On the Complexity of CCG Parsing

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

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

Peter Jonsson
Linköping University
Department of Computer and
Information Science
peter.jonsson@liu.se

We study the parsing complexity of Combinatory Categorial Grammar (CCG) in the formalism of Vijay-Shanker and Weir (1994). As our main result, we prove that any parsing algorithm for this formalism will take in the worst case exponential time when the size of the grammar, and not only the length of the input sentence, is included in the analysis. This sets the formalism of Vijay-Shanker and Weir (1994) apart from weakly equivalent formalisms such as Tree Adjoining Grammar, for which parsing can be performed in time polynomial in the combined size of grammar and input sentence. Our results contribute to a refined understanding of the class of mildly context-sensitive grammars, and inform the search for new, mildly context-sensitive versions of CCG.

1. Introduction

Combinatory Categorial Grammar (CCG; Steedman and Baldridge 2011) is a well-established grammatical framework that has supported a large amount of work both in linguistic analysis and natural language processing. From the perspective of linguistics, the two most prominent features of CCG are its tight coupling of syntactic and semantic information, and its capability to compactly encode this information entirely within the lexicon. Despite the strong lexicalization that characterizes CCG, it is able to handle non-local dependencies in a simple and effective way (Rimell, Clark, and Steedman 2009). After the release of annotated data sets (Hockenmaier and Steedman 2007), there has been a surge of interest in CCG within statistical and, more recently, neural natural

Submission received: 21 February 2017; revised version received: 4 May 2018; accepted for publication: 18 May 2018.

doi:10.1162/COLI_a_00324

language processing. The wide range of applications for which CCG has been used includes data-driven syntactic parsing (Clark and Curran 2007; Zhang and Clark 2011), natural language generation (White, Clark, and Moore 2010; Zhang and Clark 2015), machine translation (Lewis and Steedman 2013), and broad-coverage semantic parsing (Lewis and Steedman 2014; Lee, Lewis, and Zettlemoyer 2016).

In this article we study the parsing complexity of CCG. Our point of departure is the work of Vijay-Shanker and Weir (1990), who presented the first polynomial-time parsing algorithm for CCG. The runtime complexity of this algorithm is in $\mathcal{O}(n^6)$, where n is the length of the input sentence. This matches the runtime complexity of standard parsing algorithms for Tree Adjoining Grammar (TAG; Schabes 1990), which fits nicely with the celebrated result that CCG and TAG are weakly equivalent (Weir and Joshi 1988; Vijay-Shanker and Weir 1994). However, although the runtime of Vijay-Shanker and Weir's algorithm is polynomial in the length of the input sentence, it is *exponential* in the size of the grammar. This is in contrast with the situation for TAG, where the runtime is (roughly) quadratic with respect to grammar size (Schabes 1990). The only other polynomial-time parsing algorithms for CCG that we are aware of (Vijay-Shanker and Weir 1993; Kuhlmann and Satta 2014) exhibit the same behavior. Kuhlmann and Satta (2014) ask whether parsing may be inherently more complex for CCG than for TAG when grammar size is taken into account. Our main technical result in this article is that the answer to this question is positive: We show that *any* parsing algorithm for CCG in the formalism considered by Vijay-Shanker and Weir will *necessarily* take in the worst case exponential time when the size of the grammar is included in the analysis. Formally, we prove that the universal recognition problem for this formalism is EXPTIME-complete. The following paragraphs provide some context to this result.

The Mild Context-Sensitivity of Modern CCG. Our interest in the computational properties of CCG is motivated by our desire to better understand modern incarnations of this framework from a mathematical point of view. Theoretical work on CCG has always emphasized the importance of keeping the computational and generative power of the grammar as low as possible (see, for instance, Steedman 2000, page 23, and Baldridge 2002, Section 2.5), and in doing so has followed the tradition of the so-called mildly context-sensitive theories of grammar. The aforementioned polynomial-time parsing algorithm and the weak equivalence with TAG established the membership of CCG in this class of grammars even on a formal level. However, recent work has drawn attention to the fact that the specific formalism for which these results were obtained, and which we will refer to as VW-CCG (after Vijay-Shanker and Weir), differs from contemporary versions of CCG in several important aspects. In particular, it allows one to restrict and even ban the use of combinatory rules on a per-grammar basis, whereas modern CCG postulates one universal set of rules, controlled by a fully lexicalized mechanism based on typed slashes, as in other approaches to categorial grammar (Baldridge 2002; Steedman and Baldridge 2011). The difference is important because the weak equivalence result crucially depends on the availability of grammar-specific rule restrictions—without this feature, the generative power of VW-CCG is strictly smaller than that of TAG (Kuhlmann, Koller, and Satta 2015). At the same time, modern CCG includes combinatory rules that are absent from VW-CCG, specifically substitution and type-raising, and there is the possibility that this can counterbalance the loss of generative power that comes with the lexicalization of the rule control mechanism. Then again, these new rules are not supported by existing polynomial-time parsing algorithms. Moreover, the weak equivalence proof uses another feature of VW-CCG that is not available in contemporary versions of CCG: the ability to assign lexicon entries to the empty string. Such "empty categories" are ruled out by one of the fundamental linguistic principles of CCG, the Principle of Adjacency (Steedman 2000, page 54), and it is far from obvious that the weak equivalence proof can be re-written without them. In summary, the formalism of Vijay-Shanker and Weir is the only CCG formalism that has been proved to be weakly equivalent to TAG,[1] and the only one that has been shown to be parsable in polynomial time. As such, it is arguably the only CCG formalism that has been shown to be mildly context-sensitive, which is why we consider it to be of continued interest from a mathematical point of view. At the same time, we hope that the insights that we can obtain from the analysis of VW-CCG will eventually lead to the development of linguistically more adequate, provably mildly context-sensitive formalisms for CCG.

Universal Recognition. The **universal recognition problem** for a class of grammars $\mathcal{G}$ is the problem defined as follows: Given as input a grammar G in $\mathcal{G}$ and a string w, decide whether w is in $L(G)$, the language generated by G. The computational complexity of this problem is measured as a function of the combined size of G and w. The universal recognition problem should be contrasted with the **membership problem** for any specific grammar G in $\mathcal{G}$, whose complexity is measured as a function solely of the length of w. The complexity of the universal recognition problem is generally higher than that of the membership problem. For instance, the universal recognition problem for context-free grammars is PTIME-complete (complete for decision problems solvable in deterministic polynomial time), whereas the membership problem for these grammars defines the class LOGCFL (decision problems reducible in logarithmic space to a context-free language), which is generally conjectured to be a proper subset of PTIME.

The definitions of the universal recognition problem and the membership problem often generate some confusion. For instance, in applications such as parsing or translation, we work with a *fixed* grammar, so it might seem that the universal recognition problem is of little practical relevance. However, it is worth remembering that for these applications, we are primarily interested in the structural descriptions that the grammar assigns to a generated sentence, not in the membership of the sentence per se. Therefore, the universal recognition problem is a more accurate model of parsing than the membership problem, as the latter also admits decision procedures where the grammar is replaced with some other mechanism that may produce no or completely different descriptions than the ones we are interested in. The universal recognition problem is also favored when the ambition is to characterize parsing time in terms of all relevant inputs—both the length of the input string and the size and structure of the grammar (Ristad 1986). Such an analysis often reveals (and does so even in this article) how specific features of the grammar contribute to the complexity of the parsing task. More precisely, when investigating the universal recognition problem one expresses the computational complexity of parsing in terms of several parameters (other than the input string length), as for instance the number of nonterminals, maximum size of rules, or maximum length of unary derivations. This provides a much more fine-grained picture than the one that we obtain when analyzing the membership problem, and discloses the effects that each individual feature of the grammar has on parsing.

[1] Baldridge and Kruijff (2003) show that the weak generative power of their formalism for multi-modal CCG is *at most as* strong as that of TAG, but they do not show that it is *at least as* strong.

Structure of the Article. The remainder of this article is structured as follows. After presenting the VW-CCG formalism in Section 2, we first study in Section 3 the universal recognition problem for a restricted class of VW-CCG, where each category is "lexicalized" in the sense of the Principle of Adjacency. We show that for this subclass, universal recognition is NP-complete. Under the assumption that PTIME $\neq$ NP, this already implies our main result that parsing algorithms for VW-CCG will take in the worst case exponential time in the combined size of the grammar and the input string. In Section 4 we analyze the general case and show that the universal recognition problem for unrestricted VW-CCG is EXPTIME-complete. This is a stronger result than the one in Section 3, as it does not rely on any assumptions. However, we anticipate that many readers will be content with the result in Section 3, especially because the proofs of the more general result are considerably more complex. Finally, Section 5 is devoted to a general discussion of our results, its ramifications, and its relevance for current research.

2. Preliminaries

In this section we present the VW-CCG formalism. We assume the reader to be already familiar with the basic notions of categorial grammar, and in particular with the idea of categories as syntactic types. Like other categorial formalisms, a VW-CCG grammar has two central components: a **lexicon**, which specifies the categories for individual words, and a set of **rules**, which specify how to derive the categories of longer phrases from the categories of their constituent parts.

2.1 Lexicon

The VW-CCG lexicon is a set of pairs $\sigma := X$, where σ is a word (formalized as a symbol from some finite vocabulary) and X is a category. Formally, the set of **categories** over a given set $\mathcal{A}$ is the smallest set $\mathcal{C}(\mathcal{A})$ such that (i) $\mathcal{A} \subseteq \mathcal{C}(\mathcal{A})$ and (ii) if $X \in \mathcal{C}(\mathcal{A})$ and $Y \in \mathcal{C}(\mathcal{A})$ then $X/Y \in \mathcal{C}(\mathcal{A})$ and $X\backslash Y \in \mathcal{C}(\mathcal{A})$. Categories of form (i) are called **atomic**, those of form (ii) are called **complex**.

Categories as Stacks. Categories are usually viewed as directed versions of the function types in the simply typed lambda calculus. Here we follow authors such as Baldridge (2002, page 195f.) and view them as **stacks**. We treat slashes as left-associative operators and omit unnecessary parentheses. This lets us write every category $X \in \mathcal{C}(\mathcal{A})$ in the form

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

where $m \geq 0$, $A \in \mathcal{A}$ 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. Based on this notation we view X as a pair consisting of the target A and a stack whose elements are the arguments of X, with the argument $|_m X_m$ at the top of the stack. Note that arguments can in general contain complex categories.

Example 1
In the derivation shown in Figure 1, lexicon assignments are indicated by dotted lines: The verb *prove*, for example, is associated with the complex category (S\NP)/NP. In our notation, the same category can also be written as S\NP/NP. The target of this category

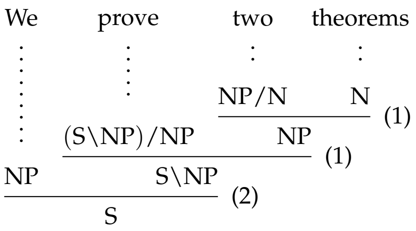

Figure 1
A sample derivation tree for the sentence "We prove two theorems."

is S, and the stack consists of the two arguments \NP and /NP, with /NP at the top of the stack. Note that we follow the standard convention in CCG and draw derivations with the leaves at the top and the root at the bottom.

2.2 Rules

The rules of VW-CCG are derived from two **combinatory schemata**,

$$\begin{array}{llll} X/Y & Y|_1 Z_1 \cdots |_d Z_d & \Rightarrow & X|_1 Z_1 \cdots |_d Z_d \qquad \text{(forward schema)} \\ Y|_1 Z_1 \cdots |_d Z_d & X\backslash Y & \Rightarrow & X|_1 Z_1 \cdots |_d Z_d \qquad \text{(backward schema)} \end{array}$$

where $d \geq 0$, the $|_i$ are slashes (forward or backward), and X, Y, and the Z_i are variables ranging over categories. Each schema specifies how to combine a primary input category (highlighted) and a secondary input category into an output category. The integer d is called the **degree** of the schema. Rules derived from the schemata where $d = 0$ are called **application** rules; the others are called **composition** rules.

Example 2
Figure 2 shows all (six) combinatory schemata with degree at most 1, together with their conventional names (these are labeled as rules (1)–(6).) In the derivation shown in Figure 1, each branching of the tree is annotated with the schema used in that step.

