A New Parsing Algorithm for Combinatory Categorial Grammar

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

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

Abstract

We present a polynomial-time parsing algorithm for CCG, based on a new decomposition of derivations into small, shareable parts. Our algorithm has the same asymptotic complexity, $\mathcal{O}(n^6)$, as a previous algorithm by Vijay-Shanker and Weir (1993), but is easier to understand, implement, and prove correct.

1 Introduction

Combinatory Categorial Grammar (CCG; Steedman and Baldridge (2011)) is a lexicalized grammar formalism that belongs to the class of so-called mildly context-sensitive formalisms, as characterized by Joshi (1985). CCG has been successfully used for a wide range of practical tasks including data-driven parsing (Clark and Curran, 2007), wide-coverage semantic construction (Bos et al., 2004; Kwiatkowski et al., 2010; Lewis and Steedman, 2013) and machine translation (Weese et al., 2012).

Several parsing algorithms for CCG have been presented in the literature. Earlier proposals show running time exponential in the length of the input string (Pareschi and Steedman, 1987; Tomita, 1988). A breakthrough came with the work of Vijay-Shanker and Weir (1990) and Vijay-Shanker and Weir (1993) who report the first polynomial-time algorithm for CCG parsing. Until this day, this algorithm, which we shall refer to as the V&W algorithm, remains the *only* published polynomial-time parsing algorithm for CCG. However, we are not aware of any practical parser for CCG that actually uses it. We speculate that this has two main reasons: First, some authors have argued that linguistic resources available for CCG can be covered with context-free fragments of the formalism (Fowler and Penn, 2010), for which more efficient parsing algorithms can be given. Second, the V&W algorithm is considerably more complex than parsing algorithms for equivalent mildly context-sensitive formalisms, such as Tree-Adjoining Grammar (Joshi and Schabes, 1997), and is quite hard to understand, implement, and prove correct.

The V&W algorithm is based on a special decomposition of CCG derivations into smaller parts that can then be shared among different derivations. This sharing is the key to the polynomial runtime. In this article we build on the same idea, but develop an alternative polynomial-time algorithm for CCG parsing. The new algorithm is based on a different decomposition of CCG derivations, and is arguably simpler than the V&W algorithm in at least two respects: First, the new algorithm uses only three basic steps, against the nine basic steps of the V&W parser. Second, the correctness proof of the new algorithm is simpler than the one reported by Vijay-Shanker and Weir (1993). The new algorithm runs in time $\mathcal{O}(n^6)$ where n is the length of the input string, the same as the V&W parser.

We organize our presentation as follows. In Section 2 we introduce CCG and the central notion of derivation trees. In Section 3 we start with a simple but exponential-time parser for CCG, from which we derive our polynomial-time parser in Section 4. Section 5 further simplifies the algorithm and proves its correctness. We then provide a discussion of our algorithm and possible extensions in Section 6. Section 7 concludes the article.

Transactions of the Association for Computational Linguistics, 2 (2014) 405–418. Action Editor: Mark Steedman.
Submitted 4/2014; Revised 8/2014; Published 10/2014.

2 Combinatory Categorial Grammar

We assume basic familiarity with CCG in general and the formalism of Weir and Joshi (1988) in particular. In this section we set up our terminology and notation. A CCG has two main parts: a *lexicon* that associates words with categories, and *rules* that specify how categories can be combined into other categories. Together, these components give rise to *derivations* such as the one shown in Figure 1.

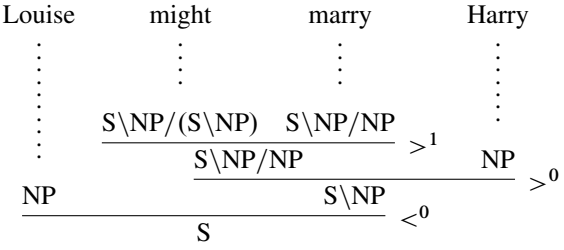


Figure 1: A sample derivation tree.

2.1 Lexicon

The CCG lexicon is a finite set of word–category pairs $w := X$.[1] Categories are built from a finite set of *atomic categories* and two binary operators: forward slash (/) and backward slash (\). Atomic categories represent the syntactic types of complete constituents; they include a distinguished category S for complete sentences. A constituent with the complex category X/Y represents a function that seeks a constituent of category Y immediately to its right and returns a constituent of category X; similarly, $X\backslash Y$ represents a function that seeks a Y to its left. We treat slashes as left-associative operators and omit unnecessary parentheses. By this convention, every category X can be written as

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

where $m \geq 0$, A is an atomic category called the *target* of X and the $|_i X_i$ are slash–category pairs called the *arguments* of X. We view these arguments as being arranged in a stack with $|_1 X_1$ at the top and $|_m X_m$ at the bottom. Thus another way of writing the category X above is as $X = A\alpha$, where α is a (possibly empty) stack of m arguments. The number m is called the *arity* of X; we denote it by $ar(X)$.

2.2 Rules

The rules of CCG are directed versions of (generalized) functional composition. There are two forms, *forward rules* and *backward rules*:

$$X/Y \quad Y|_d Y_d \cdots |_1 Y_1 \Rightarrow X|_d Y_d \cdots |_1 Y_1 \quad (>^d)$$
$$Y|_d Y_d \cdots |_1 Y_1 \quad X\backslash Y \Rightarrow X|_d Y_d \cdots |_1 Y_1 \quad (<^d)$$

[1] The formalism of Weir and Joshi (1988) also allows lexicon entries for the empty string, a feature that we ignore here.

Every rule is obtained by choosing a specific *degree* $d \geq 0$ and specific directions (forward or backward) for each of the slashes $|_i$, while X, Y and the Y_i are variables ranging over categories. Thus for every degree $d \geq 0$ there are 2^d forward rules and 2^d backward rules. The rules of degree 0 are called *application* rules. In contexts where we refer to both application and composition, we use the latter term for "proper" composition rules of degree $d > 0$. Note that in most of this article we ignore additional rules required for linguistic analysis with CCG, in particular type-raising and substitution. We briefly discuss these rules in Section 6.

