can say, however, is that w and w' will be equal up to permutation. Thus we have established that $\mathcal{Y}$ and $L(G)$ are Parikh-equivalent. ■

What remains in order to prove Lemma 6 is to show that the yields of the transformed derivations form a context-free language.

3.4.3 Context-Freeness of the Sublanguage. In a derivation tree, every node except the root node is labeled with either the primary or the secondary input category of a combinatory rule. We refer to these two types of nodes as **primary nodes** and **secondary nodes**, respectively. To simplify our presentation, we shall treat the root node as a secondary node. We restrict our attention to derivation trees for strings in $L(G)$; in these trees, the root node is labeled with the distinguished atomic category S. For a leaf node u, the **projection path** of u is the path that starts at the parent of u and ends at the first secondary node that is encountered on the way towards the root node. We denote a projection path as a sequence $X_1, \ldots, X_n$ ($n \geq 1$), where X_1 is the category at the parent of u and X_n is the category at the secondary node. Note that the category X_1 is taken from the lexicon, while every other category is derived by combining the preceding category on the path with some secondary input category (not on the path) by means of some combinatory rule.

Example 10
In the derivation in Figure 6, the projection path of the first b goes all the way to the root, while all other projection paths have length 1, starting and ending with a lexical category. In Figure 7, the projection path of the first b ends at the root, while the projection paths of the remaining bs end at the nodes with category B, and the projection paths of the cs end at the nodes with category C.

A projection path $X_1, \ldots, X_n$ is **split** if it can be segmented into two parts

$$X_1, \ldots, X_s \qquad \text{and} \qquad X_s, \ldots, X_n \qquad (1 \leq s \leq n)$$

such that the first part only uses application rules and the second part only uses composition rules. Note that any part may consist of a single category only, in which case no combinatory rule is used in that part. If $n = 1$, then the path is trivially split. All projection paths in Figures 6 and 7 are split, except for the path of the first b in Figure 6, which alternates between composition (with $C\backslash A$) and application (with A).

Lemma 10
In transformed derivations, every projection path is split.

Proof
We show that as long as a derivation d contains a projection path that is not split, it can be rewritten. A projection path that is not split contains three adjacent categories U, V, W, such that V is derived by means of a *composition* with primary input U, and W is derived by means of an *application* with primary input V. Suppose that both the composition and the application are forward. (The arguments for the other three cases are similar.) Then U can be written as X/Y for some category X and argument $/Y$, V can be written as $X\beta/Z$ for some argument $/Z$ and some (possibly empty) sequence of arguments β, and W can be written as $X\beta$. We can then convince ourselves that d contains the following configuration, which matches the left-hand side of rewriting rule R1:

$$\dfrac{\dfrac{X/Y \quad Y\beta/Z}{X\beta/Z} \qquad Z}{X\beta}$$

■

Lemma 11
The set of all categories that occur in transformed derivations is finite.

Proof
Every category that occurs in transformed derivations occurs on some of its projection paths. Consider any such path. By Lemma 10 we know that this path is split; its two parts, here called P_1 and P_2, are visualized in Figure 13. We now reason about the arities of the categories in these two parts.

1. Because P_1 only uses application, the arities in this part get smaller and smaller until they reach their minimum at X_s. This means that the arities of P_1 are bounded by the arity of the first category on the path, which is a category from the lexicon.
2. Because P_2 only uses composition, the arities in this part either get larger or stay the same until they reach a maximum at X_n. This means that the arities of P_2 are bounded by the arity of the last category on the path, which is either the distinguished atomic category S or a secondary input category.

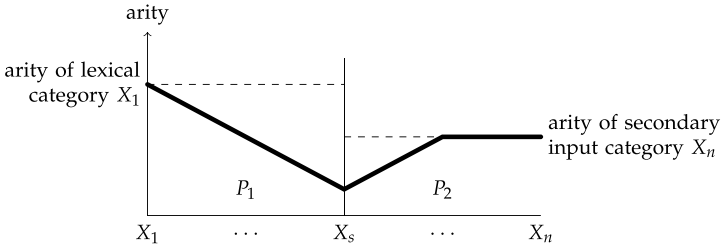


Figure 13
Illustration of the argument used in the proof of Lemma 11.

Thus the arities of our chosen path are bounded by the maximum of three grammar-specific constants: the maximal arity of a lexical category, the arity of S (which is 0), and the maximal arity of a secondary input category. The latter value is well-defined because there are only finitely many such categories (by Lemma 2). Let k be the maximum among the three constants, and let $\mathcal{K}$ be the set of all categories of the form $A|_m X_m \cdots |_1 X_1$ where A is an atomic category of G, $m \leq k$, and each $|_i X_i$ is an argument that may occur in derivations of G. The set $\mathcal{K}$ contains all categories that occur on some projection path, and therefore all categories that occur in transformed derivations, but it may also include other categories. As there are only finitely many atomic categories and finitely many arguments (Lemma 1), we conclude that the set $\mathcal{K}$, and hence the set of categories that occur in transformed derivations, are finite as well. ■

Lemma 12
The transformed derivations yield a context-free language.