A **combinatory rule** over a set of categories $\mathcal{C}(\mathcal{A})$ is obtained from a combinatory schema by optionally restricting the ranges of some of the variables. Two types of

$$\begin{array}{lllll} X/Y \quad Y & \Rightarrow & X & \text{(forward application)} & (1) \\ Y \quad X\backslash Y & \Rightarrow & X & \text{(backward application)} & (2) \\[1ex] X/Y \quad Y/Z & \Rightarrow & X/Z & \text{(forward harmonic composition)} & (3) \\ X/Y \quad Y\backslash Z & \Rightarrow & X\backslash Z & \text{(forward crossed composition)} & (4) \\ Y\backslash Z \quad X\backslash Y & \Rightarrow & X\backslash Z & \text{(backward harmonic composition)} & (5) \\ Y/Z \quad X\backslash Y & \Rightarrow & X/Z & \text{(backward crossed composition)} & (6) \end{array}$$

Figure 2
The combinatory schemata with degree 0 (application; top) and 1 (composition; bottom).

restrictions are possible: (i) We may require the variable Y or any of the Z_i to take the value of some specific category in $\mathcal{C}(\mathcal{A})$. For example, we could derive a restricted version of backward crossed composition that applies only if $Y = \mathrm{S\backslash NP}$:

$$(\mathrm{S\backslash NP})/Z \quad X\backslash(\mathrm{S\backslash NP}) \quad \Rightarrow \quad X/Z \tag{7}$$

(ii) We may restrict the range of the variable X to categories with a specific target $A \in \mathcal{A}$. For example, we could restrict backward crossed composition to apply only in situations where the target of X is S, the category of complete sentences. We denote the resulting rule using the "\$" notation of Steedman (2000), where the symbol \$ is used as a variable for the part of the category stack below the topmost stack element:

$$Y/Z \quad \mathrm{S}\$\backslash Y \quad \Rightarrow \quad \mathrm{S}\$/Z \tag{8}$$

Example 3
Backward crossed composition (Equation (6)) can be used for the analysis of heavy NP shift in sentences such as *Kahn blocked skillfully a powerful shot by Rivaldo* (example from Baldridge 2002). A derivation for this sentence is shown in the upper half of Figure 3. However, the schema cannot be universally active in English, as this would cause the grammar to also accept strings such as **Kahn blocked skillfully a powerful by Rivaldo shot*, which is witnessed by the derivation in the lower half of Figure 3 (a dagger † marks the problematic step). To rule out this derivation, instead of the unrestricted schema, a VW-CCG grammar of English may select only certain instances of this schema as rules—in particular, instances that combine the two restrictions in Equations (7) and (8). In this way the unwanted derivation in Figure 3 can be blocked, while the other derivation

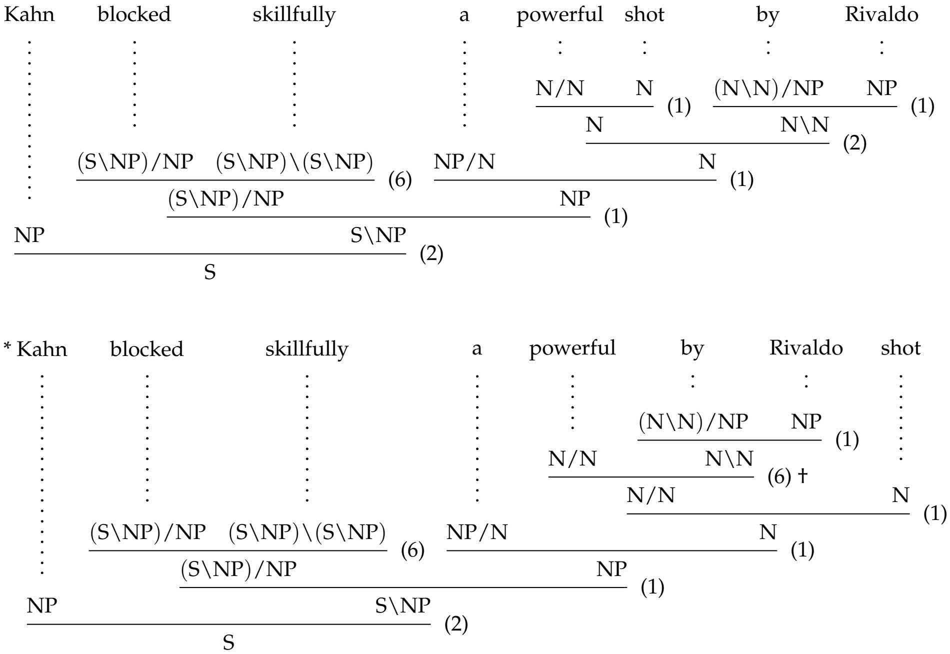

Figure 3
Overgeneration caused by unrestricted backward crossed composition.

is still admissible. Other syntactic phenomena require other grammar-specific restrictions, including the complete ban of certain combinatory schemata (cf. Steedman 2000, Sections 4.2.1–4.2.2).

A **ground instance** of a combinatory rule over $\mathcal{C}(\mathcal{A})$ is obtained by replacing every variable with a concrete category from $\mathcal{C}(\mathcal{A})$. We denote ground instances using a triple arrow. For example, the two instances of backward crossed composition in Figure 3 are:

$$\mathrm{(S\backslash NP)/NP}\ \ \mathrm{(S\backslash NP)\backslash(S\backslash NP)} \Rrightarrow \mathrm{(S\backslash NP)/NP} \qquad \mathrm{N/N}\ \ \mathrm{N\backslash N} \Rrightarrow \mathrm{N/N}$$

Every combinatory rule over $\mathcal{C}(\mathcal{A})$ has infinitely many ground instances. In particular, the variable X in such a rule can be replaced with infinitely many concrete categories.

2.3 Grammars

A VW-CCG grammar fixes a finite lexicon and a finite set of combinatory rules. Formally, a grammar is defined as a structure $G = (\Sigma, \mathcal{A}, :=, R, S)$ where Σ is a finite vocabulary, $\mathcal{A}$ is a finite set of atomic categories, $:=$ is a finite relation between the sets $\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$ and $\mathcal{C}(\mathcal{A})$, R is a finite set of combinatory rules over $\mathcal{C}(\mathcal{A})$, and $S \in \mathcal{A}$ is a distinguished atomic category. In what follows, we simply refer to the elements of R as rules.

Derivations. Derivations of G are represented as binary trees whose nodes are labeled with either lexicon entries (leaves) or categories (inner nodes). In order to represent such trees by linear terms, we use the following notation. Let A be some unspecified alphabet. For $a \in A$, the term a represents a tree with a single node labeled by a. For tree terms $t_1, \dots, t_m$, $m \geq 1$, the term $a(t_1, \dots, t_m)$ represents the tree whose root node is labeled by a and has m children, which are the root nodes of the trees represented by $t_1, \dots, t_m$. With this notation, we define the set of **derivation trees** of G and the associated mappings *top* (which returns the category at the root node of the tree) and *yield* (which returns the left-to-right concatenation of the symbols at the leaves) recursively as follows:

- Every lexicon entry $\sigma := X$ forms a (single-node) derivation tree τ. We define $top(\tau) = X$ and $yield(\tau) = \sigma$.
- Let τ_L and τ_R be derivation trees with $top(\tau_L) = X_L$, $yield(\tau_L) = w_L$, $top(\tau_R) = X_R$, and $yield(\tau_R) = w_R$, and let $X_L\ X_R \Rrightarrow X$ be a ground instance of some combinatory rule in R. Then $\tau = X(X_L, X_R)$ is a derivation tree. We define $top(\tau) = X$ and $yield(\tau) = w_L w_R$, where juxtaposition denotes string concatenation.

The connection between this formal definition and the graphical notation for derivation trees that we have used in Figures 1 and 3 should be clear. The only thing to note is that in a formal derivation tree, leaf nodes correspond to lexicon entry $\sigma := X$, whereas in our graphical notation, leaf nodes are split into a parent node with the category X and a child, leaf node with the symbol σ.

Generated Language. Based on the concept of derivation trees, we can now define the string language generated by a grammar G. The grammar G generates a string w if there exists a derivation tree whose root node is labeled with the distinguished category S and whose yield equals w. The **language** generated by G, denoted by $L(G)$, is the set of all strings generated by G. As mentioned before, Weir and Joshi (1988) and

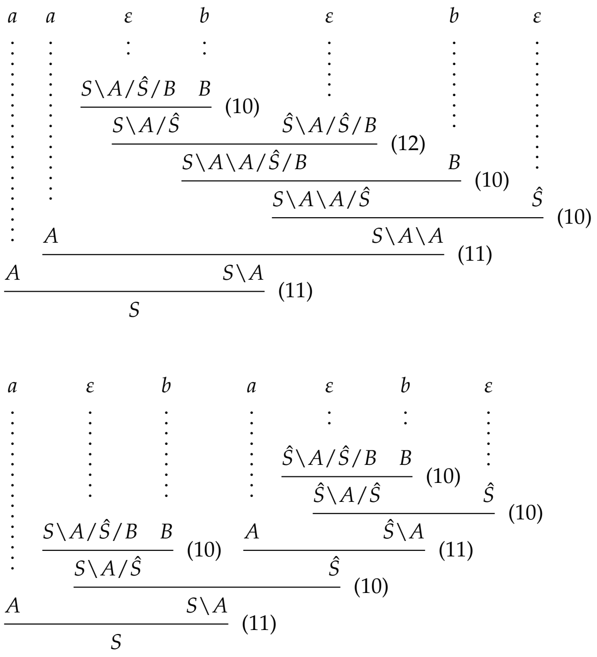

Figure 4
Two derivations of the grammar from Example 3.3 of Vijay-Shanker and Weir (1994).

Vijay-Shanker and Weir (1994) show VW-CCG generates the same class of languages as TAG (Joshi and Schabes 1997).

Example 4
Vijay-Shanker and Weir (1994) construct the following VW-CCG G (their Example 3.3). We only specify the lexicon and the set of rules; the vocabulary, set of atomic categories, and distinguished atomic category are left implicit. The lexicon is defined as follows:

$$a := A\,, \quad b := B\,, \quad \varepsilon := S\backslash A/\hat{S}/B\,, \quad \varepsilon := \hat{S}\backslash A/\hat{S}/B\,, \quad \varepsilon := S\,, \quad \varepsilon := \hat{S} \tag{9}$$

The set of rules consists of all instances of application and all instances of forward composition of degree at most 3 where the target of the secondary input category is restricted to one of the "hatted" categories. We write Y for a variable restricted to the set $\{S, A, B\}$, $\hat{Y}$ for a variable restricted to the set $\{\hat{S}, \hat{A}, \hat{B}\}$, and Z_i for an unrestricted variable. As before, the $|_i$ are slashes (forward or backward).

$$X/Y \quad Y \quad \Rightarrow \quad X \tag{10}$$

$$Y \quad X\backslash Y \quad \Rightarrow \quad X \tag{11}$$

$$X/\hat{Y} \quad \hat{Y}|_1 Z_1 \cdots |_d Z_d \quad \Rightarrow \quad X|_1 Z_1 \cdots |_d Z_d \qquad \text{where } 0 \leq d \leq 3 \tag{12}$$

As witnessed by the derivations in Figure 4, the language generated by this grammar contains the subsets $\{a^n b^n \mid n \geq 0\}$ and $\{(ab)^n \mid n \geq 0\}$.[2]

[2] Vijay-Shanker and Weir (1994) are mistaken in claiming that this grammar generates *exactly* $\{a^n b^n \mid n \geq 0\}$.

The ability to impose restrictions on the applicability of rules plays an important role in terms of generative power: without them, VW-CCG is strictly less powerful than TAG (Kuhlmann, Koller, and Satta 2015).

3. Complexity Without Categories for the Empty String

As already mentioned, the ability of VW-CCG to assign lexicon entries to the empty string contradicts one of the central linguistic principles of CCG, the Principle of Adjacency, by which combinatory rules may only apply to entities that are phonologically realized (Steedman 2000, page 54). In this section we therefore first investigate the computational complexity of the universal recognition problem for a restricted version of VW-CCG, where this feature is dropped and every lexical category is projected by an overt word. We will say that a grammar G whose lexicon does not contain any assignments of the form $\varepsilon := X$ is **ε-free**. We show the following result:

Theorem 1
The universal recognition problem for ε-free VW-CCG is NP-complete.

We split the proof of this theorem into two parts: Section 3.1 shows hardness, and Section 3.2 shows membership. Section 3.3 contains a brief discussion of the result. For a gentle introduction to computational complexity and the relevant proof techniques, we refer the reader to Papadimitriou (1994).

3.1 NP-Hardness

Our hardness proof is by a polynomial-time reduction from the Boolean Satisfiability Problem (SAT) to the universal recognition problem for ε-free VW-CCG. Because SAT is an NP-hard problem, this proves that the recognition problem is NP-hard as well. An instance of SAT is given by a Boolean formula ϕ in conjunctive normal form. This means that ϕ is a conjunction of clauses c_i, where each clause consists of a disjunction of one or more literals. A literal is either a variable v_j or a negated variable $\overline{v_j}$. The question asked about ϕ is whether it is satisfiable—that is, whether there is a truth assignment to the variables that makes ϕ evaluate to $\mathbf{1}$ (true). Our reduction is a polynomial-time procedure for transforming an arbitrary instance ϕ into an ε-free grammar G and an input string w such that ϕ is satisfiable if and only if $w \in L(G)$. We additionally note that the combined size of G and w is polynomial in the total number of literals in ϕ, and thus obtain the following:

Lemma 1
The universal recognition problem for ε-free VW-CCG is NP-hard.

We start with a description of how to obtain the input string w in Section 3.1.1, and then turn to the grammar G. The lexicon and the rules of the grammar G will be set up in such a way that every derivation for w consists of three clearly separated parts. We will present these parts in sequence in Sections 3.1.2–3.1.4, introducing the relevant lexicon entries and rules as we go along. The vocabulary and the set of atomic categories will be implicit. We will say that we **construct** the string w and the grammar G based on ϕ. Throughout the description of this construction we write m for the number of clauses in ϕ and n for the total number of distinct variables in ϕ. The index i will always range over values from 1 to m (clauses), and the index j will range over values from 1 to n (variables). To illustrate our construction we will use the following instance of SAT:

$$\phi = (v_1 \vee \overline{v_2}) \wedge (v_1 \vee v_2) \wedge (\overline{v_1} \vee \overline{v_2})$$

For this instance we have $m = 3$ and $n = 2$. We can verify that the only truth assignment satisfying ϕ is $\{v_1 \mapsto \mathbf{1}, v_2 \mapsto \mathbf{0}\}$. We set $c_1 = (v_1 \vee \overline{v_2})$, $c_2 = (v_1 \vee v_2)$, and $c_3 = (\overline{v_1} \vee \overline{v_2})$.

3.1.1 Input String. We construct the input string as

$$w = c_m \cdots c_1 c_0 v_1 \cdots v_n v_{n+1} d_n \cdots d_1$$

where the c_i and v_j are symbols representing the clauses and variables of the input formula ϕ, respectively. The symbols c_0 and v_{n+1} as well as the d_j are special symbols that we use for technical reasons, as explained subsequently. For our running example we have $w = c_3 c_2 c_1 c_0 v_1 v_2 v_3 d_2 d_1$.