Every CCG grammar restricts itself to a finite set of rules, but each such rule may give rise to infinitely many *rule instances*. A rule is instantiated by substituting concrete categories for the variables. For example, the derivation in Figure 1 contains the following instance of forward composition ($>^1$):

$$\text{S}\backslash\text{NP}/(\text{S}\backslash\text{NP})\ \text{S}\backslash\text{NP}/\text{NP} \Rightarrow \text{S}\backslash\text{NP}/\text{NP}$$

Note that we overload the double arrow to denote not only rules but also rule instances. Given a rule instance, the category that instantiates the pattern X/Y (forward) or $X\backslash Y$ (backward) is called the *primary input*, and the category that instantiates the pattern $Y|_d Y_d \cdots |_1 Y_1$ is called the *secondary input*. Adopting our stack-based view, each rule can be understood as an operation on argument stacks: pop $|Y$ off the stack of the primary input; pop the $|_i Y_i$ off the stack of the secondary input and push them to the stack of the primary input (preserving their order).

The formalism of Weir and Joshi (1988) allows to restrict the valid instances of individual rules. Similar to our treatment of additional combinatory rules, in most of this article we ignore these rule restrictions; but see the discussion in Section 6.

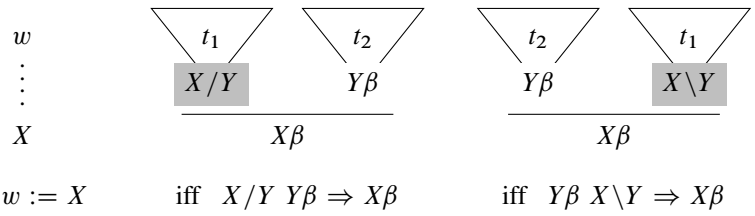


Figure 2: Recursive definition of derivation trees. Nodes labeled with primary input categories are shaded.

2.3 Derivation Trees

The set of *derivation trees* of a CCG can be formally defined as in Figure 2. There and in the remainder of this article we use β and other symbols from the beginning of the Greek alphabet to denote a (possibly empty) stack of arguments. Derivation trees consist of unary branchings and binary branchings: unary branchings (drawn with dotted lines) correspond to lexicon entries; binary branchings correspond to (valid instances of) composition rules. The *yield* of a derivation tree is the left-to-right sequence of its leaves. The *type* of a derivation tree is the category at its root.

3 CKY-Style Parsing Algorithm

As the point of departure for our own work, we now introduce a straightforward, CKY-style parsing algorithm for CCGs. It is a simple generalization of the algorithm presented by Shieber et al. (1995), which is restricted to grammars with rules of degree 0 or 1. As in that article, we specify our algorithm in terms of a *grammatical deduction system*.

3.1 Deduction System

We are given a CCG and a string $w = w_1 \cdots w_n$ to be parsed, where each w_i is a lexical token. As a general notation, for integers i, j with $0 \leq i \leq j \leq n$ we write $w[i, j]$ to denote the substring $w_{i+1} \cdots w_j$ of w. As usual, we take $w[i, i]$ to be the empty string.

Items The CKY-style algorithm uses a logic with items of the form $[X, i, j]$ where X is a category and i, j are fencepost positions in w. The intended interpretation of such an item is to assert that we can build a derivation tree with yield $w[i, j]$ and type X. The goal of the algorithm is the construction of the item $[S, 0, n]$, which asserts the existence of a derivation tree for the entire input string. (Recall that S is the distinguished category for sentences.)

Axioms and Inference Rules The steps of the algorithm are specified by means of inference rules over items. These rules implement the recursive definition of derivation trees given in Figure 2. The construction starts with axioms of the form $[X, i, i + 1]$ where $w_{i+1} := X$ is a lexicon entry; these items assert the existence of a unary-branching derivation tree of the form shown in the left of Figure 2 for each lexical token w_{i+1}. There is one inference rule for every forward rule (application or composition):

$$\frac{[X/Y, i, j] \quad [Y\beta, j, k]}{[X\beta, i, k]} \; X/Y \; Y\beta \Rightarrow X\beta \qquad (1)$$

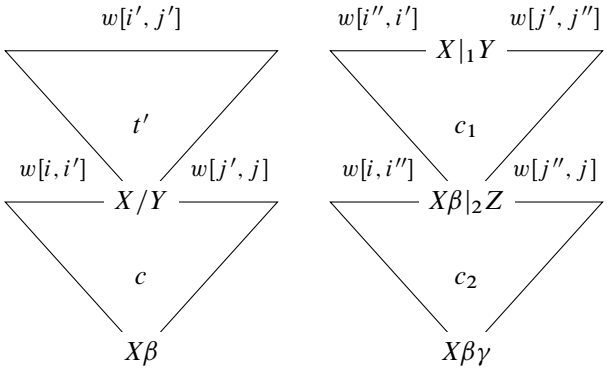
A symmetrical rule is used for backward application and composition. However, here and in the remainder of the article we only specify the forward version of each rule and leave the backward version implicit.

3.2 Correctness and Runtime

The soundness and completeness of the CKY-style algorithm can be proved by induction on the number of inferences and the number of nodes in a derivation tree, respectively.

It is not hard to see that, in the general case, the algorithm uses an amount of time and space exponential with respect to the length of the input string, n. This is because rule (1) may be used to grow the arity of primary input categories up to some linear function of n, resulting in exponentially many categories.[2] Note that this is only possible if there are rules with degree 2 or more. For grammars restricted to rules with degree 0 or 1, such as those considered by Shieber et al. (1995), the runtime of the algorithm is cubic in n. This restricted class of grammars only holds context-free generative power, while the power of general CCG is beyond that of context-free grammars (Vijay-Shanker and Weir, 1994).

[2] Categories whose arity is not bounded by a linear function of n are not useful, in the sense that they cannot occur in complete derivations.



(a) tree t (b) context c

Figure 3: Decomposition of derivations.

4 A Polynomial-Time Algorithm

We now introduce our polynomial-time algorithm. This algorithm uses the same axioms and the same goal item as the CKY-style algorithm, but features new items and new inference rules.