Proof
We construct a context-free grammar H that generates the set $\mathcal{Y}$ of yields of the transformed derivations. To simplify the presentation, we first construct a grammar H' that generates a superset of $\mathcal{Y}$.

Construction of H'. The construction of the grammar H' is the same as the construction in the classical proof that showed the context-freeness of AB-grammars, by Bar-Hillel, Gaifman, and Shamir (1960): The production rules of H' are set up to correspond to the valid rule instances of G. The reason that this construction is not useful for VW-CCGs in general is that these may admit infinitely many rule instances, whereas a context-free grammar can only have finitely many productions. The set of rule instances may be infinite because VW-CCG has access to composition rules (specifically, rules of degrees greater than 1); in contrast, AB-grammars are restricted to application. Crucially though, by Lemma 11 we know that as long as we are interested only in *transformed* derivations it is sufficient to use a *finite* number of rule instances—more specifically those whose input and output categories are included in the set $\mathcal{K}$ of arity-bounded categories. Thus for every instance $X/Y \;\; Y\beta \;\Rightarrow\; X\beta$ where all three categories are in $\mathcal{K}$, we construct a production

$$[X\beta] \rightarrow [X/Y]\,[Y\beta]$$

and similarly for backward rules. (We enclose categories in square brackets for clarity.) In addition, for every lexicon entry $\sigma := X$ in G we add to H' a production $[X] \to \sigma$. As the terminal alphabet of H' we choose the vocabulary of G; as the nonterminal alphabet we choose the set $\mathcal{K}$; and as the start symbol we choose the distinguished atomic category S. Every transformed derivation of G corresponds (in an obvious way) to some derivation in H', which proves that $\mathcal{Y} \subseteq L(H')$. Conversely, every derivation of H' represents a derivation of G (though not necessarily a transformed derivation), thus $L(H') \subseteq L(G)$.

Construction of H. The chain of inclusions $\mathcal{Y} \subseteq L(H') \subseteq L(G)$ is sufficient to prove Lemma 6: Because $\mathcal{Y}$ and $L(G)$ are Parikh-equivalent (which we observed at the beginning of Section 3.4.2), so are $L(H')$ and $L(G)$, which means that $L(H')$ satisfies all of the properties claimed in Lemma 6, even though this does not suffice to prove our current lemma. However, once H' is given, it is not hard to also obtain a grammar H that generates *exactly* $\mathcal{Y}$. For this, we need to filter out derivations whose projection paths do not have the characteristic property of transformed derivations that we established in Lemma 10. (It is not hard to see that every derivation that *does* have this property is a transformed derivation.) We annotate the left-hand side nonterminals in the productions of H' with a flag $t \in \{a, c\}$ to reflect whether the corresponding category has been derived by means of application ($t = a$) or composition ($t = c$); the value of this flag is simply the type of combinatory rule that gave rise to the production. The nonterminals in the right-hand sides are annotated in all possible ways, except that the following combinations are ruled out:

$$[X]_a \to [X/Y]_c[Y]_t \quad \text{and} \quad [X]_a \to [Y]_t[X\backslash Y]_c \qquad\qquad t \in \{a, c\}$$

These combinations represent exactly the cases where the output category of a composition rule is used as the primary input category of an application rule, which are the cases that violate the "split" property that we established in Lemma 10. ■

This concludes the proof of Lemma 6, and therefore the proof of Theorem 3.

3.5 Discussion

Theorem 3 pinpoints the exact mechanism that VW-CCG uses to achieve weak equivalence to TAG: At least for the class of prefix-closed grammars, TAG equivalence is achieved if and only if we allow target restrictions. Although target restrictions are frequently used in linguistically motivated grammars, it is important and perhaps surprising to realize that they are indeed *necessary* to achieve the full generative capacity of VW-CCG.

In the grammar formalisms folklore, the generative capacity of CCG is often attributed to generalized composition, and indeed we have seen (in Lemma 4) that even grammars without target restrictions can generate non-context-free languages such as $L(G_2)$. However, our results show that composition by itself is not enough to achieve weak equivalence with TAG: The yields of the transformed derivations from Section 3.4 form a context-free language *despite* the fact that these derivations may still contain compositions, including compositions of degree $n > 2$. In addition to composition, VW-CCG also needs target restrictions to exert enough control on word order to block unwanted permutations. One way to think about this is that target restrictions can enforce alternations of composition and application (as in the derivation shown in

Figure 6), while transformed derivations are characterized by projection paths without such alternations (Lemma 10).

We can sharpen the picture even more by observing that the target restrictions that are crucial for the generative capacity of VW-CCG are not those on generalized composition, but those on function application. To see this we can note that the proof of Lemma 8 goes through also only if application rules such as (9) and (10) are without target restrictions. This means that we have the following qualification of Theorem 1.