3.1.2 Guessing a Truth Assignment. The first part of a derivation for w "guesses" a truth assignment for the variables in ϕ by assigning a complex category to the substring $c_0 v_1 \cdots v_n v_{n+1}$. Figure 5 shows what this could look like for our running example. Reading from the leaves to the root, for every symbol v_j in w, the derivation nondeterministically chooses between two lexicon entries, $v_j := [¢]/[v_j \mapsto \mathbf{1}]/[¢]$ and $v_j := [¢]/[v_j \mapsto \mathbf{0}]/[¢]$; these entries represent the two possible truth assignments to the variable. Note that we use square brackets to denote atomic categories. The derivation then uses compositions (13) and (14) to "push" these variable-specific categories to the argument stack of the lexical category for the special symbol c_0, and a final application (15) to yield a complex category that encodes the complete assignment.

Lexicon Entries and Rules. To support derivations such as the one in Figure 5, we introduce the following lexicon entries:

$$c_0 := [c_0]/[¢] \qquad v_j := [¢]/[v_j \mapsto \mathbf{1}]/[¢] \qquad v_j := [¢]/[v_j \mapsto \mathbf{0}]/[¢] \qquad v_{n+1} := [¢]$$

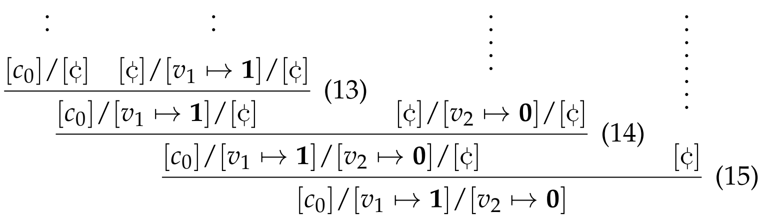
c_0
v_1
v_2
v_3
$[c_0]/[¢]$
$[¢]/[v_1 \mapsto \mathbf{1}]/[¢]$
(13)
$[c_0]/[v_1 \mapsto \mathbf{1}]/[¢]$
$[¢]/[v_2 \mapsto \mathbf{0}]/[¢]$
(14)
$[c_0]/[v_1 \mapsto \mathbf{1}]/[v_2 \mapsto \mathbf{0}]/[¢]$
$[¢]$
(15)
$[c_0]/[v_1 \mapsto \mathbf{1}]/[v_2 \mapsto \mathbf{0}]$

Figure 5
Derivation fragment that "guesses" the truth assignment for the running example.

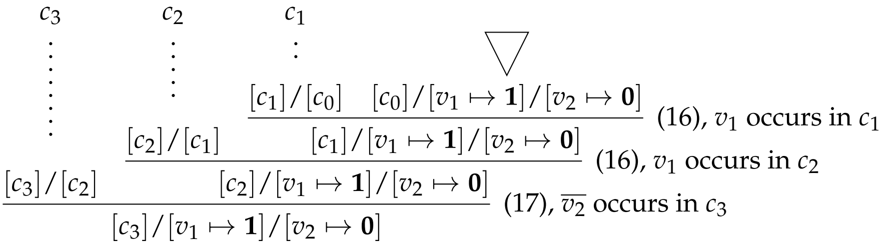
c_3
c_2
c_1
$[c_1]/[c_0]$
$[c_0]/[v_1 \mapsto \mathbf{1}]/[v_2 \mapsto \mathbf{0}]$
(16), v_1 occurs in c_1
$[c_2]/[c_1]$
$[c_1]/[v_1 \mapsto \mathbf{1}]/[v_2 \mapsto \mathbf{0}]$
(16), v_1 occurs in c_2
$[c_3]/[c_2]$
$[c_2]/[v_1 \mapsto \mathbf{1}]/[v_2 \mapsto \mathbf{0}]$
(17), $\overline{v_2}$ occurs in c_3
$[c_3]/[v_1 \mapsto \mathbf{1}]/[v_2 \mapsto \mathbf{0}]$

Figure 6
Derivation fragment that verifies the assignment for the running example. The white triangle represents the derivation shown in Figure 5.

We also introduce rules according to the following restricted schemata. Schemata (13) and (14) yield composition rules of degree 2; schema (15) yields application rules.

$$X/[¢] \quad [¢]/[v_j \mapsto \mathbf{1}]/[¢] \quad \Rightarrow \quad X/[v_j \mapsto \mathbf{1}]/[¢] \tag{13}$$

$$X/[¢] \quad [¢]/[v_j \mapsto \mathbf{0}]/[¢] \quad \Rightarrow \quad X/[v_j \mapsto \mathbf{0}]/[¢] \tag{14}$$

$$X/[¢] \quad [¢] \quad \Rightarrow \quad X \tag{15}$$

Derivational Ambiguity. It is worth mentioning here that our rules support other derivation orders than the left-branching order shown in Figure 5. In particular, we could first combine the variable-specific categories with each other, and then combine the result with the category for c_0. One could rule out this derivational ambiguity by restricting the target of the primary input of each of the rules above to the category $[c_0]$, obtaining rules such as the following:[3]

$$[c_0]\,\$\,/[¢] \quad [¢]/[v_j \mapsto \mathbf{1}]/[¢] \quad \Rightarrow \quad [c_0]\,\$\,/[v_j \mapsto \mathbf{1}]/[¢] \tag{13'}$$

For the purposes of our reduction, the different derivation orders are irrelevant, and we therefore abstain from using target restrictions.

3.1.3 Verifying the Truth Assignment. The second part of a derivation for w verifies that the truth assignment hypothesized in the first part satisfies all clauses. It does so by using compositions to "pass" the stack of atomic categories encoding the truth assignment from one clause to the next, right to left. For the running example, this could be done as in Figure 6. Crucially, the rules used in this part are restricted in such a way that the assignment can be "passed" to the next clause c_i only if c_i is satisfied by at least one assignment $v_j \mapsto b$. This can happen in two ways: either the assignment sets $b = \mathbf{1}$ and v_j occurs in c_i, or the assignment sets $b = \mathbf{0}$ and the negated variable $\overline{v_j}$ occurs in c_i. For example, the lowermost composition (16) is licensed because v_1 occurs in c_1. At the end of this part of the derivation, we have a complex category encoding a truth assignment as before, but where we now also have checked that this assignment satisfies all clauses.

[3] Recall from Section 2.2 that \$ is a variable for the part of the category stack below the topmost element.

Lexicon Entries and Rules. To implement the second part of the derivation, for each clause c_i we introduce a lexicon entry $c_i := [c_i]/[c_{i-1}]$. Our rules make crucial use of variable restrictions. To introduce them we define the following notational shorthands:

$$\mathbf{1}_j \equiv /Y_1 \cdots /Y_{j-1}/[v_j \mapsto \mathbf{1}]/Y_{j+1} \cdots /Y_n$$

$$\mathbf{0}_j \equiv /Y_1 \cdots /Y_{j-1}/[v_j \mapsto \mathbf{0}]/Y_{j+1} \cdots /Y_n$$

Thus $\mathbf{1}_j$ is a sequence of n slash–variable pairs, except that the jth variable has been replaced with the concrete (atomic) category $[v_j \mapsto \mathbf{1}]$, and similarly for $\mathbf{0}_j$. With this notation, we include into G all rules that match one of the following two schemata:

$$X/[c_{i-1}] \quad [c_{i-1}]\mathbf{1}_j \quad \Rightarrow \quad X\mathbf{1}_j \qquad \text{if } v_j \text{ occurs in } c_i \tag{16}$$

$$X/[c_{i-1}] \quad [c_{i-1}]\mathbf{0}_j \quad \Rightarrow \quad X\mathbf{0}_j \qquad \text{if } \overline{v_j} \text{ occurs in } c_i \tag{17}$$

For example, the two lowermost (when reading the tree from the root to the leaves) compositions in Figure 6 are both instances of schema (16), but their use is licensed by two different variable–clause matchings.

Derivational Ambiguity. Similarly to what we noted for the first part (Section 3.1.2), the derivation of this part can proceed in several ways, because at each step we may be able to choose more than one rule to satisfy a clause c_i. For example, in the derivation in Figure 6, instead of using the rule of schema (16) with witness "v_1 occurs in c_2" we could also have used the rule of schema (17) with witness "$\overline{v_2}$ occurs in c_2." However, also as before, there is no need to eliminate this derivational ambiguity for the purposes of this reduction.

3.1.4 Finalizing the Derivation. The third and final part of a derivation of w reduces the complex category encoding the truth assignment to the distinguished category of G, which we define to be $[c_m]$, by a sequence of applications. For the running example, this is illustrated in Figure 7.

Lexicon Entries and Rules. This part of the derivation requires two lexicon entries for each of the auxiliary symbols: $d_j := [v_j \mapsto \mathbf{1}]$ and $d_j := [v_j \mapsto \mathbf{0}]$. The rules are:

$$X/[v_j \mapsto \mathbf{1}] \quad [v_j \mapsto \mathbf{1}] \quad \Rightarrow \quad X \tag{18}$$

$$X/[v_j \mapsto \mathbf{0}] \quad [v_j \mapsto \mathbf{0}] \quad \Rightarrow \quad X \tag{19}$$

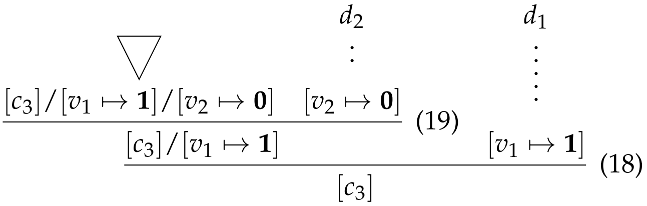
d_2
d_1
$[c_3]/[v_1 \mapsto \mathbf{1}]/[v_2 \mapsto \mathbf{0}]$
$[v_2 \mapsto \mathbf{0}]$
(19)
$[c_3]/[v_1 \mapsto \mathbf{1}]$
$[v_1 \mapsto \mathbf{1}]$
(18)
$[c_3]$

Figure 7
Final derivation fragment for the running example. The white triangle represents the derivation shown in Figure 6.

3.1.5 Time Complexity. We now analyze the time complexity of our reduction. For a given clause c_i, let $|c_i|$ be the number of literals in c_i. We define $|\phi| = \sum_i |c_i|$. The number of rules added in the first and the third part of the construction of G is in $\mathcal{O}(n)$, and the size of each such rule is bounded by a constant that does not depend on $|\phi|$. For the second part of the construction of G, for each clause c_i we add a number of rules that is at most $|c_i|$, and possibly less if there are repeated occurrences of some literal in c_i. Thus the total number of rules added in this part is in $\mathcal{O}(|\phi|)$. Each such rule has size in $\mathcal{O}(n)$. Putting everything together, and observing that n is in $\mathcal{O}(|\phi|)$, we see that the size of G is in $\mathcal{O}(|\phi|^2)$. It is not difficult to see then that our reduction can be carried out in time polynomial in the size of ϕ.

3.2 Membership in NP

We now turn to the membership part of Theorem 1:

Lemma 2
The universal recognition problem for ε-free VW-CCG is in NP.

For the proof of this lemma, we provide a polynomial-time nondeterministic algorithm that accepts an ε-free VW-CCG G and a string w if and only if G can derive w. We adopt the usual proof strategy where we first nondeterministically guess a derivation tree for w and then verify that this tree is valid.

Size of a Derivation Tree. We need to argue that the total number of characters needed to encode a derivation tree is polynomial in the combined size of G and w. Note that this involves both the tree structure itself and the lexicon entries and categories at the nodes of the tree. We start by observing that any derivation tree with ℓ leaf nodes (labeled with lexicon entries) has exactly $\ell - 1$ binary nodes (labeled with categories). Let τ by an arbitrary derivation tree for w. Because in G there are no lexical categories for the empty string, there is a one-to-one correspondence between the leaf nodes of τ and the symbols in w, which implies that the number of nodes of τ is exactly $2|w| - 1$.

Maximal Size of a Category. Now that we have bounded the number of nodes of τ, we will bound the sizes of the categories that these nodes are labeled with. Here, by the size of a category X, denoted by $|X|$, we simply mean the number of characters needed to write down X. Consider an internal node v of τ and its associated category X. In order to state an upper bound for $|X|$, we distinguish two cases: If v is a unary node, then $|X|$ is bounded by the largest size of a category in the lexicon of the grammar. We denote this quantity by λ. If v is a binary node, let $X = A|Y_1 \cdots |Y_q$, with A an atomic category. A rule of the grammar can increase the number of arguments of its primary category by at most d, where d is the maximum degree of a rule in the grammar. Let γ be the maximum number of arguments in a category in the lexicon. Because no more than $|w| - 1$ rules are used in τ, we conclude that q is bounded by $\gamma + d(|w| - 1)$. By Lemma 3.1 in Vijay-Shanker and Weir (1994), every argument Y_i in X must also occur as an argument in some category in the lexicon of G. Thus the size of each argument of X is bounded by the largest size of an argument appearing in a category in the lexicon, a quantity that we denote by α. Putting everything together, we have that $|X|$ is bounded by $1 + \alpha(\gamma + d(|w| - 1))$. From this it is not difficult to see that the overall space needed to encode our derivation tree τ for w along with all of the categories at its nodes is

$\mathcal{O}((\lambda + \alpha\gamma)|w| + \alpha d|w|^2)$. This is a polynomial in the combined size of the grammar and the input string.

Nondeterministic Algorithm. We can now provide our nondeterministic algorithm for testing whether G derives w. In a first step we write down a guess for a derivation tree τ for w based on the rules in G. Given our space bound on τ, we can carry out this step in time polynomial in the size of G and w. In a second step we visit each internal node v of τ and read its associated category X. If v is a unary node, we check whether X is a lexicon entry for the word at v's child. If v is a binary node, we check whether X can be obtained by some rule of the grammar applied to the categories at the two children of v. We accept if every check is successful. Even this second step can be carried out in time polynomial in the size of G and w. This concludes the proof of Lemma 2.