4.1 New Items

In a first step, in order to avoid runtime exponential in n we restrict our item set to categories whose arity is bounded by some grammar constant c_G:

$$[X, i, j] \quad \text{where} \quad ar(X) \leq c_G$$

The exact choice of the constant will be discussed in Section 5. With the restricted item set, the new algorithm behaves like the old one as long as the arity of categories does not exceed c_G. However, rule (1) alone is no longer complete: Derivations with categories whose arity exceeds c_G cannot be simulated anymore. To remedy this deficiency, we introduce a new type of item to implement a specific decomposition of long derivations into smaller pieces.

Consider a derivation t of the form shown in Figure 3(a). Note that the yield of t is $w[i, j]$. The derivation consists of two parts, named t' and c; these share a common node with a category of the form X/Y. Now assume that c has the special property that none of the combinatory rules that it uses pops the argument stack of the category X. This means that c, after popping the argument $/Y$, may push new arguments and pop them again, but may never "touch" X. We call a fragment with this special property a *derivation context*. (A formal definition will be given in Section 5.2.)

The special property of c is useful because it implies that c can be carried out for any choice of X. To be more specific, let us write β for the (possibly empty) sequence of arguments that c pushes to the argument stack of X in place of $/Y$. We shall refer to $/Y$ as the *bridging argument* and to the sequence β as the *excess* of c. Suppose now that we replace t' by a derivation tree with the same yield but with a type X'/Y where $X' \neq X$. Then because c does not touch X' we obtain another valid derivation tree with the same yield as t; the type of this tree will be $X'\beta$.

For the combination with c, the internal structure of t' is of no importance; the only important information is the extent of the yield of t' and the identity of the bridging argument $/Y$. In terms of our deduction system, this can be expressed as follows: The derivation context c can be combined with any tree t' that is associated with an item of the form $[X/Y, i', j']$, where X is any category. Similarly, the internal structure of c is of no importance either, as long as the argument stack of the category X remains untouched. It suffices to record the following:

- the extent of the yield of t, specified in terms of the positions i and j;
- the extent of the yield of t', specified in terms of the positions i' and j';
- the bridging argument $/Y$; and
- the excess β.

We represent these pieces of information in a new type of item of the form $[/Y, \beta, i, i', j', j]$. The intended interpretation of these items is to assert that, for any choice of X, if we can build a derivation tree t' with yield $w[i', j']$ and type X/Y, then we can also build a derivation tree t' with yield $w[i, j]$ and type $X\beta$. We also use items $[\backslash Y, \beta, i, i', j', j]$ with a backward slash, with a similar meaning. Like items that represent derivation trees, our items for derivation contexts are arity-restricted:

$$[|Y, \beta, i, i', j', j] \quad \text{where} \quad ar(Y\beta) \leq c_G$$

As we will see in Section 5, these restricted items suffice to simulate all derivations of a CCG. Furthermore, this can be done in time polynomial in n, because our encoding allows sharing of the same items among several derivations.

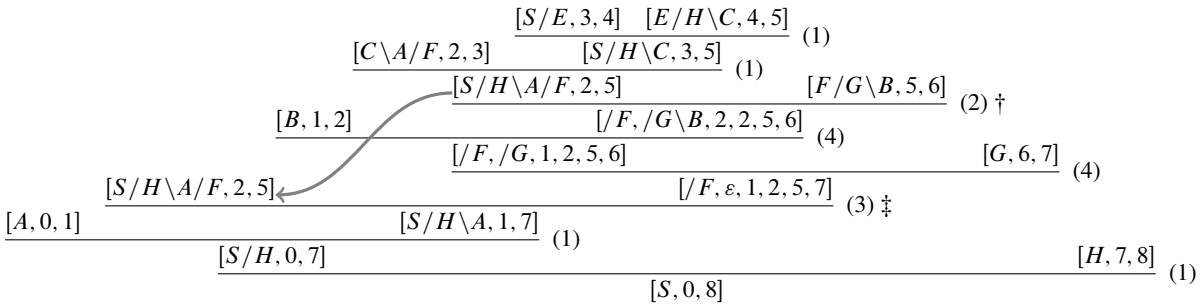


Figure 4: A sample derivation of the grammatical deduction system of Section 4. Inference † triggers a new context item from a tree item; inference ‡ reuses the tree item (as indicated by the arrow), recombining it with the (modified) context item.

4.2 New Inference Rules

In our parsing algorithm, context items are introduced whenever the composition of two categories whose arities are bounded by c_G would result in a category whose arity exceeds this bound:

$$\frac{[X/Y, i, j] \quad [Y\beta, j, k]}{[/Y, \beta, i, i, j, k]} \begin{cases} X/Y\ Y\beta \Rightarrow X\beta \\ ar(X\beta) > c_G \end{cases} \tag{2}$$

The new rule has the same antecedents as rule (1), but rather than extending the derivation asserted by the first antecedent $[X/Y, i, j]$, which is not possible because of the arity bound, it triggers a new derivation context, asserted by the item $[/Y, \beta, i, i, j, k]$. Further applications and compositions will extend the new context, and only when the excess of this context has become sufficiently small will it be recombined with the derivation that originally triggered it. This is done by the following rule:

$$\frac{[X|Y, i', j'] \quad [|Y, \beta, i, i', j', j]}{[X\beta, i, j]} \tag{3}$$

Note that this rule (like all rules in the deduction system) is only defined on valid items; in particular it only fires if the arity of the category $X\beta$ is bounded by c_G.