Lemma 13
Prefix-closed VW-CCG is weakly equivalent to TAG only because it supports target restrictions on forward and backward application.

This finding is unexpected indeed—for instance, no grammar in Steedman (2000) uses target restrictions on the application rules.

4. Generative Capacity of Multimodal CCG

After clarifying the mechanisms that "classical" CCG uses to achieve weak equivalence with TAG, we now turn our attention to "modern," multimodal versions of CCG (Baldridge and Kruijff 2003; Steedman and Baldridge 2011). These versions emphasize the use of fully lexicalized grammars in which no rule restrictions are allowed, and instead equip slashes with *types* in order to control the use of the combinatory rules. Our central question is whether the use of slash types is sufficient to recover the expressiveness that we lose by giving up rule restrictions.

We need to fix a specific variant of multimodal CCG to study this question formally. Published works on multimodal CCG differ with respect to the specific inventories of slash types they assume. Some important details, such as a precise definition of generalized composition with slash types, are typically not discussed at all. In this article we define a variant of multimodal CCG which we call O-CCG. This formalism extends our definition of VW-CCG (Definition 1) with the slash inventory and the composition rules of the popular OpenCCG grammar development system (White 2013). Our technical result is that the main Lemma (Lemma 6) also holds for O-CCG. With this we can conclude that the answer to our question is negative: Slash types are *not* sufficient to replace rule restrictions; O-CCG is strictly less powerful than TAG. Although this is primarily a theoretical result, at the end of this section we also discuss its implications for practical grammar development.

4.1 Multimodal CCG

We define O-CCG as a formalism that extends VW-CCG with the slash types of OpenCCG, but abandons rule restrictions. Note that OpenCCG has a number of additional features that affect the generative capacity; we discuss these in Section 4.4.

Slash Types. Like other incarnations of multimodal CCG, O-CCG uses an enriched notion of categories where every slash has a **type**. There are eight such types:[5]

core types: $*$ $\diamond$ $\times$ $\cdot$ left types: $\triangleleft$ $\triangleleft\times$ right types: $\triangleright$ $\times\triangleright$

5 The type system of OpenCCG is an extension of the system used by Baldridge (2002).

	$*$	$\diamond$	$\times$	$\triangleleft\times$	$\times\triangleright$	$\triangleleft$	$\triangleright$	$\cdot$
application	✓	✓	✓	✓	✓	✓	✓	✓
harmonic composition		✓				✓	✓	✓
crossed composition			✓	✓	✓	✓	✓	✓

Figure 14
Compatibility of slash types with combinatory rules.

The basic idea behind these types is as follows. Slashes with type $*$ can only be used to instantiate application rules. Type $\diamond$ also licenses *harmonic* composition rules, and type $\times$ also licenses *crossed* composition rules. Type $\cdot$ is the least restrictive type and can be used to instantiate all rules. The remaining types refine the system by incorporating a dimension of directionality. The exact type–rule compatibilities are specified in Figure 14.

Inertness. O-CCG is distinguished from other versions of multimodal CCG, such as that of Baldridge and Kruijff (2003), in that every slash not only has a type but also an **inertness status**. Inertness was introduced by Baldridge (2002, Section 8.2.2) as an implementation of the "antecedent government" (ANT) feature of Steedman (1996), which is used to control the word order in certain English relative clauses. It is a two-valued feature. Arguments whose slash type has inertness status $+$ are called **active**; arguments whose slash type has inertness status $-$ are called **inert**. Only active arguments can be eliminated by means of combinatory rules; however, an inert argument can still be consumed as part of a secondary input category. For example, the following instance of application is valid because the outermost slash of the primary input category has inertness status $+$:

$$X/^{+}(Y/^{-}Z) \quad Y/^{-}Z \quad \Rightarrow \quad X$$

We use the notations $/^{s}_{t}$ and $\backslash^{s}_{t}$ to denote the forward and backward slashes with slash type t and inertness status s.

Rules. All O-CCG grammars share a fixed set of combinatory rules, shown in Figure 15. Every grammar uses all rules, up to some grammar-specific bound on the degree of generalized composition. As mentioned earlier, a combinatory rule can only be instantiated if the slashes of the input categories have compatible types. Additionally, all composition rules require the slashes of the secondary input category to have a uniform direction. This is a somewhat peculiar feature of OpenCCG, and is in contrast to VW-CCG and other versions of CCG, which also allow composition rules with mixed directions.

Composition rules are classified into **harmonic** and **crossed** forms. This distinction is based on the direction of the slashes in the secondary input category. If these have the same direction as the outermost slash of the primary input category, then the rule is called harmonic; otherwise it is called crossed.[6]

6 In versions of CCG that allow rules with mixed slash directions, the distinction between harmonic and crossed is made based on the direction of the *innermost* slash of the secondary input category, $|_i$.