3.3 Discussion

In the previous sections we have shown that the universal recognition problem for ε-free VW-CCG is NP-complete (Theorem 1). This result is in contrast with the fact that, for the weakly equivalent TAG formalism, the universal recognition problem can be solved in polynomial time, and it naturally raises the question of what features of the VW-CCG formalism are the source of this additional complexity. We discuss this question on the basis of the reduction in our proof of Lemma 1. In this reduction we use a combination of three central features of VW-CCG, listed here. Dropping any of these features would break our reduction.

Lexical Ambiguity. The first feature of VW-CCG that we exploit in our construction is the ability to assign more than one category to some lexical items. In part 1 of the reduction (Section 3.1.2), this allows us to "guess" arbitrary truth assignments for the variables in the clause ϕ. However, the possibility to write grammars with lexical ambiguity is an essential feature of all interesting formalisms for natural language syntax, including also TAG. Therefore, at least in isolation, this feature does not seem to be able to explain the complexity of the universal recognition problem for VW-CCG. Even if our goal was to design a new version of VW-CCG that can be parsed in polynomial time in the size of the input grammar, we would not seriously consider giving up lexical ambiguity.

Unbounded Composition. The second feature of VW-CCG that we rely on is the availability of composition rules without a constant (with respect to the full class of grammars) bound on their degree. This feature is crucial for our encoding of truth assignments. In particular, without it we would not be able to percolate arbitrarily large truth assignments through derivation trees; our construction would work only for formulas with a bounded number of variables.

Unbounded composition has previously been discussed primarily in the context of generative power. Weir and Joshi (1988) show that allowing unrestricted use of *arbitrarily* many composition rules leads to a version of VW-CCG that is more powerful than the one considered here, where every grammar must restrict itself to a finite set of such rules. Other authors have suggested to put explicit (low) bounds on the maximal degree of composition. From a purely formal point of view, a bound as low as $d \leq 2$ may suffice: Weir and Joshi (1988) show how every VW-CCG grammar can be converted into a weakly equivalent Linear Index Grammar (LIG) (Gazdar 1987), and how every TAG can be converted into a weakly equivalent VW-CCG whose composition rules all have degree 2. Together with the weak equivalence of LIG and TAG (Vijay-Shanker and Joshi 1985), this shows that the subclass of VW-CCG grammars with degree of composition at most 2 can still generate the full class of languages.[4] For any degree-restricted subclass of grammars, our proof would break, which means that it may be possible (though not obvious) to devise a polynomial-time algorithm for the universal recognition problem. We will discuss unbounded composition further in Section 5.1.

Rule Restrictions. The third feature of VW-CCG that we exploit is its ability to put grammar-specific restrictions on combinatory rules. In particular, in part 2 of our construction (Section 3.1.3), we use rules whose secondary input categories contain a mix of variables and concrete categories, such as in Equation (16):

$$X/[c_{i-1}] \quad [c_{i-1}]\mathbf{1}_j \quad \Rightarrow \quad X\mathbf{1}_j \qquad\qquad \text{if } v_j \text{ occurs in } c_i$$

Like the availability of composition rules of unbounded degree, the ability to use rule restrictions seems to be a very powerful feature, and one that perhaps most clearly sets VW-CCG apart from TAG. Moreover, as already mentioned, rule restrictions also play a crucial role with respect to weak generative capacity (Kuhlmann, Koller, and Satta 2015).

Note that we could replace rules of the form (16) with rules without variables; but then, for fixed values of i and j and, say, for the assignment $[v_j \mapsto \mathbf{1}]$, we would have to include into the grammar all of the 2^{n-1} rules of the form

$$X/[c_{i-1}] \quad [c_{i-1}]/A_1 \cdots /A_{j-1}/[v_j \mapsto \mathbf{1}]/A_{j+1} \cdots /A_n \quad \Rightarrow \quad X/A_1 \cdots /A_n$$

where each A_h is a concrete atomic category of the form $[v_h \mapsto \mathbf{1}]$ or $[v_h \mapsto \mathbf{0}]$. This would break our proof because reductions must use polynomial time (and space). Note also that what is crucial here is not the use of either variables or concrete categories in a rule's secondary input; rather, it is the combination of the two that allows us to check clauses against truth assignments.

4. Complexity With Categories for the Empty String

In this section we investigate the computational complexity of the universal recognition problem for unrestricted VW-CCG, where one is allowed to assign lexicon entries even to the empty string. We show the following:

Theorem 2
The universal recognition problem for unrestricted VW-CCG is EXPTIME-complete.

The proof of this theorem is more involved than the proof of the NP-completeness result in Section 3. We start in Section 4.1 by introducing *alternating Turing machines* (Chandra, Kozen, and Stockmeyer 1981), which provide the computational framework for our proof. The use of alternating Turing machines instead of ordinary deterministic or nondeterministic Turing machines is crucial here: In order to simulate the computations of a Turing machine by a CCG grammar in a natural way, we need to restrict the

[4] Note, however, that the construction of Weir and Joshi (1988) does not produce ε-free grammars.

machine to use only polynomial space. However, if we used standard Turing machines with this space restriction, then we would only be able to prove PSPACE-hardness, a weaker result than the EXPTIME-completeness that we obtain from our proof. The hardness part of this proof is presented in Section 4.2, and the membership part in Section 4.3. We finally discuss our result in Section 4.4.

4.1 Alternating Turing Machines

The alternating Turing machine (ATM; Chandra, Kozen, and Stockmeyer 1981) is a generalization of the well-known nondeterministic Turing machine in which there are two types of states: existential states and universal states. When the machine is in an existential state, it accepts the input if there is at least one transition that eventually leads to an accepting state. In contrast, when the machine is in a universal state, it accepts input only if *every* possible transition eventually leads to an accepting state. A nondeterministic Turing machine can be viewed as an alternating Turing machine with no universal states.

As already mentioned, for our proof we use ATMs working in polynomial space, which means that the length of the tape is bounded by a polynomial in the length of the input. This resource-restricted model is well-known in the literature, and it exactly characterizes the class of all decision problems that are solvable by a *deterministic* Turing machine (i.e., a Turing machine where there is at most one possible transition given a state and a tape symbol) working in *exponential* time (Chandra, Kozen, and Stockmeyer 1981). This is the complexity class EXPTIME.

To simplify the notation and some of our proofs, we use ATMs that operate on a *circular tape*, and can only move their head to the right. The same model has previously been used by, among others, Jeż and Okhotin (2011). It is not hard to see that, as long as we work under the restriction to polynomial space, every move to the left in the standard model can be simulated by a (polynomial) number of moves to the right in the circular tape model. Thus, even ATMs with polynomially bounded circular tape precisely characterize EXPTIME.

Formal Definition. Formally, an **alternating Turing machine** is a structure

$$M = (Q, \Sigma, \delta, q_0, g)$$

where: Q is a finite set of **states**; Σ is an alphabet of **tape symbols**, which we assume includes the special blank symbol #; $\delta \subseteq (Q \times \Sigma) \times (Q \times \Sigma)$ is the **transition relation**; $q_0 \in Q$ is the initial state; and $g\colon\ Q \to \{\exists, \forall, \mathsf{A}, \mathsf{R}\}$ is a function that assigns a type to each state. The four different types for a state are existential ($\exists$), universal ($\forall$), accepting (A), and rejecting (R); their semantics will become clear subsequently.

We denote transitions in δ as $(q, a) \to (q', a')$. Transitions are subject to the restriction that the state to the left of the arrow must be either existential or universal. This means that no transition is possible out of an accepting or a rejecting state; when an ATM reaches such a state, it necessarily stops. To simplify the proof, we also require that for every universal state q and tape symbol a, there are exactly two transitions with left-hand side (q, a). This is without loss of generality: If a machine does not already have this property, then one can construct (in polynomial time) an equivalent polynomial-space ATM with circular tape satisfying it; a similar construction for general ATMs is sketched by Papadimitriou (1994, Theorem 8.2).

Configurations. Let $w \in \Sigma^*$ be an input string for w, and let $n = |w|$ and $m = p_M(|w|)$, where p_M is the machine-specific polynomial that defines the maximal tape length. A **configuration** of M relative to w is a pair $c = (q, \alpha)$, where $q \in Q$ is some state and $\alpha \in \Sigma^*$ is a sequence of tape symbols with length $|\alpha| = m$. The intended interpretation of c is that the current state of M is q, the content of the circular tape is represented by α, and the tape head is positioned to read the first symbol of α. In particular, the initial configuration of M for w, denoted by $I_M(w)$, takes the form $I_M(w) = (q_0, w\#^{m-n})$, meaning that, at the start of the computation, the machine is in the initial state, the tape consists of the n symbols of the input string w followed by $m - n$ blanks, and the tape head is positioned to read the first symbol of w. A configuration c is called existential, universal, accepting, or rejecting based on the type of its state q.

Successors. Let $t = (q, a) \rightarrow (q', a')$ be a transition. The intended interpretation of t is that if M is in state q and reads tape symbol a, then overwrites a with a', moves its tape head one cell to the right (which is always possible because the tape is circular), and continues the computation in state q'. Formally, let $c = (q, a\alpha)$ be a configuration of M. The **successor** of c with respect to t, denoted by $t(c)$, is the configuration $c' = (q', \alpha a')$, where the string $\alpha a'$ encodes the fact that the symbol a has been overwritten with a' and the circular tape has been rotated one position to the right, so that the head now is posititioned to read the first symbol of α. Note that, because of our restrictions on the transition relation, a universal configuration has exactly two successors.

Acceptance. We first discuss acceptance in ordinary nondeterministic Turing machines. As usual, a single machine configuration c may lead (in one step) to a number of successor configurations $c_1, \ldots, c_k$. Acceptance is recursively defined such that c leads to acceptance if and only if at least one of $c_1, \ldots, c_k$ leads to acceptance. One may view this as an existential condition on the successor configurations. In an alternating Turing machine, a configuration may be either existential or universal; in the universal case, c leads to acceptance if and only if *every* successor $c_1, \ldots, c_k$ leads to acceptance. To make this formal, we represent computations of M as trees whose nodes are labeled with configurations of M, and whose edges reflect the "successor of"-relation between configurations. Formally, the set of **accepting computations** is defined recursively as follows (recall our definition of tree terms in Section 2.3):

- Every accepting configuration c forms a one-node accepting computation.
- Let c be an existential configuration and let γ be an accepting computation whose root node is labeled with some successor of c. Then $c(\gamma)$ is an accepting computation.
- Let c be a universal configuration and let γ_1, γ_2 be accepting computations whose root nodes are labeled with the two successors of c. Then $c(\gamma_1, \gamma_2)$ is an accepting computation.

A sample accepting computation is shown in Figure 8. A machine M **accepts** a string w if there exists an accepting computation γ whose root node is labeled with the initial configuration $I_M(w)$. The set of all strings that are accepted by M is denoted by $L(M)$.

Characterization of EXPTIME. As already mentioned, the reason that we are interested in polynomial-space alternating Turing machines is that they exactly characterize the class of decision problems solvable in exponential time. This is expressed by the following lemma, which is basically Corollary 3.6 of Chandra, Kozen, and Stockmeyer (1981).

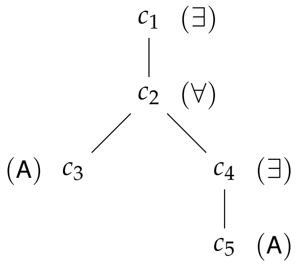

Figure 8
An accepting run. State types for each configuration c_i are indicated in parentheses.

Lemma 3
The following decision problem is EXPTIME-complete: Given a polynomial-space alternating Turing machine M and a string w, is $w \in L(M)$?

Because a polynomial-space circular-tape ATM can simulate any polynomial-space ATM at no additional asymptotic space cost, we conclude that Lemma 3 also holds for polynomial-space circular-tape ATMs. In the following we therefore use Lemma 3 to refer to polynomial-space circular-tape ATMs.

4.2 EXPTIME-Hardness

Let M be a polynomial-space circular-tape ATM and let w be an input string for M. In this section we show how to construct, in polynomial time and space, a VW-CCG grammar G such that $L(G) = \{\varepsilon\}$ if $w \in L(M)$, and $L(G) = \emptyset$ if $w \notin L(M)$. This means that we can test the condition $w \in L(M)$ by checking whether G generates the empty string. When combined with Lemma 3, this reduction proves the hardness part of Theorem 2:

Lemma 4
The universal recognition problem for unrestricted VW-CCG is EXPTIME-hard.

For the remainder of this section, we fix a polynomial-space circular-tape ATM $M = (Q, \Sigma, \delta, q_0, g)$ and an input string $w \in \Sigma^*$. Let p_M be the polynomial that bounds the length of the tape of M, and let $m = p_M(|w|)$.

Basic Idea. The basic idea behind our construction is straightforward: We will set up things in such a way that the derivations of G correspond to accepting computations of M for w. To illustrate this idea, Figure 9 shows the schematic structure of a derivation that corresponds to the accepting computation in Figure 8. Note that in order to make the correspondence more evident, contrary to our previous convention, we now draw the derivation with the root node at the top. We see that the derivation is composed of a number of smaller fragments (drawn as triangles). With the exception of the fragment at the top of the tree (which we need for technical reasons), there is one fragment per node of the accepting computation. Each fragment is labeled with a reference to the subsection of this article that describes how we set up the grammar G to derive that fragment.