The remaining rules of the algorithm parallel the three rules that we have introduced so far but take items that represent derivation contexts rather than derivation trees as their first antecedents. First out, rule (4) extends a derivation context in the same way as rule (1) extends a derivation tree.

$$\frac{[|Y, \beta/Z, i, i', j', j] \quad [Z\gamma, j, k]}{[|Y, \beta\gamma, i, i', j', k]} \begin{cases} X/Z\ Z\gamma \Rightarrow X\gamma \\ ar(Y\beta\gamma) \leq c_G \end{cases} \tag{4}$$

Rule (5) is the obvious correspondent of rule (2): It triggers a new context when the antecedent context cannot be extended because of the arity bound.

$$\frac{[|Y, \beta/Z, i, i', j', j] \quad [Z\gamma, j, k]}{[/Z, \gamma, i, i, j, k]} \begin{cases} X/Z\ Z\gamma \Rightarrow X\gamma \\ ar(Y\beta\gamma) > c_G \end{cases} \tag{5}$$

Finally, and parallel to rule (3), we need a rule to recombine a context with the context that originally triggered it. As it will turn out, we only need this in cases where the triggered context has no excess.

$$\frac{[|_1Y, \beta|_2Z, i'', i', j', j''] \quad [|_2Z, \varepsilon, i, i'', j'', j]}{[|_1Y, \beta, i, i', j', j]} \tag{6}$$

4.3 Sample Derivation

We now illustrate our algorithm on a toy grammar. The grammar has the following lexicon:

$$\begin{aligned} w_1 &:= A & w_5 &:= E/H\backslash C \\ w_2 &:= B & w_6 &:= F/G\backslash B \\ w_3 &:= C\backslash A/F & w_7 &:= G \\ w_4 &:= S/E & w_8 &:= H \end{aligned}$$

The start symbol is S. The grammar allows all instances of application and all instances of composition with degree bounded by 2. We let $c_G = 3$ (as explained later in Section 5.2).

A derivation of our deduction system on the input string $w_1 \cdots w_8$ is given in Figure 4. We start by applying rule (1) twice (once forward, once backward) to obtain the item $[S/H\backslash A/F, 2, 5]$. Combining this item with the axiom $[F/G\backslash B, 5, 6]$ is not possible using rule (1), as this would result in

a category with arity 4, exceeding the arity bound. We therefore use rule (2) to trigger the context item $[/F, /G\backslash B, 2, 2, 5, 6]$ (†). Successively, we use rule (4) twice to obtain the item $[/F, \varepsilon, 1, 2, 5, 7]$. At this point we use rule (3) (with $\beta = \varepsilon$) to recombine the context item with the tree item that originally triggered it (‡); this yields the item $[S/H\backslash A, 1, 7]$. Note that the recombination effectively retrieves the portion of the stack that was below the argument $/F$ when the context item was triggered in †. Double application of rule (1) produces the goal item $[S, 0, 8]$.

4.4 Runtime Analysis

We now turn to an analysis of the runtime complexity of our algorithm. We first consider runtime complexity with respect to the length of the input string, n. The runtime is dominated by the number of instantiations of rule (6) which involves two context items as antecedents. By inspection of this rule, we see that the number of possible instantiations is bounded by n^6. Therefore we conclude that the algorithm runs in time $\mathcal{O}(n^6)$.

We now consider runtime complexity with respect to the size of the input grammar. Here the runtime is dominated by the number of instantiations of rules (1)–(5). For example, rule (5) combines items

$$[|Y, \beta/Z, i, i', j', j] \quad \text{and} \quad [Z\gamma, j, k]\,.$$

By our restrictions on items, both the arity of $Y\beta/Z$ and the arity of $Z\gamma$ are upper-bounded by the constant c_G. Now recall that every category X can be written as $X = A\alpha$ for some atomic category A and stack of arguments α. Let $\mathcal{A}$ be the set of atomic categories in the input grammar, and let $\mathcal{L}$ be the set of all arguments occurring in any category of the lexicon. By a result of Vijay-Shanker and Weir (1994, Lemma 3.1), every argument that may occur in a derived category occurs in $\mathcal{L}$. Then the number of possible instantiations of rule (5) as well as rules (1)–(4), and hence the runtime of the algorithm, is in

$$\mathcal{O}(|\mathcal{A}|\,|\mathcal{L}|^{c_G} \cdot |\mathcal{L}|^{c_G}) = \mathcal{O}(|\mathcal{A}|\,|\mathcal{L}|^{2c_G})\,.$$

Note that both $\mathcal{A}$ and $\mathcal{L}$ may grow with the grammar size. As we will see in Section 5.2, the constant c_G also depends on the grammar size. This means that the worst-case runtime complexity of our parser is exponential in the size of the input grammar. We will return to this point in Section 7.

5 Correctness

In this section we prove the correctness of our parsing algorithm. In order to simplify the proofs, we start by simplifying our algorithm, at the cost of making it less efficient:

- We remove the rules for extending trees and contexts, rule (1) and rule (4).
- We conflate the rules for triggering contexts, rule (2) and rule (5), into the single rule

$$\frac{[Y\beta, j, k]}{[/Y, \beta, i, i, j, k]} \qquad X/Y\ Y\beta \Rightarrow X\beta \qquad (7)$$

  This rule blindly guesses the extension of the triggering tree or context (specified by positions i and j), rather than waiting for a corresponding item to be derived.

The simplified algorithm is specified in Figure 5. We now argue that this algorithm and the algorithm from Section 4 parse exactly the same derivation trees, although they use different parsing strategies. First, we observe that rule (1) in the old algorithm can be simulated by a combination of other rules in the simplified algorithm, as follows:

$$\dfrac{[X/Y, i, j] \qquad \dfrac{[Y\beta, j, k]}{[/Y, \beta, i, i, j, k]}\ (7)}{[X\beta, i, k]}\ (3)$$

Furthermore, the simplified algorithm does no longer need rule (4), whose role is now taken over by rule (7). To see this, recall that rule (4) extends an existing context whenever the composition of two categories results in a new category whose arity does not exceed c_G. In contrast, rule (7) *always* triggers a new context c, even if the result of the composition of c with some existing context satisfies the above arity restriction. Despite the difference in the adopted strategy, these two rules are equivalent in terms of stack content, leading to the same derivation trees.

5.1 Definitions

We introduce some additional terminology and notation that we will use in the proofs. For a derivation tree t and a node u of t, we write $t[u]$ to denote the category at u, and we write $t|_u$ to denote the subtree of t at u. Formally, $t|_u$ is the restriction of t to node u and all of its descendants. Each subtree of a derivation tree is another derivation tree.