Application $[*]\ [\diamond]\ [\times]\ [\lhd\times]\ [\times\rhd]\ [\lhd]\ [\rhd]\ [\cdot]$

$$X/^{+}_{t} Y \quad Y \quad \Rightarrow \quad X \qquad (>)$$

$$Y \quad X\backslash^{+}_{t} Y \quad \Rightarrow \quad X \qquad (<)$$

Harmonic composition $(n \geq 1)$ $[\diamond]\ [\lhd]\ [\rhd]\ [\cdot]$

$$X/^{+}_{t_0} Y \quad Y/^{s_n}_{t_n} Z_n \cdots /^{s_1}_{t_1} Z_1 \quad \Rightarrow \quad X/^{s_n}_{t_n} Z_n \cdots /^{s_1}_{t_1} Z_1 \qquad (>\mathbf{B}^n)$$

$$Y\backslash^{s_n}_{t_n} Z_n \cdots \backslash^{s_1}_{t_1} Z_1 \quad X\backslash^{+}_{t_0} Y \quad \Rightarrow \quad X\backslash^{s_n}_{t_n} Z_n \cdots \backslash^{s_1}_{t_1} Z_1 \qquad (<\mathbf{B}^n)$$

Crossed composition $(n \geq 1)$ $[\times]\ [\lhd\times]\ [\times\rhd]\ [\lhd]\ [\rhd]\ [\cdot]$

$$X/^{+}_{t_0} Y \quad Y\backslash^{s_n}_{t_n} Z_n \cdots \backslash^{s_1}_{t_1} Z_1 \quad \Rightarrow \quad X\backslash^{\sigma(s_n)}_{t_n} Z_n \cdots \backslash^{\sigma(s_1)}_{t_1} Z_1 \qquad (>\mathbf{B}^n_\times)$$

where $\sigma(s_i) = s_i$ if $\text{right}(t_0)$ and $\text{left}(t_1), \ldots, \text{left}(t_n)$; $\sigma(s_i) = -$ otherwise

$$Y/^{s_n}_{t_n} Z_n \cdots /^{s_1}_{t_1} Z_1 \quad X\backslash^{+}_{t_0} Y \quad \Rightarrow \quad X/^{\sigma(s_n)}_{t_n} Z_n \cdots /^{\sigma(s_1)}_{t_1} Z_1 \qquad (<\mathbf{B}^n_\times)$$

where $\sigma(s_i) = s_i$ if $\text{left}(t_0)$ and $\text{right}(t_1), \ldots, \text{right}(t_n)$; $\sigma(s_i) = -$ otherwise

Figure 15
The rules of O-CCG. For a rule to apply, all slashes in the scope of the rule must have one of the specified compatible types (cf. Figure 14). The predicates left(t)/right(t) are true if and only if t is equal to either a left/right type or one of the four undirected core types.

When a rule is applied, in most cases the arguments of the secondary input category are simply copied into the output category, as in VW-CCG. The one exception happens for crossed composition rules if not all slash directions match the direction of their slash type (left or right). In this case, the arguments of the secondary input category become inert. Thus the inertness status of an argument may change over the course of a derivation—but only from active to inert, not back again.

Definition 5
A **multimodal combinatory categorial grammar** in the sense of OpenCCG, or O-CCG for short, is a structure

$$G = (\Sigma, \mathcal{A}, :=, d, S)$$

where Σ is a finite vocabulary, $\mathcal{A}$ is a finite set of atomic categories, $:=$ is a finite relation between Σ and the set of (multimodal) categories over $\mathcal{A}$, $d \geq 0$ is the maximal degree of generalized composition, and $S \in A$ is a distinguished atomic category.

We generalize the notions of rule instances, derivation trees, and generated language to categories over slashes with types and inertness statuses in the obvious way: Instead of two slashes, we now have one slash for every combination of a direction, type, and inertness status. Similarly, we generalize the concepts of a grammar being prefix-closed (Definition 2) and without target restrictions (Definition 3) to O-CCG.

4.2 Generative Capacity

We now investigate the generative capacity of O-CCG. We start with the (unsurprising) observation that O-CCG can describe non-context-free languages.

Lemma 14
The languages generated by O-CCG properly include the context-free languages.

Proof
Inclusion follows from the fact that every AB-grammar can be written as an O-CCG with only application ($d = 0$). To show that the inclusion is proper, we use the same argument as in the proof of Lemma 4. The grammar G_2 that we constructed there can be turned into an equivalent O-CCG by decorating each slash with $\cdot$, the least restrictive type, and setting its inertness status to $+$. ■

What is less obvious is whether O-CCG generates the same class of languages as VW-CCG and TAG. Our main result is that this is not the case.

Theorem 4
The languages generated by O-CCG are properly included in the TAG languages.