One way to view our construction is that it establishes a structure-preserving map from accepting computations of M to derivations of G. This map replaces each configuration c in an accepting computation by a fragment, and continues the transformation at the subtrees under c. A fragment is like a small derivation tree, except that one or two

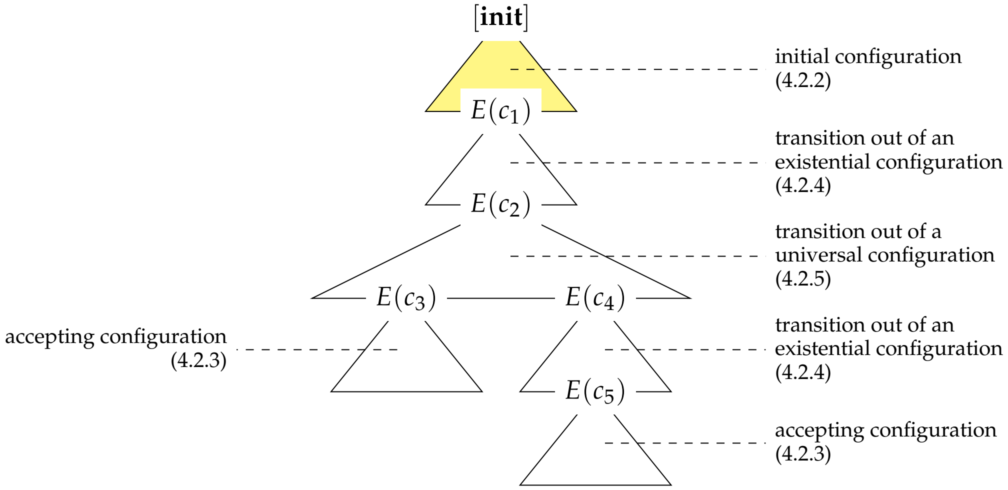
[**init**]
$E(c_1)$
$E(c_2)$
$E(c_3)$
$E(c_4)$
$E(c_5)$
initial configuration (4.2.2)
transition out of an existential configuration (4.2.4)
transition out of a universal configuration (4.2.5)
accepting configuration (4.2.3)
transition out of an existential configuration (4.2.4)
accepting configuration (4.2.3)

Figure 9
Schematic structure of the derivation in G that corresponds to the accepting computation in Figure 8. The derivation is displayed with the root node at the top to make the correspondence with Figure 8 more evident. Each $E(c_i)$ is the category that encodes the configuration c_i.

of its leaf nodes may be labeled with (possibly complex) categories instead of lexicon entries. These nodes, which we refer to as the **distinguished** leaf nodes of the fragment, serve as slots at which the fragments that result from the recursive transformation of the subtrees can be substituted. The root node of a fragment is labeled with a category that encodes the configuration c that the fragment replaces; we denote this category by $E(c)$. More specifically, the shape of the fragment depends on the type of c:

- For every accepting configuration c, the grammar derives a fragment with no distinguished leaf nodes. The category at the root node of this fragment is $E(c)$. The lexicon entries and rules required to derive the fragment are described in Section 4.2.3.
- For every existential configuration c and for every transition t that can be applied to c, the grammar derives a fragment with a single distinguished leaf node. The category at the root node of this fragment is $E(c)$, and the category at the distinguished leaf node is $E(t(c))$, the encoding of the successor of c under t (Section 4.2.4).
- For every universal configuration c, the grammar derives a fragment with two distinguished leaf nodes. The category at the root node of this fragment is $E(c)$, and the categories at the distinguished leaf nodes are $E(t_1(c))$ and $E(t_2(c))$, the encodings of the two successors of c (Section 4.2.5).
- Finally, for the initial configuration $I_M(w)$, the grammar derives a fragment with a single distinguished leaf node. The category at the root node of this fragment is the distinguished category of G, and the category at the distinguished leaf node is $E(I_M(w))$. This is the highlighted fragment in Figure 9 (Section 4.2.2).

Those leaf nodes of a fragment that are not distinguished leaf nodes will always be labeled with lexicon entries for the empty string, that is, entries of the form $\varepsilon := X$. Because of this, the only string that our grammar may accept is the empty string. As we will make sure that all (and only) the accepting computations of M over w receive corresponding derivations in G, this amounts to saying that G has at least one derivation if and only if $w \in L(M)$. This implies that we do not need a distinguished "reject" lexical category or any other mechanism that takes care of the case when the Turing machine rejects its input.

Technical Remark. Before we continue, we would like to make a technical remark that may make the following construction easier to understand. In the proof of Lemma 1, we constructed a grammar that produced derivations simulating a process of guessing and verifying a variable assignment for an instance of SAT. One feature of the construction is that this process has a purely linear (albeit nondeterministic) structure, which is reflected in the fact that the derivation trees produced by the grammar are essentially unary-branching. For such trees, it does not make much of a difference whether we read them bottom–up (from the leaves to the root) or top–down (from the root to the leaves), and in our construction we simply adopted the former perspective, which is the conventional one for CCG.

In this proof, because of the branching nature of the computations of M, the derivation trees of the grammar G will no longer be unary-branching; and because the branching in an accepting computation of M occurs on the paths from the initial configuration to the accepting configurations, the derivation trees of the grammar G need to have the encoding of the initial configuration at the root and the encodings of the accepting configurations at the leaves. This will require us to change perspective and read the derivation trees top–down—and consequently the rules of G from the output category to the input categories. This is the reverse of what is conventional for CCG.

4.2.1 Encoding Configurations. We start the presentation of the construction of G by explaining how we encode configurations of M as categories. Let $c = (q, a_1 \cdots, a_m)$ be a configuration of M. We encode this configuration by a category

$$E(c) = [q]/[a_1] \cdots /[a_m]$$

where we follow the same convention as in Section 3.1 and use square brackets to represent atomic categories. Note that in this encoding, the target of $E(c)$ is an atomic category representing the current state, and the arguments of $E(c)$ represent the circular tape, with the innermost argument corresponding to the symbol under the tape head. With this representation, the encoding of the successor of the configuration c under a transition $t = (q, a_1) \to (q', a')$ can be written as

$$E(t(c)) = [q']/[a_2] \cdots /[a_m]/[a']$$

4.2.2 Initial Configuration. We now present the derivation fragment for the initial configuration of M for w. Let $c_I = I_M(w)$. To give a concrete example, suppose that this configuration takes the form $c_I = (q_0, ab)$. Then the corresponding fragment looks as in Figure 10. The category at the root node is the distinguished category [**init**]. The derivation starts by nondeterministically pushing symbols to the tape stack of the categories along the path (highlighted in Figure 10) from the root to the distinguished node. This is done through rules of type (20). In a last step (rule (21)), the derivation checks whether the tape stack matches the initial tape content ab, and simultaneously changes the target from [**init**] to $[q_0]$. After this, the category at the distinguished leaf of the fragment is $E(c_I)$. We will see the idea of "nondeterministic guessing followed by

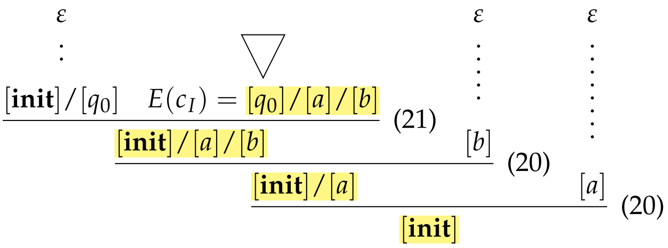

Figure 10
Derivation fragment for an initial configuration $c_I = (q_0, ab)$. Note how the symbols of c_I are collected in several steps along the highlighted path.

verification" once again in Section 4.2.5, where it will be used for making copies of the tape stack. The reader may rightfully wonder whether there are more direct methods for performing such straightforward manipulations of the tape stack. Unfortunately, we have not been able to define such methods within the VW-CCG formalism.

Lexicon Entries and Rules. More generally now, assume that the initial configuration for M on w is $c_I = (q_0, a_1 \cdots a_m)$, where $w = a_1 \cdots a_n$ and $a_h = \#$ for each h with $n < h \leq m$. To support fragments as the one in Figure 10, we introduce lexicon entries $\varepsilon := [\mathbf{init}]/[q_0]$ and $\varepsilon := [a]$, where $a \in \Sigma$ is any tape symbol. We also introduce the following rules:

$$[\mathbf{init}]\,\$\,/\,[a] \quad [a] \quad \Rightarrow \quad [\mathbf{init}]\,\$ \tag{20}$$

$$[\mathbf{init}]\,\$\,/\,[q_0] \quad [q_0]\,/\,[a_1]\cdots/\,[a_m] \quad \Rightarrow \quad [\mathbf{init}]\,\$\,/\,[a_1]\cdots/\,[a_m] \tag{21}$$

The \$ symbol is, as usual, a variable for the part of the category stack below the topmost stack element. A rule of the form (20) allows the application of a category with target $[\mathbf{init}]$ to any atomic category $[a]$ representing a tape symbol; this implements the nondeterministic pushing to the tape stack that we introduced above. A rule of the form (21) is a composition rule of degree m that restricts the target of the primary input category to the distinguished category $[\mathbf{init}]$, and the secondary input to the category $E(c_I)$. This implements the check in the final step of the fragment—if the category at the distinguished leaf does *not* encode the initial configuration at this point, then the derivation will reach a dead end.

Computational Complexity. We now deal with the computational resources required by this step of the construction. Each of the lexicon entries above is size-bounded by a constant that does not depend on $|M|$ or $|w|$. This size bound also holds for each rule of the form (20). The size of a rule of the form (21) is in $\mathcal{O}(m)$. We can then construct and store each lexical entry and each rule with time (and space) in $\mathcal{O}(m)$. Furthermore, the total number of lexicon entries and rules added to the grammar in this step is in $\mathcal{O}(|\Sigma|)$. We thus conclude that this step of the construction can be carried out in time (and space) polynomial in $|M|$ and $|w|$.

4.2.3 Accepting Configurations. Next, we present lexicon entries and rules needed to terminate derivations of the accepting configurations of M. To give a concrete example, suppose that $c = (q, ab)$ is accepting, and that the grammar has already derived the category $E(c)$. Then the grammar also derives the fragment shown in Figure 11.

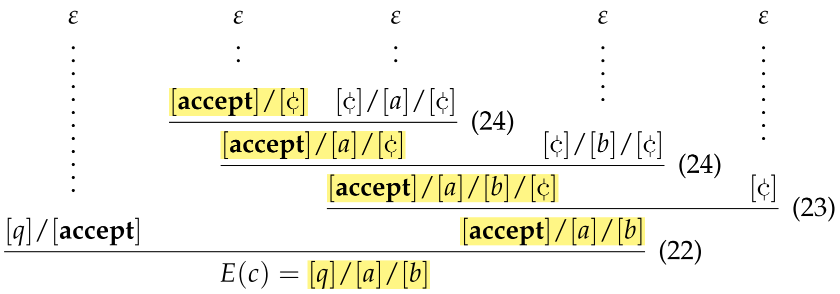

Figure 11
Derivation fragment for an accepting configuration $c = (q, ab)$.

Following the highlighted path from the root to the leaf, the derivation first checks whether $E(c)$ indeed encodes an accepting configuration, and then changes the target to a special atomic category [**accept**] shown in rule (22). After this, the fragment empties the tape stack, using derivations similar to those that we used to assemble the truth assignment in Section 3.1.2 (see Figure 5).

Lexicon Entries and Rules. Let $q \in Q$ with $g(q) = \mathsf{A}$, and let $a \in \Sigma$. We introduce the following lexicon entries:

$$\varepsilon := [q] \,/\, [\mathbf{accept}] \qquad \varepsilon := [\mathbf{accept}] \,/\, [¢] \qquad \varepsilon := [¢] \,/\, [a] \,/\, [¢] \qquad \varepsilon := [¢]$$

We also introduce the following rules:

$$\begin{aligned} [q]\,\$\,/\,[\mathbf{accept}] \quad [\mathbf{accept}]\,/\,X_1 \cdots /\,X_m \quad &\Rightarrow \quad [q]\,\$\,/\,X_1 \cdots /\,X_m && (22)\\ [\mathbf{accept}]\,\$\,/\,[¢] \quad [¢] \quad &\Rightarrow \quad [\mathbf{accept}]\,\$ && (23)\\ [\mathbf{accept}]\,\$\,/\,[¢] \quad [¢]\,/\,[a]\,/\,[¢] \quad &\Rightarrow \quad [\mathbf{accept}]\,\$\,/\,[a]\,/\,[¢] && (24) \end{aligned}$$

A rule of the form (22) is a composition rule of degree m that restricts the target of its primary input to an accepting state; this ensures that only categories encoding accepting configurations can yield subderivations of the form shown in Figure 11. The derivation will either rewrite the configuration (for existential and universal configurations—the details will be given in Sections 4.2.4 and 4.2.5, respectively), or else will reach a dead end (for rejecting configurations). The only rules that can be used after a rule of the form (22) are rules of the forms (23) and (24), which jointly implement the emptying of the tape stack that we described earlier.

Computational Complexity. Each of the lexicon entries here is size-bounded by a constant that does not depend on $|M|$ and $|w|$. This size bound also holds for rules of the form of (23) and (24). The sizes of rules of the form (22) are in $\mathcal{O}(m)$. The number of lexicon entries and rules that we add to G in this step is in $\mathcal{O}(|\Sigma|)$.

4.2.4 Transitions Out of Existential Configurations. We now turn to the fragments that simulate transitions out of existential configurations. Figure 12 shows what such a fragment looks like for a configuration $c = (q, ab)$ and a transition $t = (q, a) \to (q', a')$. The derivation starts by checking whether the category at the root node indeed encodes an existential configuration, and then changes the target to the transition-specific

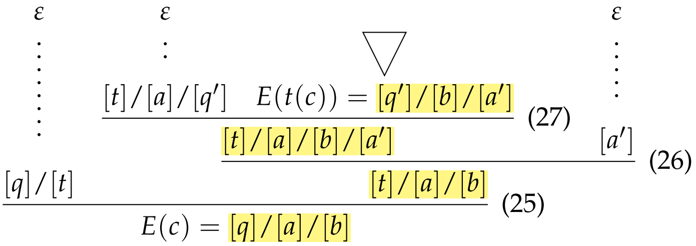

Figure 12
Fragment for a transition $t = (q, a) \to (q', a')$ out of an existential configuration $c = (q, ab)$.

category $[t]$ (25). The derivation then extends the tape stack by the new symbol a' (26). In a last step it simultaneously discards the category for the previous tape symbol a and changes the target to $[q']$ (27). After this, the category at the distinguished leaf encodes the configuration $t(c) = (q', ba')$. We remind the reader that an ATM accepts an existential configuration if and only if there is at least one transition that leads to an accepting configuration. In the grammar G, this corresponds to the fact that there exists an applicable rule that leads to acceptance. Therefore, we do not need to explicitly simulate all possible transitions. For universal configurations (which we will consider in Section 4.2.5), the situation is different, which will necessitate a more involved construction.