Items of type 1: $[X, i, j],\ ar(X) \leq c_G$ **Items of type 2:** $[|Y, \beta, i, i', j', j],\ ar(Y\beta) \leq c_G$

Axioms: $[X, i, i+1],\ w_{i+1} := X$ **Goal:** $[S, 0, n]$

Inference rules: $\dfrac{[Y\beta, j, k]}{[/Y, \beta, i, i, j, k]}\ X/Y\ Y\beta \Rightarrow X\beta$ (7)

$$\frac{[X/Y, i', j'] \quad [/Y, \beta, i, i', j', j]}{[X\beta, i, j]}\ (3) \qquad \frac{[|_1 Y, \beta|_2 Z, i'', i', j', j''] \quad [|_2 Z, \varepsilon, i, i'', j'', j]}{[|_1 Y, \beta, i, i', j', j]}\ (6)$$

Figure 5: Items and inference rules of the simplified algorithm for an input string $w_1 \cdots w_n$.

Definition 1 Let t be a derivation tree with root r. Then t has *signature* $[X, i, j]$ if

1. the yield of t is $w[i, j]$ and
2. the type of t is X, that is, $t[r] = X$.

Note that while we use the same notation for signatures as for items, the signature of a derivation tree is a purely structural concept, whereas an item is an object in the algorithm.

A central concept in our proof is the notion of *spine*. Recall that a derivation tree consists of unary branchings and binary branchings. In each binary branching, we refer to the two children of the branching's root node as the *primary child* and the *secondary child*, depending on which of the two is labeled with the primary and secondary input category of the corresponding rule instance. In Figure 2, the primary children of the root node are shaded.

Definition 2 For a derivation tree t, the *spine* of t is the unique path that starts at the root node of t and at each node u continues to the primary child of u.

The spine of a derivation tree always ends at a node that is labeled with a category from the lexicon.

Definition 3 Let t be a derivation tree with root r. A *derivation context* c is obtained by removing all proper descendants of some node $f \neq r$ on the spine of t, under the restriction that $ar(t[u]) > ar(t[r])$ for every node u on the spine properly between f and r.

The node f is called the *foot node* of c. The *yield* of c is the pair whose first component is the yield of t to the left of f and whose second component is the yield of t to the right of f. For a derivation context c and a node u of c, we write $c[u]$ to denote the category at u.

Definition 3 formalizes the concept of derivation contexts that we introduced in Section 4.1. First, because f is on the spine and $f \neq r$, the category $c[f]$ takes the form X/Y. The arity restriction implies that the category of every node u on the spine properly between f and r takes the form $X\beta_u$, $|\beta_u| > 0$, and that the category at the root takes the form $X\beta$, $|\beta| \geq 0$. Thus the category X is never exposed in c, except perhaps at r. As we will see in Section 5.4, this property, together with a careful selection of "split nodes", will allow us to decompose derivations into smaller, shareable parts. The basic idea is the same as in the tabulation of pushdown automata (Lang, 1974; Nederhof and Satta, 2004), where the pushdown in our case is the argument stack of the primary input categories along a spine.

The concepts of signature and spine are generalized to derivation contexts as follows:

Definition 4 Let c be a derivation context with root node r and foot node f. Then c has *signature* $[|Y, \beta, i, i', j', j]$ if

1. the yield of c is $(w[i, i'], w[j', j])$;
2. for some X, $c[f] = X|Y$ and $c[r] = X\beta$.

Definition 5 For a derivation context c, the *spine* of c is the path from its root node to its foot node.

5.2 Grammar Constant

Before we start with the proof as such, we turn to the choice of the grammar constant c_G, which was left pending in previous sections. Recall that we are using c_G as a bound on the arity of X in type 1 items $[X, i, j]$. Since these items are produced by our axioms from the set of categories in the lexicon, c_G must not be smaller than the maximum arity ℓ of a category in this finite set.

We also use c_G as a bound on the arity of the category $Y\beta$ in type 2 items $[|Y, \beta, i, i', j', j]$. These items are produced by inference rule (7) to simulate instances of composition of the form $X/Y \ Y\beta \Rightarrow X\beta$. Here the length of β is bounded by the maximum degree d of a composition rule in the grammar, and $ar(Y)$ is bounded by the maximum arity a of an argument from the (finite) set $\mathcal{L}$ of arguments in the lexicon (recall Section 4.4). Therefore c_G cannot be smaller than $a + d$. Putting everything together, we obtain the condition

$$c_G \geq \max\{\ell, a + d\}. \tag{8}$$

The next lemma will be used in several places later.

Lemma 1 *Let c be a derivation context with signature $[|Y, \beta, i, i', j', j]$. Then $ar(Y\beta) \leq c_G$.*

Proof. Let r and f be the root and the foot node, respectively, of c. From the definition of signature, there must be some X such that $c[r] = X\beta$ and $c[f] = X|Y$. Now let p be the parent node of f, and assume that the rule used at p is instantiated as $X/Y \ Y\beta' \Rightarrow X\beta'$, so that $c[p] = X\beta'$. If $p = r$ then $\beta' = \beta$; otherwise, because of the arity restriction in the definition of derivation contexts (Definition 3) we have $|\beta'| > |\beta|$. Then

$$ar(Y\beta) \leq ar(Y\beta') \leq c_G,$$

where the right inequality follows from the assumption that $X/Y \ Y\beta' \Rightarrow X\beta'$ is a rule instance of the grammar, and from inequality (8).

5.3 Soundness

We start the correctness proof by arguing for the soundness of the deduction system in Figure 5. More specifically, we show that for every item of type 1 there exists a derivation tree with the same signature, and that for every item of type 2 there exists a derivation context with the same signature.

The soundness of the axioms is obvious.

Rule (7) states that, if we have built a derivation tree t with signature $[Y\beta, j, k]$ then we can build a derivation context c with signature $[/Y, \beta, i, i, j, k]$. Under the condition that the grammar admits the rule instance $X/Y \ Y\beta \Rightarrow X\beta$, this inference is sound; the context can be built as shown in Figure 6.