O-CCG without Inertness. To approach Theorem 4, we set inertness aside for a moment and focus on the use of the slash types as a mechanism for imposing rule restrictions. Each of the rules in Figure 15 requires all of the slash types of the n outermost arguments of its secondary input category to be compatible with the rule, in the sense specified in Figure 14. If we now remove one or more of these arguments from a valid rule instance, then the new instance is clearly still valid, as we have reduced the number of potential violations of the type–rule compatibility. This shows that the rule system is prefix-closed. As none of the rules is conditioned on the target of the primary input category, the rule system is even without target restrictions. With these two properties established, Theorem 4 can be proved by literally the same arguments as those that we gave in Section 3. Thus we see directly that the theorem holds for versions of multimodal CCG without inertness, such as the formalism of Baldridge and Kruijff (2003).

O-CCG with Inertness. In the general case, the situation is complicated by the fact that the crossed composition rules change the inertness status of some argument categories if the slash types have conflicting directions. This means that the crossed composition rules in O-CCG are not entirely prefix-closed, as illustrated by the following example.

Example 11
Consider the following two rule instances:

$$X/^{+}_{\times\triangleright} Y \quad Y\backslash^{+}_{\triangleleft\times} Z_2 \backslash^{+}_{\times\triangleright} Z_1 \quad \Rightarrow \quad X\backslash^{-}_{\triangleleft\times} Z_2 \backslash^{-}_{\times\triangleright} Z_1 \tag{12}$$

$$X/^{+}_{\times\triangleright} Y \quad Y\backslash^{+}_{\triangleleft\times} Z_2 \quad \Rightarrow \quad X\backslash^{-}_{\triangleleft\times} Z_2 \tag{13}$$

Instance (12) is a valid instance of forward crossed composition. Prefix-closedness would require instance (13) to be valid as well; but it is not. In instance (12) the inertness status of $\backslash^{+}_{\triangleleft\times} Z_2$ is changed for the only reason that the slash type of $\backslash^{+}_{\times\triangleright} Z_1$ does not match the required direction. In instance (13) the argument $\backslash^{+}_{\times\triangleright} Z_1$ is not present, and

therefore the inertness status of $\backslash^{+}_{\lhd\times} Z_2$ is not changed, but is carried over to the output category.

We therefore have to prove that the following analogue of Lemma 6 holds for O-CCG:

Lemma 15 (Main Lemma for O-CCG)
For every language L generated by some O-CCG there is a sublanguage $L' \subseteq L$ such that

1. L' and L are Parikh-equivalent, and
2. L' is context-free.

This lemma implies that the language $L_1 = \{a^n b^n c^n \mid n \geq 1\}$ from Example 2 cannot be generated by O-CCG (but by prefix-closed VW-CCG with target restrictions, and by TAG). The argument is the same as in the proof of Theorem 3.

4.3 Proof of the Main Lemma for O-CCG

The proof of Lemma 15 adapts the rewriting system from Figure 12. We simply let each rewriting step copy the type and inertness status of each slash from the left-hand side to the right-hand side of the rewriting rule. With this change, it is easy to verify that the proofs of Lemma 7 (termination and confluence), Lemma 10 (projection paths in transformed derivations are split), Lemma 11 (transformed derivations contain a finite number of categories), and Lemma 12 (transformed derivations yield a context-free language) go through without problems. The proof of Lemma 8, however, is not straightforward, because of the dynamic nature of the inertness statuses. We therefore restate the lemma for O-CCG:

Lemma 16
The rewriting system transforms O-CCG derivations into O-CCG derivations.

Proof
As in the proof of Lemma 8 we establish the stronger result that the claimed property holds for every single rewriting step. We only give the argument for rewriting under R3, which involves instances of forward crossed composition. The argument for R4 is analogous, and R1 and R2 are simpler cases because they involve harmonic composition, where the inertness status does not change.

Suppose that R3 is applied to a derivation of some O-CCG. In their most general form, the rule instances in the scope of the left-hand side of R3 may be written as follows, where the function σ is defined as specified in Figure 15:

$$X/^{+}_{t_0} Y \quad Y\backslash^{s_n}_{t_n} Z_n \cdots \backslash^{s_2}_{t_2} Z_2 \backslash^{s_1}_{t_1} Z_1 \quad \Rrightarrow \quad X\backslash^{\sigma(s_n)}_{t_n} Z_n \cdots \backslash^{\sigma(s_2)}_{t_2} Z_2 \boxed{\backslash^{\sigma(s_1)}_{t_1} Z_1} \tag{14}$$

$$Z_1 \quad X\backslash^{\sigma(s_n)}_{t_n} Z_n \cdots \backslash^{\sigma(s_2)}_{t_2} Z_2 \boxed{\backslash^{+}_{t_1} Z_1} \quad \Rrightarrow \quad X\backslash^{\sigma(s_n)}_{t_n} Z_n \cdots \backslash^{\sigma(s_2)}_{t_1} Z_2 \tag{15}$$