Lexicon Entries and Rules. Let $q \in Q$ be any existential state, and let $t = (q, a) \to (q', a')$ be any transition out of q. We introduce the following new lexicon entries:

$$\varepsilon := [q] / [t] \qquad \varepsilon := [t] / [a] / [q']$$

We also reuse the lexicon entries $\varepsilon := [a]$ that we introduced in Section 4.2.2. Finally, we introduce the following rules:

$$[q]\,\$\,/[t] \quad [t]/X_1 \cdots /X_m \quad \Rightarrow \quad [q]\,\$\,/X_1 \cdots /X_m \tag{25}$$

$$[t]\,\$\,/[a'] \quad [a'] \quad \Rightarrow \quad [t]\,\$ \tag{26}$$

$$[t]\,\$\,/[q'] \quad [q']/X_1 \cdots /X_m \quad \Rightarrow \quad [t]\,\$\,/X_1 \cdots /X_m \tag{27}$$

A rule of the form (25) is a composition rule of degree m that simultaneously restricts the target of its primary input to q and the target of its secondary input to t. A rule of the form (26) is an application rule that matches t (the target of its primary input) with the tape symbol a' produced by t (its secondary input). A rule of the form (27) is a composition rule of degree m that matches t (the target of its primary input) with the state q' resulting from the application of t.

Computational Complexity. Again, each of the lexical entries and rules has size in $\mathcal{O}(m)$. The number of rules of the form (27) added to the grammar is bounded by the possible choices of the transition t (q' is unique, given t), and is thus a polynomial function of $|M|$. Similar analyses apply to the other rules and lexical entries. We thus conclude that the overall contribution to $|G|$ in this step is polynomial in the size of the input, and the construction can be carried out in polynomial time, too.

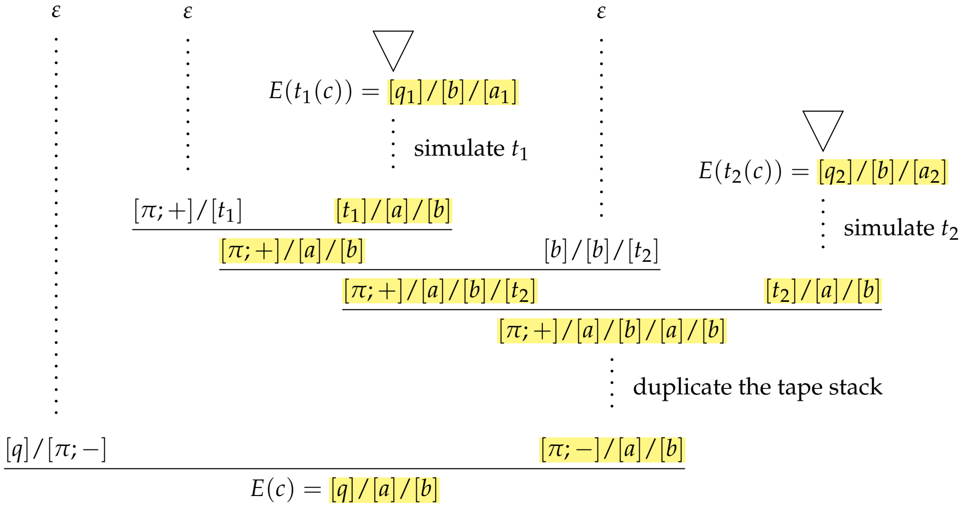

Figure 13
Bird's-eye view of the derivation fragment for a pair of transitions $\pi = (t_1, t_2)$ out of a universal configuration (q, ab), where $t_1 = (q, a) \to (q_1, a_1)$ and $t_2 = (q, a) \to (q_2, a_2)$.

4.2.5 Transitions Out of Universal Configurations. We finally present the lexicon entries and rules needed to simulate transitions out of universal configurations. This is the most involved part of our construction. To give an intuition, Figure 13 provides a bird's eye view of the fragment that our grammar derives for a universal configuration of the form $c = (q, ab)$ and a pair of transitions $\pi = (t_1, t_2)$, where $t_1 = (q, a) \to (q_1, a_1)$ and $t_2 = (q, a) \to (q_2, a_2)$. Recall that we may assume that every universal configuration has exactly two applicable transitions. On a high level, we construct two identical copies of the tape stack and then simulate the transition t_1 on one copy and the transition t_2 on the second copy. This is done in such a way that both t_1 and t_2 must lead to accepting configurations in order to terminate the derivation process. The actual derivation in the fragment proceeds in three phases as follows. First, it duplicates the tape stack of the root category $E(c)$. Second, it splits the duplicated stack into two identical halves, each targeted at one of the two transitions. Third, it simulates (in two separate branches) the two transitions on their respective halves to arrive at the two leaf categories $E(t_1(c))$ and $E(t_2(c))$. (Note that the fragment here differs from the one in Figure 12 in that it has *two* distinguished leaves, not one.) In the following we describe the three phases in detail.

Phase 1: Duplicating the Tape Stack. We illustrate this phase of the derivation in Figure 14. The derivation starts by checking whether the root category $E(c)$ indeed encodes a universal configuration, and records this fact by changing the target to the atomic category $[\pi; -]$ (28). The intended interpretation of this category is that the derivation is simulating the transition pair π but has not yet duplicated the tape stack ($-$). The derivation then nondeterministically extends the tape stack (29), in much the same way as for the initial configuration in Section 4.2.2. At the end of the phase, the derivation tests whether the two halves of the stack are equal, that is, whether the nondeterministic extension indeed created an exact copy of the initial stack. This test is crucial for our construction, and we describe it in more detail subsequently. If the test is successful, the

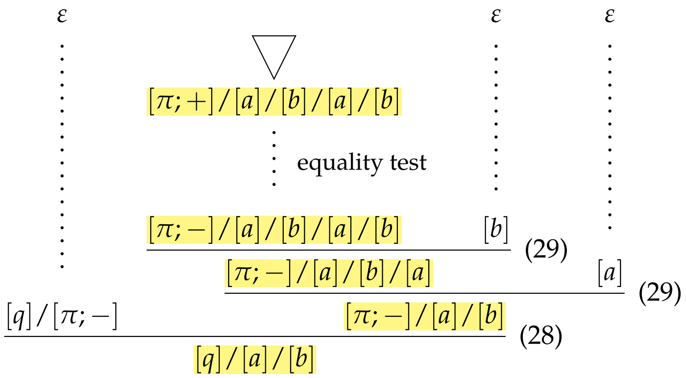

Figure 14
Simulation of transitions out of universal configurations. Phase 1: Duplicating the tape stack.

derivation changes the target to $[\pi; +]$, signifying that the derivation has successfully duplicated the tape stack.

To support derivations such as the one in Figure 14, we introduce the following lexicon entries and rules. Let $q \in Q$ be any universal state, and let $\pi = (t_1, t_2)$ be any pair of transitions where $t_1 = (q, a) \to (q_1, a_1)$ and $t_2 = (q, a) \to (q_2, a_2)$. Let also $b \in \Sigma$ be any tape symbol. We introduce the lexicon entry $\varepsilon := [q] / [\pi; -]$ and reuse the entries of the form $\varepsilon := [b]$ that we introduced in Section 4.2.2. The following rules implement the change of the target of $E(c)$ and nondeterministic extension of the tape stack:

$$[q]\,\$\,/\,[\pi; -] \quad [\pi; -]\,/\,X_1 \cdots /\,X_m \quad \Rightarrow \quad [q]\,\$\,/\,X_1 \cdots /\,X_m \tag{28}$$

$$[\pi; -]\,\$\,/\,[b] \quad [b] \quad \Rightarrow \quad [\pi; -]\,\$ \tag{29}$$

A rule of the form (28) is a composition rule of degree m that simultaneously restricts the target of its primary input to $[q]$ and the target of its secondary input to $[\pi; -]$. A rule of the form (29) is an application rule that restricts the target of its primary input to $[\pi; -]$.

It remains to describe how to implement the equality test. To give an intuition, Figure 15 shows a derivation that implements the test for the tape ab of our running example. For readability, we have numbered the arguments on the tape stack. Step (30) uses a composition rule of degree $2m$ to change the target of the root category to a new atomic category $[\pi; =_1]$. The intended interpretation of this category is that the derivation needs to check whether the two halves of the tape stack agree at position 1 and $m + 1$. Accordingly, the composition used in step (31) is restricted in such a way that it can only be instantiated if the two atomic categories at positions 1 and 3 are equal. Similarly, the composition used in step (32) can only be instantiated if the two categories at positions 2 and 4 are equal. It is not hard to see that this can be scaled up to m tests, each of which tests the equality of the categories at positions i and $m + i$. Taken together, these tests check whether the two halves of the tape are indeed identical.

More formally now, let $b \in \Sigma$ be any tape symbol. We introduce the following shorthand notation, where $1 \leq i \leq m$ is any tape position:

$$\eta_{i,b} \quad \equiv \quad /X_1 \cdots /X_{i-1}/[b]/X_{i+1} \cdots /X_m/X_{m+1} \cdots /X_{m+i-1}/[b]/X_{m+i+1} \cdots /X_{2m}$$

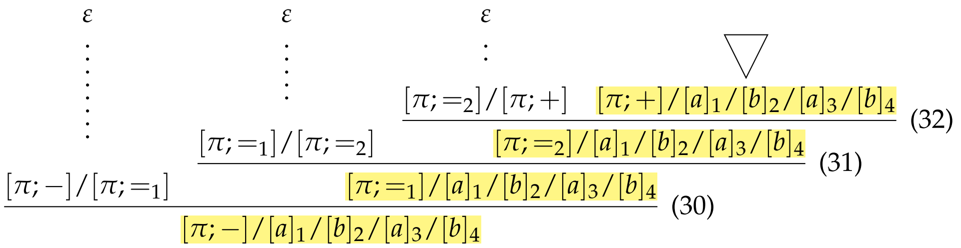
ε
ε
ε
$[\pi;=_2]\,/\,[\pi;+]$
$[\pi;+]\,/\,[a]_1\,/\,[b]_2\,/\,[a]_3\,/\,[b]_4$
(32)
$[\pi;=_1]\,/\,[\pi;=_2]$
$[\pi;=_2]\,/\,[a]_1\,/\,[b]_2\,/\,[a]_3\,/\,[b]_4$
(31)
$[\pi;-]\,/\,[\pi;=_1]$
$[\pi;=_1]\,/\,[a]_1\,/\,[b]_2\,/\,[a]_3\,/\,[b]_4$
(30)
$[\pi;-]\,/\,[a]_1\,/\,[b]_2\,/\,[a]_3\,/\,[b]_4$

Figure 15
Equality test for the tape *abab*.

Thus $\eta_{i,b}$ is a sequence of $2m$ slash–variable pairs, except that X_i and X_{m+i} have been replaced with the concrete atomic category $[b]$. Then to support derivations such as the one in Figure 15, we introduce the following lexicon entries, where $1 \leq i < m$:

$$\varepsilon := [\pi;-]\,/\,[\pi;=_1] \qquad \varepsilon := [\pi;=_i]\,/\,[\pi;=_{i+1}] \qquad \varepsilon := [\pi;=_m]\,/\,[\pi;+]$$

We also introduce the following composition rules of degree $2m$, for $1 \leq i < m$:

$$[\pi;-]\,\$\,/\,[\pi;=_1] \quad [\pi;=_1]\,/\,X_1 \cdots /\,X_{2m} \quad \Rightarrow \quad [\pi;-]\,\$\,/\,X_1 \cdots /\,X_{2m} \tag{30}$$

$$[\pi;=_i]\,\$\,/\,[\pi;=_{i+1}] \quad [\pi;=_{i+1}]\eta_{i,b} \quad \Rightarrow \quad [\pi;=_i]\,\$\,\eta_{i,b} \tag{31}$$

$$[\pi;=_m]\,\$\,/\,[\pi;+] \quad [\pi;+]\eta_{m,b} \quad \Rightarrow \quad [\pi;=_m]\,\$\,\eta_{m,b} \tag{32}$$

Phase 2: Splitting the Tape Stack. In the second phase, the derivation branches off into two subtrees, as was illustrated in Figure 13. We present this second phase in more detail in Figure 16. This derivation simulates the "splitting" of the tape stack into two (identical) halves. To implement it, we introduce lexicon entries $\varepsilon := [\pi;+]\,/\,[t_1]$ and $\varepsilon := [b]\,/\,[b]\,/\,[t_2]$, where $b \in \Sigma$ is any tape symbol. We also introduce the following rules:

$$[\pi;+]\,\$\,/\,[t_2] \quad [t_2]\,/\,X_1 \cdots /\,X_m \quad \Rightarrow \quad [\pi;+]\,\$\,/\,X_1 \cdots /\,X_m \tag{33}$$

$$[\pi;+]\,\$\,/\,[b] \quad [b]\,/\,[b]\,/\,[t_2] \quad \Rightarrow \quad [\pi;+]\,\$\,/\,[b]\,/\,[t_2] \tag{34}$$

$$[\pi;+]\,\$\,/\,[t_1] \quad [t_1]\,/\,X_1 \cdots /\,X_m \quad \Rightarrow \quad [\pi;+]\,\$\,/\,X_1 \cdots /\,X_m \tag{35}$$

Rule of the forms (33) and (35) are composition rules of degree m. Note that this ensures that the categories targeted at $[t_1]$ and $[t_2]$ encode a tape of m symbols. A rule of the form (34) is a composition rule of degree 2.