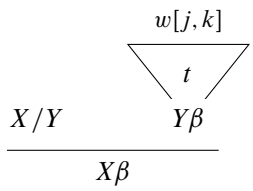


Figure 6: Soundness of rule (7).

Rule (3) states that, if we have built a derivation tree t' with signature $[X/Y, i', j']$ and a context c with signature $[/Y, \beta, i, i', j', j]$, then we can build a new tree t with signature $[X\beta, i, j]$. We obtain t by substituting t' for the foot node of c (see Figure 3(a)).

Rule (6) states that, if we have built a derivation context c_1 with signature $[|_1 Y, \beta|_2 Z, i'', i', j', j'']$ and another context c_2 with signature $[|_2 Z, \varepsilon, i, i'', j'', j]$, then we can build a derivation context c with signature $[|_1 Y, \beta, i, i', j', j]$. We obtain c by substituting c_1 for the foot node of c_2; this is illustrated by Figure 3(b) if we assume $\gamma = \varepsilon$.

5.4 Completeness

In the final part of our correctness proof we now prove the completeness of the deduction system in Figure 5. Specifically we show the following stronger statement: For every derivation tree and for every derivation context with a signature I satisfying the arity bounds for items of Figure 5, the deduction system infers the corresponding item I. From this statement we can immediately conclude that the system constructs the goal item whenever there exists a derivation tree whose yield is the complete input string and whose type is the distinguished category S.

Our proof is by induction on a measure that we call *rank*. The rank of a derivation tree or context is the number of its non-leaf nodes. Note that this definition implies that the foot node of a context is not counted against its rank. The rank of a tree or context is always at least 1, with rank 1 only realized for derivation trees consisting of a single node.

5.4.1 Base Case

Consider a derivation tree t with signature $[X, i, j]$ and $rank(t) = 1$. The tree t takes the form shown in the left of Figure 2, and we have $j = i + 1$ and $w_j := X$. The item $[X, i, j]$ is then produced by one of the axioms of our deduction system.

5.4.2 Inductive Case

The general idea underlying the inductive case can be stated as follows. We consider a derivation tree or context φ with signature I satisfying the bounds stated in Figure 5 for items of type 1 or 2. We then identify a special node s in φ's spine, which we call the *split* node. We use s to "split" φ into two parts that are either derivation trees or contexts, that both satisfy the bounds for items of type 1 or 2, and that both have rank smaller than the rank of φ. We then apply the induction hypothesis to obtain two items that can be combined by one of the inference rules of our algorithm, resulting in the desired item I for φ. We first consider the case in which φ is a tree, and later the case in which φ is a context.

5.4.3 Splitting Trees

Consider a derivation tree t with signature $[X', i, j]$, root node r, and $rank(t) > 1$. Then the spine of t consists of at least 2 nodes. Now assume that $ar(X') \leq c_G$, that is, $[X', i, j]$ is a valid item.

Choose the split node s to be the highest (closest to the root) non-root node on the spine for which $ar(t[s]) \leq c_G$. Node s always exists, as the arity constraint is satisfied at least for the lowest (farthest from the root) node on the spine, which is labeled with a category from the lexicon.

Consider the subtree $t' = t|_s$; thus s is the root node of t'. Because s is a primary node in t, the category at s has at least one argument. We deal here with the case where this category takes the form $t[s] = X/Y$; the case $t[s] = X\backslash Y$ is symmetrical. Thus the signature of t' takes the form $[X/Y, i', j']$ where i', j' are integers with $i \leq i' < j' \leq j$. By our choice of s, $ar(X/Y) \leq c_G$, and therefore $[X/Y, i', j']$ is a valid item. Furthermore, $rank(t|_s) < rank(t)$, as r does not belong to $t|_s$. We may then use the induction hypothesis to deduce that our algorithm constructs the item $[X/Y, i', j']$.

Now consider the context c that is obtained from t by removing all proper descendants of s; thus r is the root node of c and s is its foot node. To see that c is well-defined, note that our choice of s guarantees that $ar(t[u]) > ar(t[r])$ for every node u that is properly between s and r: If there was a node u such that $ar(t[u]) \leq ar(t[r])$ then because of $t[r] = X'$ and our assumption that $ar(X') \leq c_G$ we would have chosen u instead of s.

Now let β be the excess of c; then $X' = t[r] = t[s]\beta = X\beta$. Thus the signature of c takes the form $[/Y, \beta, i, i', j', j]$. Applying Lemma 1 to c, we get $ar(Y\beta) \leq c_G$, and therefore $[/Y, \beta, i, i', j', j]$ is also a valid item. Furthermore, $rank(c) < rank(t)$, since node s is counted in $rank(t)$ but not in $rank(c)$. By the induction hypothesis, we conclude that our algorithm constructs the item $[/Y, \alpha, i, i', j', j]$.

Finally, we apply the inference rule (3) to the previously constructed items $[X/Y, i', j']$ and $[/Y, \beta, i, i', j', j]$. This yields the item $[X\beta, i, j] = [X', i, j]$ for t, as desired.

5.4.4 Splitting Contexts

Consider a derivation context c with signature $[/Y, \beta, i, i', j', j]$, root r, and foot f. (The case where we have $\backslash Y$ instead of $/Y$ can be covered with a symmetrical argument.) From Definition 4 we know that there is a category X such that $c[f] = X/Y$ and $c[r] = X\beta$, and from the definition of context we know that for every spinal node u that is properly between f and r it holds that $ar(c[u]) > ar(c[r])$. Now assume that $ar(Y\beta) \leq c_G$, that is, $[/Y, \beta, i, i', j', j]$ is a valid item. We distinguish two cases below.

Case 1 Suppose that the spine of c consists of exactly 2 nodes. In this case the foot f is the left child of the root r and $i = i'$. Let f' be the right sibling of f and consider the subtree $t' = t|_{f'}$; thus f' is the root node of t'. The signature of t' takes the form $[Y\beta, j', j]$. By our assumption, $ar(Y\beta) \leq c_G$, and then $[Y\beta, j', j]$ is a valid item. Furthermore, $rank(t') < rank(c)$ since the root node r is counted in $rank(c)$ but not in $rank(t')$. Then, by the induction hypothesis, the item $[Y\beta, j', j]$ is constructed by our algorithm. We now apply inference rule (7) to this item; this yields the item $[/Y, \beta, i, i', j', j]$ for c, as required.