Here instance (14) is an instance of forward-crossed composition, so each of the types t_i is compatible with that rule. Because the two marked arguments are identical, we have $\sigma(s_1) = +$. This is only possible if the inertness statuses of the slashes $\backslash_{t_i}^{s_i}$ do not change in the context of derivation (14), that is, if $\sigma(s_i) = s_i$ for all $1 \leq i \leq n$. Note that in this case, t_0 is either a right type or one of the four undirected core types, and each $t_1, \ldots, t_n$ is either a left type or a core type. We can now alternatively write instances (14) and (15) as

$$X/_{t_0}^{+}Y \quad Y\backslash_{t_n}^{s_n}Z_n \cdots \backslash_{t_2}^{s_2}Z_2\backslash_{t_1}^{+}Z_1 \quad \Rightarrow \quad X\backslash_{t_n}^{s_n}Z_n \cdots \backslash_{t_2}^{s_2}Z_2\backslash_{t_1}^{+}Z_1 \tag{14'}$$

$$Z_1 \quad X\backslash_{t_n}^{s_n}Z_n \cdots \backslash_{t_2}^{s_2}Z_2\backslash_{t_1}^{+}Z_1 \quad \Rightarrow \quad X\backslash_{t_n}^{s_n}Z_n \cdots \backslash_{t_1}^{s_2}Z_2 \tag{15'}$$

Then the rule instances in the rewritten derivation can be written as follows:

$$Z_1 \quad Y\backslash_{t_n}^{s_n}Z_n \cdots \backslash_{t_2}^{s_2}Z_2\backslash_{t_1}^{+}Z_1 \quad \Rightarrow \quad Y\backslash_{t_n}^{s_n}Z_n \cdots \backslash_{t_2}^{s_2}Z_2 \tag{16}$$

$$X/_{t_0}^{+}Y \quad Y\backslash_{t_n}^{s_n}Z_n \cdots \backslash_{t_2}^{s_2}Z_2 \quad \Rightarrow \quad X\backslash_{t_n}^{s_n}Z_n \cdots \backslash_{t_2}^{s_2}Z_2 \tag{17}$$

Here instance (16) is clearly a valid instance of backward application. Based on our earlier observations about the t_i and their compatibility with crossed composition, we also see that instance (17) is a valid instance of forward crossed composition (if $n > 1$), or of forward application (if $n = 1$). ■

This completes the proof of Lemma 15. To finish the proof of Theorem 4 we have to also establish the inclusion of the O-CCG languages in the TAG languages. This is a known result for other dialects of multimodal CCG (Baldridge and Kruijff 2003), but O-CCG once again requires some extra work because of inertness.

Lemma 17
The O-CCG languages are included in the TAG languages.

Proof (Sketch)
It suffices to show that the O-CCG languages are included in the class of languages generated by LIG (Gazdar 1987); the claim then follows from the weak equivalence of LIG and TAG. Vijay-Shanker and Weir (1994, Section 3.1) present a construction that transforms an arbitrary VW-CCG into a weakly equivalent LIG. It is straightforward to adapt their construction to O-CCG. As we do not have the space here to define LIG, we only provide a sketch of the adapted construction.

As in the case of VW-CCG, the valid instances of an O-CCG rule can be written down using our template notation. The adapted construction converts each such template into a production rule of a weakly equivalent LIG. Consider for instance the following instance of forward crossed composition from Example 11:

$$A\,\square/_{\times\rhd}^{+}Y \quad Y\backslash_{\lhd\times}^{+}Z_2\backslash_{\times\rhd}^{+}Z_1 \quad \Rightarrow \quad A\,\square\backslash_{\lhd\times}^{-}Z_2\backslash_{\times\rhd}^{-}Z_1$$

This template is converted into the following LIG rule. We adopt the notation of Vijay-Shanker and Weir (1994) and write $\circ\circ$ for the tail of a stack of unbounded size.

$$A[\circ\circ \backslash^{-}_{\lhd\times} Z_2 \backslash^{-}_{\times\rhd} Z_1] \quad \rightarrow \quad A[\circ\circ /^{+}_{\times\rhd} Y] \; Y[\backslash^{+}_{\lhd\times} Z_2 \backslash^{+}_{\times\rhd} Z_1]$$

In this way, every O-CCG can be written as a weakly equivalent LIG. ■

4.4 Discussion

In this section we have shown that the languages generated by O-CCG are properly included in the languages generated by TAG, and equivalently, in the languages generated by VW-CCG. This means that the multimodal machinery of OpenCCG is not powerful enough to express the rule restrictions of VW-CCG in a fully lexicalized way. The result is easy to obtain for O-CCG without inertness, which is prefix-closed and without target restrictions; but it is remarkably robust in that it also applies to O-CCG with inertness, which is not prefix-closed. As we have already mentioned, the result carries over also to other multimodal versions of CCG, such as the formalism of Baldridge and Kruijff (2003).