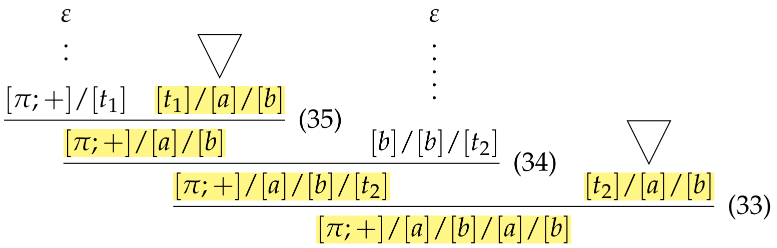

Figure 16
Simulation of transitions out of universal configurations. Phase 2: Splitting the tape stack.

Phase 3: Simulating the Two Transitions. In the third and final phase, the derivation simulates the two transitions t_1 and t_2. To implement this phase we do not need to introduce any new lexicon entries or rules; we can simply reuse part of the construction that we presented in Section 4.2.4 for the existential states. More specifically, we can reuse the part of that construction that starts with a category with target $[t]$ and uses rules (26) and (27).

Computational Complexity. All of the introduced lexical entries have size bounded by some constant independent of the input size. At the same time, all of the rules introduced in the three phases have degree bounded by $2m$. It is easy to see that we can construct each of these elements in time $\mathcal{O}(m)$. Furthermore, the number of lexical entries and rules produced in this step is bounded by a polynomial function of the input size. For instance, we add to G a number $|\Sigma| \cdot m$ of rules of types (30) and (31), because there is a single rule for each choice of a tape symbol a and index i with $1 \leq i < m$. Similar analyses can be carried out for the remaining elements. We then conclude that the overall contribution to $|G|$ in this step is polynomial in $|M|$ and $|w|$, and the construction can be carried out in the same amount of time.

4.2.6 Correctness. With all of the grammar components in place, we now address the correctness of our construction. We argue that the sentential derivations of G exactly correspond to the accepting computations of M when applied to the input string w. To do this, we read G's derivations in the canonical direction, that is, from the leaves to the root. First of all, observe that the fragments introduced in the various steps of the construction all use reserved target categories, and they all use rules with target restrictions for these categories. In this way, it is not possible in a derivation to mix fragments from different steps—that is, fragments cannot be broken apart.

Accepting Configurations. A (sentential) derivation in G starts with the fragments introduced in Section 4.2.3. Each of these fragments uses composition rules to combine several tape symbols into a category of the form $[\textbf{accept}]\alpha$, and then switches to a category of the form $[q]\alpha$ with $g(q) = \mathsf{A}$. Because of the degree restriction of rule (22), the switch is only possible if α has exactly m arguments; in all other cases the derivation will come to a dead end, that is, it will not derive the distinguished category of the grammar. The categories $[q]\alpha$ composed by the fragments of Section 4.2.3 encode accepting configurations of M, and it is not difficult to see that all possible accepting configurations with $g(q) = \mathsf{A}$ can be generated by these fragments. These are exactly the leaves of our valid computation trees.

Existential Configurations. The derivation freely attempts to apply the transitions of M to the categories obtained as above and, recursively, to all the categories that result from the application of these transitions. More precisely, a transition t applying to an existential configuration is simulated (in reverse) on a category $[q]\alpha$ using the fragment of Section 4.2.4. This is done using rule (27), which switches from a category with target $[q]$ to a category with target $[t]$, and produces a new category whose stack has $m+1$ arguments. At this point, only some rule of type (26) can be applied, resulting in the reduction of the stack, immediately followed by some rule of type (25), which is a composition rule of degree m. If the derivation were to use more than one occurrence of (26), then it would derive a category whose stack contains fewer than m elements. As a consequence, rule (25) would no longer be applicable, because of the restriction on the composition degree, and the whole derivation would come to a dead end.

Universal Configurations. The derivation can also simulate (in reverse) the two transitions t_1 and t_2 applying to a universal state q. This is done using the fragments of Section 4.2.5. In this case the derivation starts with rules (27) and (26) used for the existential states; but now the involved categories have targets $[t_i]$ disjoint from the targets used for the existential states, since transitions t_i apply to universal states. The simulation of t_1 and t_2 results in categories $[q]\alpha$ and $[q]\alpha'$ with the same target $[q]$ and with m arguments each. These categories are then merged into a new category $[q]\alpha\alpha'$ by concatenating their stacks, and an equality test is successively carried out on $\alpha\alpha'$. If the test is successful, the derivation pops an arbitrary number of arguments from $[q]\alpha\alpha'$, resulting in a new category of the form $[q]\alpha''$. Rule (28) can then be applied only in case $[q]\alpha''$ has exactly m arguments. This means that $[q]\alpha''$ encodes one of the configurations of M, and that $\alpha = \alpha' = \alpha''$.

Initial Configuration. Finally, if this process ever composes a category $[q_0]\alpha$ encoding the initial configuration of M relative to the input string w, then the derivation uses the fragment of Section 4.2.2. The rules of this fragment switch the target category from $[q_0]$ to $[\mathbf{init}]$, the distinguished category of G, and then pop the stack arguments, providing thus a sentential derivation for ε.

This correctness argument finally concludes the proof of our Lemma 4.

4.3 Membership in EXPTIME

It remains to prove the following:

Lemma 5
The universal recognition for unrestricted VW-CCG is in EXPTIME.

To show this, we extend an existing recognition algorithm by Kuhlmann and Satta (2014) that takes as input a VW-CCG G with no empty categories and a string w, and decides whether $w \in L(G)$.

Complexity of the Algorithm of Kuhlmann and Satta. The algorithm of Kuhlmann and Satta (2014) is based on a special decomposition of CCG derivations into elementary pieces, adapting an idea first presented by Vijay-Shanker and Weir (1990). These elementary pieces are specially designed to satisfy two useful properties: First, each elementary piece can be stored using an amount of space that does not depend on the length of w. Second, elementary pieces can be shared among different derivations of w under G. The algorithm then uses dynamic programming to construct and store in a multi-dimensional parsing table all possible elementary pieces pertaining to the derivations of w under G. From such a table one can directly check whether $w \in L(G)$. Despite the fact that the number of derivations of w under G can grow exponentially with the length of w, the two properties of elementary pieces allow the algorithm to run in time polynomial in the length of w. However, the runtime is not bounded by a polynomial function in the size of G, as should be expected from the hardness results reported in Section 3.1.

More specifically, let $\mathcal{A}$ be the set of all atomic categories of the input grammar G, and let $\mathcal{L}$ be the set of all arguments occurring in the categories in G's lexicon. Let also d be the maximum degree of a composition rule in G, let a be the maximum arity of an argument in $\mathcal{L}$, and let ℓ be the maximum number of arguments in the categories in G's lexicon. We set $c_G = \max\{d + a, \ell\}$. Kuhlmann and Satta (2014) report for their algorithm a running time in $\mathcal{O}(|\mathcal{A}| \cdot |\mathcal{L}|^{2c_G} \cdot |w|^6)$.

To see that this upper bound is an exponential function in the size of the input, observe that the quantities $|\mathcal{A}|$ and $|\mathcal{L}|$ are both bounded by $|G|$, because each category in $\mathcal{A}$ or in $\mathcal{L}$ must also occur in G. Furthermore, ℓ is bounded by the maximum length of a category in G's lexicon, and thus by $|G|$. Similarly, d is bounded by the length of some secondary component in a composition rule of G, and a is bounded by the length of some category in G's lexicon. Then $d + a$ is bounded by $|G|$ as well. Combining the previous observations, we have that c_G is bounded by $|G|$. We can then conclude that the runtime of the recognition algorithm is bounded by

$$|G| \cdot |G|^{2|G|} \cdot |w|^6 \;=\; |G|^{1+2|G|} \cdot |w|^6 \;=\; 2^{\log|G|+2|G|\log|G|} \cdot |w|^6 \tag{36}$$

which is indeed exponential in the size of the input.

Extension of the Algorithm of Kuhlmann and Satta to VW-CCG. As already mentioned, the algorithm by Kuhlmann and Satta (2014) works for a grammar G with no empty categories. More specifically, the algorithm starts by adding to the parsing table items of the form $[X, i, i+1]$ for each category X that is assigned by G's lexicon to the ith word in the input string w. Here $[X, i, i+1]$ represents an elementary piece of derivation consisting of a tree with two nodes: a root with label X and a child node with label the ith word of w. In order to extend the algorithm to unrestricted VW-CCG, all we need to do is add to the parsing table items of the form $[X, i, i]$ for every empty category X in G's lexicon and for every integer i with $0 \leq i \leq |w|$. This creates new elementary pieces of derivations accounting for the empty string. These pieces can be combined with each other, as well as with other pieces already existing in the table, triggering the construction of derivations that involve empty categories.

The proof of the correctness of the recognition algorithm by Kuhlmann and Satta (2014) immediately extends to the new algorithm. This is so because the proof only rests on structural properties of CCG derivations, without any assumption about the fact that these derivations involve words from w or the empty string. Furthermore, the exponential runtime reported here still holds for the new algorithm. This is a consequence of the fact that we use the same item representation as in the original algorithm for the elementary pieces of derivations involving the empty string.

Although the algorithms discussed here are designed for the random access machine architecture, or RAM for short, it is well known that any algorithm working on a RAM can be computed on a deterministic Turing machine with only polynomial-time overhead; see, for instance, Papadimitriou (1994, Theorem 2.5). We can thus conclude that the universal recognition problem for VW-CCG can still be solved in exponential time on a deterministic Turing machine.

4.4 Discussion

In view of our proof of Theorem 2, we now come back to the question raised in Section 3.3. Specifically, we want to further investigate the features that are responsible for the additional complexity that comes with unrestricted VW-CCG. In our reduction in Section 4.2 we have used two features of the formalism that were already discussed in Section 3.3, namely, the capability to define rules with restrictions on their secondary input categories, and the capability to define rules whose secondary input categories do not have any constant bound on their arity. In addition, we have also exploited two new features, listed subsequently. Again, dropping any one of these four features would break our proof (but see our discussion in Section 5.1).

Derivational Ambiguity. Our grammar makes crucial use of derivational ambiguity. More precisely, G encodes M's configurations relative to w into its own categories. We assign all of these categories to the empty string ε, thus introducing massive ambiguity in G's derivations. Our proof of Lemma 4 would not work if we restricted ourselves to the use of unambiguous grammars, and we note that the computational complexity of the universal recognition problem for the class VW-CCG restricted to unambiguous grammars is currently unknown. Furthermore, on a par with lexical ambiguity discussed in Section 3.3, syntactic ambiguity is an essential feature in most formalisms for the modeling of natural language syntax, including those whose universal recognition problem can be parsed in polynomial time, such as TAGs. As such, this individual feature cannot be held responsible, at least in isolation, for the complexity of the recognition problem for the class VW-CCG, and in designing a polynomially parsable version of VW-CCG we would not be interested in blocking derivational ambiguity.

Unlexicalized Rules. We remind the reader that, broadly speaking, a lexicalized rule in a string rewriting formalism is a rule that (directly) produces some lexical token. The rule is also thought to be specialized for that token, meaning that the rule contributes to the derivation by introducing some structure representing the syntactic frame and valencies of the token itself. The basic idea of our proof is to simulate valid computations of M on w through derivations of G. In particular, each fragment in a derivation of G uniquely represents some node of a valid computation. It is therefore essential in our proof that each fragment uses unlexicalized rules, that is, generates the empty string. One may view this phenomenon in the light of computational complexity. It is not difficult to verify that given an unlexicalized grammar G, it cannot be transformed into a lexicalized grammar G' in polynomial time unless NP = EXPTIME.[5] For assume that such a polynomial-time transformation T exists. This would imply that an arbitrary instance (G, w) of the universal recognition problem with ε-entries can be converted (in polynomial time) into an equivalent instance $(T(G), w)$ without ε-entries. We know that the former problem is EXPTIME-complete and the latter problem is NP-complete. Thus, the existence of T implies that NP = EXPTIME. One should observe that this is not an effect of $T(G)$ being prohibitively large: the size of the lexicalized grammar $T(G)$ is polynomially bounded in the size of G, because T is computable in polynomial time.

5. General Discussion

The computational effect of grammar structure and grammar size on the parsing problem is rather well understood for several formalisms currently used in computational linguistics, including context-free grammar and TAG. However, to the best of our knowledge, this problem has not been investigated before for VW-CCG or other versions of CCG; see, for instance, Kuhlmann and Satta (2014) for discussion. In this article we have shed some light on the impact of certain features of VW-CCG on the computational complexity of the parsing problem. We have shown that the universal

[5] This equality is considered unlikely to hold because it would, for instance, imply that NP = PSPACE, and that the polynomial hierarchy collapses.

recognition problem for VW-CCG is dramatically more complex than the corresponding problem for TAG, despite the already-mentioned weak equivalence between these two formalisms. Our results therefore solve an open problem for VW-CCG and raise important questions about the computational complexity of contemporary incarnations of CCG. In this section we would like to conclude the article with a discussion of our results in the broader context of current research.

5.1 Sources of Complexity

The two features of VW-CCG that are at the core of our complexity results are the rule restrictions and the unbounded degree of composition rules. As already mentioned, dropping any one of these two features would break our specific constructions. At the same time, it is important to consider the problem from the dual perspective: We do not know whether dropping *any combination* of these two features would admit a polynomial-time parsing algorithm for VW-CCG—and this holds true regardless of the grammars being ε-free or not. Perhaps most important for current practice, this means that we do not know whether modern versions of CCG can be parsed in polynomial time. To illustrate the point, the algorithm by Kuhlmann and Satta (2014) (Section 4.3) takes as input an ε-free VW-CCG G and a string w, and decides whether $w \in L(G)$. Even if G has no rule restrictions and the degree of its composition rules is considered as a constant, the upper bound that we would get from the analysis in Equation (36) would still be exponential in the grammar size. This shows that our understanding of the computational properties of CCG is still quite limited.