Case 2 Suppose that the spine of c consists of more than 2 nodes. This means that there is at least one spinal node that is properly between f and r.

Choose the split node s to be the deepest (farthest from the root) node properly between f and r for which $ar(c[s]) = ar(c[r]) + 1$. Node s always exists, as the arity constraint is satisfied at least for the primary child of r. This is because the definition of context states that $ar(c[u]) > ar(c[r])$ for every

node u in the spine, and at the same time, no combinatory rule can reduce the arity of its primary input category by more than one unit.

Consider the context c_1 that is obtained by restricting c to node s and all of its descendants; thus s is the root node of c_1 and f is the foot node. To see that c_1 is well-defined, note that our choice of s guarantees that $ar(c[u]) > ar(c[s])$ for every node u that is properly between f and s. To see this, suppose that there was a node $u \neq s$ such that $ar(c[u]) \leq ar(c[s])$. Since $ar(c[s]) = ar(c[r]) + 1$ and $ar(c[u]) \neq ar(c[s])$, by our definition of s, we would have $ar(c[u]) \leq ar(c[r])$, which cannot be because in c, every node u properly between f and s has arity $ar(c[u]) > ar(c[r])$.

Because f is a primary node in c, the category at f has at least one argument; call it $|_1 Y$. The node s is a primary node in c as well, so the excess of c_1 takes the form $\beta|_2 Z$, where $|_2 Z$ is the topmost argument of the category at s. Thus the signature of c_1 takes the form $[|_1 Y, \beta|_2 Z, i'', i', j', j'']$ where i'', j'' are integers with $i \leq i'' \leq i'$ and $j' \leq j'' \leq j$. Applying Lemma 1 to c_1, we get $ar(Y\beta|_2 Z) \leq c_G$, and therefore $[|_1 Y, \beta|_2 Z, i'', i', j', j'']$ is a valid item. Finally, we note that $rank(c_1) < rank(c)$, since the root node r is counted in c but not in c_1. By the induction hypothesis we conclude that our algorithm constructs the item $[|_1 Y, \beta|_2 Z, i'', i', j', j'']$.

Now consider the context c_2 that is obtained from c by removing all proper descendants of the node s; thus r is the root node of c_2 and s is the foot node. To see that c_2 is well-defined, note that $ar(c[u]) > ar(c[r])$ for every node u that is properly between s and r simply because every such node is also properly between f and r. The excess of c_2 is the empty stack ε by our choice of s. Thus the signature of c_2 is $[|_2 Z, \varepsilon, i, i'', j'', j]$. We apply Lemma 1 once more, this time to c_2, to show that $ar(Z) \leq c_G$, and conclude that $[|_2 Z, \varepsilon, i, i'', j'', j]$ is also a valid item. Finally we note that $rank(c_2) < rank(c)$, as the node s is counted in c but not in c_2. By the induction hypothesis we conclude that our algorithm constructs the item $[|_2 Z, \varepsilon, i, i'', j'', j]$.

We now apply inference rule (6) to the previously constructed items $[|_1 Y, \beta|_2 Z, i'', i', j', j'']$ and $[|_2 Z, \varepsilon, i, i'', j'', j]$. This yields the item $[|_1 Y, \beta, i, i', j', j]$ as desired.

6 Discussion

We round off the article with a discussion of our algorithm and possible extensions.

6.1 Support for Rule Restrictions

As we mentioned in Section 2, the CCG formalism of Weir and Joshi (1988) allows a grammar to impose certain restrictions on valid rule instances. More specifically, for every rule a grammar may restrict
(a) the target of the primary input category and/or
(b) parts of or the entire secondary input category.[3]

The algorithm in Figure 5 can be extended to support such rule restrictions. Note that already in its present form, the algorithm only allows inferences that are licensed by valid instances of a given rule. Supporting restrictions on the secondary input category (restrictions of type b) is straightforward—assuming that these restrictions can be efficiently tested. To also support restrictions on the target of the primary input category (restrictions of type a) the items can be extended by an additional component that keeps track of that target category for the corresponding derivation subtree or context. With this information, rule (7) can perform a check against the restrictions specified for the composition rule, and rules (3) and (6) merely need to test whether the target categories of their two antecedents match, and propagate the common target category to the conclusion. This is essentially the same solution as the one adopted in the V&W algorithm.

6.2 Support for Multi-Modal CCG

The modern version of CCG has abandoned rule restrictions in favor of a new, lexicalized control mechanism in the form of *modalities* or *slash types* (Steedman and Baldridge, 2011). However, as shown by Baldridge and Kruijff (2003), every multi-modal CCG can be translated into an equivalent CCG with rule restrictions. The basic idea is to specialize the target of each category and argument for a slash type, and to reformulate the multi-modal rules as rules with restrictions that reference this information. With this simulation, our parsing algorithm can also be used as a parsing algorithm for multi-modal CCG.

[3] Such restrictions can be used, for example, to impose the linguistically relevant distinction between *harmonic* and *crossed* forms of composition.

6.3 Comparison with the V&W Algorithm

As already mentioned in Section 1, apart from the algorithm presented in this article, the only other algorithm known to run in polynomial time in the length of the input string is the one presented by Vijay-Shanker and Weir (1993). At an abstract level, the two algorithms are based on the same basic idea of decomposing CCG derivations into pieces of two different types, one of which spans a portion of the input string that includes a gap. This idea actually underlies several parsing algorithms for equivalent mildly context-sensitive formalisms, such as Tree-Adjoining Grammar (Joshi and Schabes, 1997).