Our result has implications for practical grammar development with OpenCCG. To illustrate this, recall Example 7, which showed that every VW-CCG without target restrictions for Swiss German that allows cross–serial word orders as in derivation (5) also permits alternative word orders, as in derivation (6). By Lemma 15, this remains true for O-CCG or weaker multimodal formalisms. This is not a problem in the case of Swiss German, where the alternative word orders are grammatical. However, there is at least one language, Dutch, where dependencies in subordinate clauses *must* cross. For this case, our result shows that the modalized composition rules of OpenCCG are not powerful enough to write adequate grammars. Consider the following classical example:

…	ik	Cecilia	de paarden	zag	voeren
…	I	Cecilia	the horses	saw	feed

"… I saw Cecilia feed the horses"

The straightforward derivation of the cross–serial dependencies in this sentence (adapted from Steedman 2000, p. 141) is exemplified in Figure 16. It takes the same form as derivation (5) for Swiss German: The verbs and their NP arguments lie on a single, right-branching path projected from the tensed verb *zag*. This projection path is not split; specifically, it starts with a composition that produces a category which acts as the primary input category of an application. As a consequence, the derivation can be transformed (by our rewriting rule R3) in exactly the same way as instance (5) could be transformed into derivation (6). The crucial difference is that the yield of the transformed derivation, **ik Cecilia zag de paarden voeren*, is not a grammatical clause of Dutch.

To address the problem of ungrammatical word orders in Dutch subordinate clauses, the VW-CCG grammar of Steedman (2000) and the multimodal CCG grammar of Baldridge (2002, Section 5.3.1) resort to combinatorial rules other than composition. In particular, they assume that all complement noun phrases undergo *obligatory* type-raising, and become primary input categories of application rules. This gives rise to derivations such as the one shown in Figure 17, which cannot be transformed using our rewriting rules because the result of the forward crossed composition $>^1$

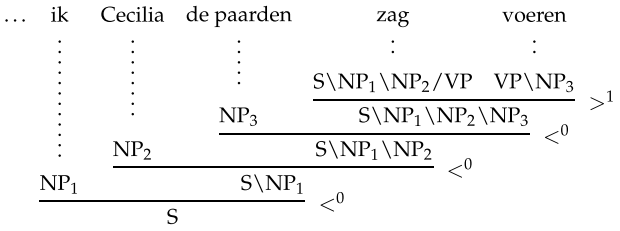


Figure 16
Straightforward derivation of Dutch cross–serial dependencies.

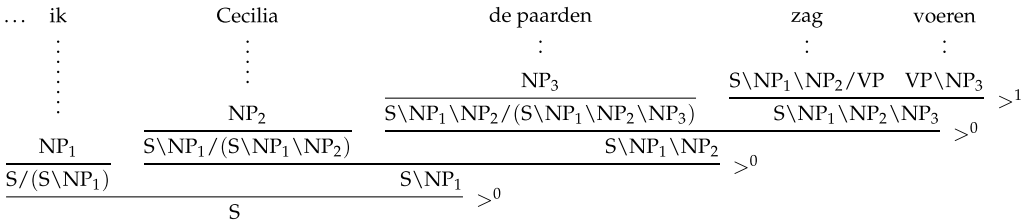


Figure 17
Derivation of Dutch cross–serial dependencies with type-raised noun complements.

now is a secondary rather than a primary input category. As a consequence, this grammar is capable of enforcing the obligatory cross-serial dependencies of Dutch. However, it is important to note that it requires type-raising over arbitary categories with target S (observe the increasingly complex type-raised categories for the NPs). This kind of type-raising is allowed in many variants of CCG, including the full formalism underlying OpenCCG. VW-CCG and O-CCG, however, are limited to generalized composition, and can only support derivations like the one in Figure 17 if all the type-raised categories for the noun phrases are available in the lexicon. The unbounded type-raising required by the Steedman–Baldridge analysis of Dutch would translate into an infinite lexicon, and so this analysis is not possible in VW-CCG and O-CCG.

We conclude by discussing the impact of several other constructs of OpenCCG that we have not captured in O-CCG. First, OpenCCG allows us to use generalized composition rules of arbitrary degree; there is no upper bound d on the composition degree as in an O-CCG grammar. It is known that this extends the generative capacity of CCG beyond that of TAG (Weir 1988). Second, OpenCCG allows categories to be annotated with feature structures. This has no impact on the generative capacity, as the features must take values from finite domains and can therefore be compiled into the atomic categories of the grammar. Finally, OpenCCG includes the combinatory rules of substitution and coordination, as well as *multiset slashes*, another extension frequently used in linguistic grammars. We have deliberately left these constructs out of O-CCG to establish the most direct comparison to the literature on VW-CCG. It is conceivable that their inclusion could restore the weak equivalence to TAG, but a proof of this result would require a non-trivial extension of the work of Vijay-Shanker and Weir (1994). Regarding multiset slashes, it is also worth noting that these were introduced with the expressed goal of allowing *more flexible* word

order, whereas restoration of weak equivalence would require *more controlled* word order.