The main difference between the V&W algorithm and the one presented in this article is the use of different decompositions for CCG derivations. In our algorithm we allow the excess of a derivation context to be the empty list of arguments, something that is not possible in the V&W algorithm. There, when an application operation empties the excess β of some context, one is forced to retrieve, in the same elementary step, the portion of the stack placed right below β. This requires the distinction of several possible cases, resulting in four different realizations of the application rule (Vijay-Shanker and Weir, 1993, p. 616). As a consequence, the V&W algorithm uses nine (forward) inference rules, against our algorithm in Figure 5 which uses only three. Furthermore, some of the inference rules in the V&W algorithm use three antecedent items, while our use a maximum of two. This results in a runtime complexity of $\mathcal{O}(n^7)$ for the V&W algorithm, n the length of the input string; however, Vijay-Shanker and Weir (1993) show how their algorithm can be implemented in time $\mathcal{O}(n^6)$ at the cost of some extra bookkeeping. In contrast, our algorithm directly runs in time $\mathcal{O}(n^6)$.

The relative proliferation of inference rules, combined with the increase in their complexity, makes, in our own opinion, the specification of the V&W parser more difficult to understand and implement, and calls for a more articulated correctness proof.

6.4 Support for Additional Types of Rules

Like the V&W algorithm, our algorithm currently only supports (generalized) composition but no other combinatory rules required for linguistic analysis, in particular type-raising and substitution.

Type-raising is a unary rule of the (forward) form $X \Rightarrow T/(T\backslash X)$ where T is a variable over categories. Under the standard assumption that $T\backslash X$ is limited to a finite set of categories (Steedman, 2000), this rule can be implemented in our algorithm by introducing a new unary inference rule and choosing the constant c_G large enough to accomodate all instances of $T\backslash X$.

Substitution is a binary rule of the (forward) form $X/Y|Z\ Y|Z \Rightarrow X/Z$. This rule is easy to implement if both $/Y$ and $|Z$ are stored in the same item. Otherwise, we need to pass $|Z$ to any item storing the $/Y$. This can be done by changing the second antecedent of rule (6) to allow a single argument $|Z$ instead of the empty excess ε. The price of this change is spurious ambiguity in the derivations of the grammatical deduction system.

7 Conclusion

Recently, there has been a surge of interest in the mathematical properties of CCG; see for instance Hockenmaier and Young (2008), Koller and Kuhlmann (2009), Fowler and Penn (2010) and Kuhlmann et al. (2010). Following this line, this article has revisited the parsing problem for CCG.

Our work, like the polynomial-time parsing algorithm previously discovered by Vijay-Shanker and Weir (1993), is based on the idea of decomposing large CCG derivations into smaller, shareable pieces. Here we have proposed a derivation decomposition different from the one adopted by Vijay-Shanker and Weir (1993). This results in an algorithm which, in our own opinion, is simpler and easier to understand.

Although we have specified only a recognition version of the algorithm, standard techniques can be applied to obtain a derivation forest from our parsing table. This consists in saving backpointers at each inference rule, linking newly inferred items to their antecedent items.

As observed in Section 4.4, the worst case runtime of our algorithm is exponential in the size of the grammar. The same holds true for the algorithm of Vijay-Shanker and Weir (1993). We are not aware of any published discussion on this issue, and we therefore leave as an open problem the question whether CCG parsing can be done in polynomial time when the grammar is considered as part of the input.

Acknowledgments

We thank the Action Editor and the reviewers for their detailed and insightful feedback on the first version of this article. GS has been partially supported by MIUR under project PRIN No. 2010LYA9RH_006.

References

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

Johan Bos, Stephen Clark, Mark Steedman, James R. Curran, and Julia Hockenmaier. 2004. Wide-coverage semantic representations from a CCG parser. In *Proceedings of the 20th International Conference on Computational Linguistics (COLING)*, pages 1240–1246, Geneva, Switzerland.

Stephen Clark and James R. Curran. 2007. Wide-coverage efficient statistical parsing with CCG and log-linear models. *Computational Linguistics*, 33(4):493–552.

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

Julia Hockenmaier and Peter Young. 2008. Non-local scrambling: The equivalence of TAG and CCG revisited. In *Ninth International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+)*, Tübingen, Germany.

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

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

Alexander Koller and Marco Kuhlmann. 2009. Dependency trees and the strong generative capacity of CCG. In *Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, pages 460–468, Athens, Greece.

Marco Kuhlmann, 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, Sweden.

Tom Kwiatkowski, Luke Zettlemoyer, Sharon Goldwater, and Mark Steedman. 2010. Inducing probabilistic CCG grammars from logical form with higher-order unification. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1223–1233, Cambridge, MA, USA.

Bernard Lang. 1974. Deterministic techniques for efficient non-deterministic parsers. In Jacques Loecx, editor, *Automata, Languages and Programming, 2nd Colloquium, University of Saarbrücken, July 29–August 2, 1974*, number 14 in Lecture Notes in Computer Science, pages 255–269. Springer.

Mike Lewis and Mark Steedman. 2013. Combined distributional and logical semantics. *Transactions of the Association for Computational Linguistics*, 1(May):179–192.

Mark-Jan Nederhof and Giorgio Satta. 2004. Tabular parsing. In Carlos Martín-Vide, Victor Mitrana, and Gheorghe Păun, editors, *Formal Languages and Applications*, volume 148 of *Studies in Fuzziness and Soft Computing*, pages 529–549. Springer.

Remo Pareschi and Mark Steedman. 1987. A lazy way to chart-parse with categorial grammars. In *Proceedings of the 25th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 81–88, Stanford, CA, USA.

Stuart M. Shieber, Yves Schabes, and Fernando Pereira. 1995. Principles and implementation of deductive parsing. *Journal of Logic Programming*, 24(1–2):3–36.

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

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

Masaru Tomita. 1988. Graph-structured stack and natural language parsing. In *Proceedings of the 26th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 249–257, Buffalo, USA.

K. Vijay-Shanker and David J. Weir. 1990. Polynomial time parsing of combinatory categorial grammars. In *Proceedings of the 28th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 1–8, Pittsburgh, USA.

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

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

Jonathan Weese, Chris Callison-Burch, and Adam Lopez. 2012. Using categorial grammar to label translation

rules. In *Proceedings of the Seventh Workshop on Statistical Machine Translation*, pages 222–231, Montréal, Canada.

David J. Weir 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, USA.