5. Conclusion

In this article we have contributed two technical results to the literature on CCG. First, we have refined the weak equivalence result for CCG and TAG (Vijay-Shanker and Weir 1994) by showing that prefix-closed grammars are weakly equivalent to TAG only if target restrictions are allowed. Second, we have shown that O-CCG, the formal, composition-only core of OpenCCG, is not weakly equivalent to TAG. These results point to a tension in CCG between lexicalization and generative capacity: Lexicalized versions of the framework are less powerful than classical versions, which allow rule restrictions.

What conclusions one draws from these technical results depends on the perspective. One way to look at CCG is as a system for defining formal languages. Under this view, one is primarily interested in results on generative capacity and parsing complexity such as those obtained by Vijay-Shanker and Weir (1993, 1994). Here, our results clarify the precise mechanisms that make CCG weakly equivalent to TAG. Perhaps surprisingly, it is not the availability of generalized composition rules by itself that explains the generative power of CCG, but the ability to constrain the interaction between generalized composition and function application by means of target restrictions.

On the other hand, one may be interested in CCG primarily as a formalism for developing grammars for natural languages (Steedman 2000; Baldridge 2002; Steedman 2012). From this point of view, the suitability of CCG for the development of lexicalized grammars has been amply demonstrated. However, our technical results still serve as important reminders that extra care must be taken to avoid overgeneration when designing a grammar. In particular, it is worth double-checking that an OpenCCG grammar does not generate word orders that the grammar developer did not intend. Here the rewriting system that we presented in Figure 12 can serve as a useful tool: A grammar developer can take any derivation for a grammatical sentence, transform the derivation according to our rewriting rules, and check whether the transformed derivation still yields a grammatical sentence.

It remains an open question how the conflicting desires for generative capacity and lexicalization might be reconciled. A simple answer is to add some lexicalized method for enforcing target restrictions to CCG, specifically on the application rules. However, we are not aware that this idea has seen widespread use in the CCG literature, so it may not be called for empirically. Alternatively, one might modify the rules of O-CCG in such a way that they are no longer prefix-closed—for example, by introducing some new slash type. Finally, it is possible that the constructs of OpenCCG that we set aside in O-CCG (such as type-raising, substitution, and multiset slashes) might be sufficient to achieve the generative capacity of classical CCG and TAG. A detailed study of the expressive power of these constructs would make an interesting avenue for future research.

Acknowledgments

We are grateful to Mark Steedman and Jason Baldridge for enlightening discussions of the material presented in this article, and to the four anonymous reviewers of the article for their detailed and constructive comments.

References

Ajdukiewicz, Kazimierz. 1935. Die syntaktische Konnexität. *Studia Philosophica*, 1:1–27.

Baldridge, Jason. 2002. *Lexically Specified Derivational Control in Combinatory*

Categorial Grammar. Ph.D. thesis, University of Edinburgh, Edinburgh, UK.

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

Bar-Hillel, Yehoshua, Haim Gaifman, and Eli Shamir. 1960. On categorial and phrase structure grammars. *Bulletin of the Research Council of Israel*, 9F(1):1–16. Reprinted in Yehoshua Bar-Hillel. *Language and Information: Selected Essays on Their Theory and Application*, pages 99–115. Addison-Wesley, 1964.

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

Eisner, Jason. 1996. Efficient normal-form parsing for Combinatory Categorial Grammar. In *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 79–86, Santa Cruz, CA.

Gazdar, Gerald. 1987. Applicability of indexed grammars to natural language. In Uwe Reyle and Christian Rohrer, editors, *Natural Language Parsing and Linguistic Theories*. D. Reidel, pages 69–94.

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

Joshi, Aravind K. and Yves Schabes. 1997. Tree-Adjoining Grammars. In Grzegorz Rozenberg and Arto Salomaa, editors, *Handbook of Formal Languages*, volume 3. Springer, pages 69–123.

Kuhlmann, Marco, Alexander Koller, and Giorgio Satta. 2010. The importance of rule restrictions in CCG. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 534–543, Uppsala.

Moortgat, Michael. 2011. Categorial type logics. In Johan van Benthem and Alice ter Meulen, editors, *Handbook of Logic and Language*. Elsevier, second edition, chapter 2, pages 95–179.

Pollard, Carl J. 1984. *Generalized Phrase Structure Grammars, Head Grammars, and Natural Language*. Ph.D. thesis, Stanford University.

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

Steedman, Mark. 1996. *Surface Structure and Interpretation*, volume 30 of *Linguistic Inquiry Monographs*. MIT Press.

Steedman, Mark. 2000. *The Syntactic Process*. MIT Press.

Steedman, Mark. 2012. *Taking Scope*. MIT Press.

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

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

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

Vijay-Shanker, K., David J. Weir, and Aravind K. Joshi. 1986. Tree adjoining and head wrapping. In *Proceedings of the Eleventh International Conference on Computational Linguistics (COLING)*, pages 202–207, Bonn.

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

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

White, Michael. 2013. OpenCCG: The OpenNLP CCG Library. `http://openccg.sourceforge.net/` Accessed November 13, 2013.