Epsilon Entries. One important issue that needs further discussion here is the role of ε-entries in VW-CCG. From the linguistic perspective, ε-entries violate one of the central principles of CCG, the Principle of Adjacency (Steedman 2000, page 54). From the computational perspective, ε-entries represent the boundary between the results in Section 3 and Section 4. However, because we do not know whether the classes NP and EXPTIME can be separated, we cannot draw any precise conclusion about the role of ε-entries in the parsing problem. Even under the generative perspective, we do not know the exact role of ε-entries. More precisely, the proof of the weak equivalence between VW-CCG and TAG provided by Vijay-Shanker and Weir (1994) makes crucial use of ε-entries, and it is currently unknown whether the generative capacity of VW-CCG without ε-entries is still the same as that of TAG, or if it is strictly smaller. This is another important open problem that attests our lack of theoretical understanding of CCG.

Unbounded Composition. A second issue that we would like to discuss is related to the notion of degree of composition rules in VW-CCG. According to the original definition of VW-CCG, each *individual* grammar in this class has a specific bound on the degree of its composition rules, but there is no constant bound holding for all grammars. As already discussed in Sections 3.3 and 4.4, the complexity results in this article do exploit this property in a crucial way. However, there are two alternative scenarios that we want to consider here. In a first scenario, one could state that there exists some language-independent constant that bounds the degree of composition rules *for all* grammars in the class VW-CCG. This would break all of the constructions in this article. The second possible scenario is one that has been discussed by, among others, Weir and Joshi (1988, Section 5.2) and Steedman (2000, page 210): We could define a formalism alternative to VW-CCG in which an individual grammar is allowed to use composition rules of unbounded degree. This would mean that the \$ notation introduced in Equation (8) must be used in the primary category as well as in the secondary category of a composition rule. Such a move would go in the opposite direction with respect to the first scenario above, reaching the power of Turing machines, as informally explained in what follows. Recall that in Section 4 we have used VW-CCG derivations to simulate moves of an ATM working with a circular tape whose size is bounded by some polynomial function of the length of the input. Specifically, we have encoded such a tape into some category X, and have used X as a primary or as a secondary input category in composition rules, in order to simulate the moves of the ATM. If we now allow the use of composition rules of arbitrarily large degree within an individual grammar, we can simulate the moves of a general Turing machine, in a way very similar to what we have done with our ATMs. This shows that the degree of composition rules can play a very powerful role in the definition of CCG formalisms.

A Note on Worst-Case Analysis. Our analysis of parsing complexity examines how the parser will perform in the least favorable situations. This perspective is justified by our interest in the question of where CCG sits within the landscape of mildly context-sensitive grammars, which are characterized by worst-case polynomial-time parsing (Joshi 1985). On the other hand, our results do not allow us to draw strong conclusions about practical average-case or expected parsing complexity, a question that many practitioners in the field may be more interested in when choosing a formalism for a problem. At the same time, recent progress on the development of practical CCG parsers has shown that with suitable heuristics, this formalism can be processed with very high efficiency (Lee, Lewis, and Zettlemoyer 2016; Lewis and Steedman 2014). We tend to view empirical and worst-case complexity as two orthogonal issues, where the latter enriches our understanding of the problem and might lead to the development of new, improved formalisms and algorithms, often with further advancements on the practical side.

5.2 Succinctness

As already mentioned, VW-CCG is known to be generatively equivalent to TAG, in the weak sense, as shown by Weir and Joshi (1988) and Vijay-Shanker and Weir (1994). Schabes (1990) reports that the universal recognition problem for TAG can be decided in time $\mathcal{O}(|G|^2|w|^6)$, where $|G|$ is the size of the input grammar G and $|w|$ is the length of the input sentence w. One could hope then to efficiently solve the universal recognition problem for VW-CCG by translating an input VW-CCG G into an equivalent TAG G', and then applying to G' and the input string any standard recognition method for the latter class. However, the part of the equivalence proof by Vijay-Shanker and Weir (1994) showing how to translate VW-CCG to TAG requires the instantiation of a number of elementary trees in G' that is exponential in $|G|$. (Trees are the elementary objects encoding the rules in a TAG.)

The fact that the same class of languages can be generated by grammar formalisms with substantially different parsing complexity naturally leads us to the notion of the succinctness of a grammar, cf. Hartmanis (1980). In formal language theory, grammar **succinctness** is used to measure the expressive capacity of a grammar formalism, as opposed to its generative capacity. More precisely, grammar succinctness measures the number of resources that different grammar formalisms put in place in order to generate the same language class. As a simple example, it is well known that certain finite languages can be generated by context-free grammars that are very compact, that is, small in size, while the same languages require finite state automata of size exponentially larger. In computational linguistics, succinctness was first discussed in the context of the formalism of immediate dominance/linear precedence constraint (ID/LP) grammar, a variant of context-free grammar where the ordering of the nonterminals in the right-hand side of a rule can be relaxed. Moshier and Rounds (1987) show that ID/LP grammars are exponentially more succinct than context-free grammars. As in the example above, this means that there are languages for which any context-free grammar must necessarily be at least super-polynomially larger than the smallest ID/LP grammar. A similar fact holds for VW-CCG: By our result in Section 3 there are languages for which there exist small VW-CCGs but where the weakly equivalent TAG must necessarily be at least exponentially larger (unless PTIME = NP). If we allow ε-entries, then Section 4 provides a stronger result: we can get rid of the qualification "unless PTIME = NP", as PTIME $\neq$ EXPTIME holds unconditionally (cf. Papadimitriou 1994, Theorem 7.1 and the subsequent corollary). Because we can also translate any TAG into an equivalent VW-CCG without blowing up the size of the grammar, following the construction by Vijay-Shanker and Weir (1994), we conclude that VW-CCG is more succinct than TAG. However, the price we have to pay for this gain in expressivity is the extra parsing complexity of VW-CCG.

5.3 The Landscape of Mildly Context-Sensitive Grammars

Finally, connecting back to the original motivation of this work that we gave in Section 1, we would like to conclude our discussion by placing our results for VW-CCG in the broader scenario of the class of mildly context-sensitive formalisms. This provides a more complete picture of this class than what we had before. The (informal) class of mildly context-sensitive grammar formalisms had originally been proposed by Joshi (1985) to provide adequate descriptions of the syntactic structure of natural language. This class includes formalisms whose generative power is only slightly more powerful than context-free grammars, that is, far below the one of context-sensitive grammars.

In Figure 17 we map several known mildly context-sensitive formalisms into a two-dimensional grid defined by the generative capacity of the formalism (horizontal axis) and the computational complexity of the universal recognition problem (vertical

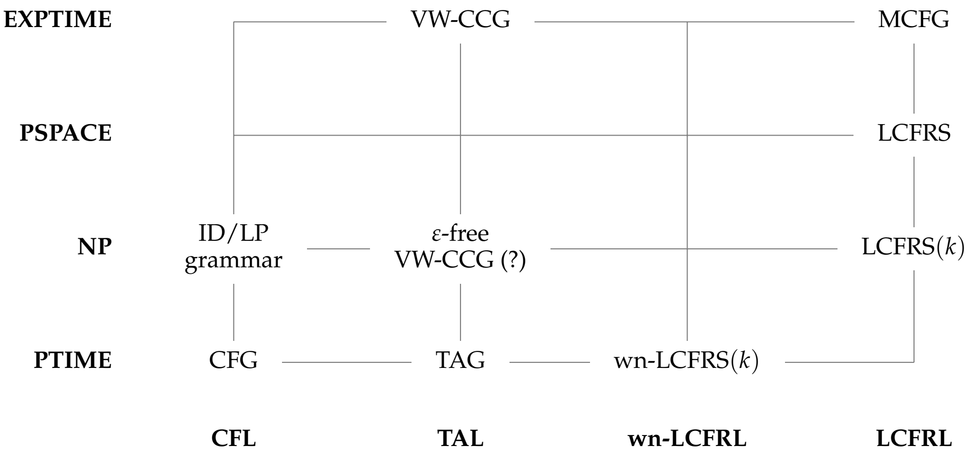

Figure 17
Weak generative capacity (horizontal axis) versus computational complexity (vertical axis) of various mildly context-sensitive grammar formalisms.

axis). For comparison, we also include in the picture some formalisms generating the context-free languages. We thus start at the leftmost column of the grid with the class of context-free grammar and the class of ID/LP grammar. As already mentioned, although these two classes are generatively equivalent, ID/LP grammar is more succinct than context-free grammar and, as a consequence, the two classes do not have the same computational complexity. On the next column to the right, we reach the generative power of tree-adjoining languages, which is strictly larger than that of context-free languages. Both TAG and VW-CCG are in this column but, by the results in this article, the computational complexity of VW-CCG is far above the complexity of TAG, again because of the increase in expressivity for the latter class. We have also tentatively placed ε-free VW-CCG in this column, although we do not know at this time whether the generative capacity of this class is the same as that of general VW-CCG, hence our question mark in the figure. In the next column to the right we find the class of well-nested linear context-free rewriting system with fan-out bounded by k, written wn-LCFRS(k). A rewriting system in wn-LCFRS(k) generates languages of string tuples, where the number of components in each tuple is bounded by a constant k called the fan-out of the system. The system exploits rules that work by combining tuple components in a way that satisfies the so-called *well-nestedness* condition, a generalization of the standard condition on balanced brackets. Although this class further extends the generative capacity of TAG (as a special case, the class wn-LCFRS(2) is generatively equivalent to TAG), it manages to keep the complexity of the universal recognition problem in PTIME, as shown by Gómez-Rodríguez, Kuhlmann, and Satta (2010). In the last column of our grid we have placed the class of linear context-free rewriting system (LCFRS) and the class of LCFRS with fan-out bounded by a constant k (LCFRS(k)), which have been originally defined by Vijay-Shanker, Weir, and Joshi (1987). Historically, LCFRS has been introduced before wn-LCFRS(k), and the latter class was investigated as a restricted version of LCFRS. In this column we also find the class of multiple context-free grammar (MCFG) defined by Seki et al. (1991), who also prove the generative equivalence result with LCFRS. The computational complexity results displayed in this column are from Kaji et al. (1992) and Satta (1992). All these systems generate string tuples but they do not satisfy the well-nestedness condition of wn-LCFRS(k). As a result, even in case of the class LCFRS(k), where the fan-out is bounded by a constant k, these systems cannot be parsed in polynomial time, unless PTIME $=$ NP, in contrast with the class wn-LCFRS(k).

Acknowledgments

We are grateful to the anonymous reviewers of the article for their detailed and constructive comments.

References

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

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

Chandra, Ashok K., Dexter C. Kozen, and Larry J. Stockmeyer. 1981. Alternation. *Journal of the Association for Computing Machinery*, 28(1):114–133.

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

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.

Gómez-Rodríguez, Carlos, Marco Kuhlmann, and Giorgio Satta. 2010. Efficient parsing of well-nested linear context-free rewriting systems. In

Proceedings of Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics, pages 276–284, Los Angeles.

Hartmanis, Juris. 1980. On the succinctness of different representations of languages. *SIAM Journal on Computing*, 9(1):114–120.

Hockenmaier, Julia and Mark Steedman. 2007. CCGbank. A corpus of CCG derivations and dependency structures extracted from the Penn Treebank. *Computational Linguistics*, 33:355–396.

Jeż, Arthur and Alexander Okhotin. 2011. Complexity of equations over sets of natural numbers. *Theory of Computing Systems*, 48(2):319–342.

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.

Kaji, Yuichi, Ryuichi Nakanishi, Hiroyuki Seki, and Tadao Kasami. 1992. The universal recognition problems for multiple context-free grammars and for linear context-free rewriting systems. *IEICE Transactions on Information and Systems*, E75-D(1):78–88.

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

Kuhlmann, Marco and Giorgio Satta. 2014. A new parsing algorithm for combinatory categorial grammar. *Transactions of the Association for Computational Linguistics*, 2(Oct):405–418.

Lee, Kenton, Mike Lewis, and Luke Zettlemoyer. 2016. Global neural CCG parsing with optimality guarantees. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2366–2376, Austin.

Lewis, Mike and Mark Steedman. 2013. Unsupervised induction of cross-lingual semantic relations. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 681–692, Seattle, WA.

Lewis, Mike and Mark Steedman. 2014. A* CCG parsing with a supertag-factored model. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, pages 990–1000, Doha.

Moshier, M. Drew and William C. Rounds. 1987. On the succinctness properties of unordered context-free grammars. In *Proceedings of the 25th Annual Meeting of the Association for Computational Linguistics*, pages 112–116, Stanford, CA.

Papadimitriou, Christos H. 1994. *Computational Complexity*. Addison-Wesley.

Rimell, Laura, Stephen Clark, and Mark Steedman. 2009. Unbounded dependency recovery for parser evaluation. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, pages 813–821, Singapore.

Ristad, Eric S. 1986. The computational complexity of current GPSG theory. In *Proceedings of the 24th Annual Meeting of the Association for Computational Linguistics*, pages 30–39, New York, NY.

Satta, Giorgio. 1992. Recognition of linear context-free rewriting systems. In *Proceedings of the 30th Annual Meeting of the Association for Computational Linguistics*, pages 89–95, Newark, DE.

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

Seki, Hiroyuki, Takashi Matsumura, Mamoru Fujii, and Tadao Kasami. 1991. On multiple context-free grammars. *Theoretical Computer Science*, 88(2):191–229.

Steedman, Mark. 2000. *The Syntactic Process*. 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, pages 181–224.

Vijay-Shanker, K. and Aravind K. Joshi. 1985. Some computational properties of tree adjoining grammars. In *Proceedings of the 23rd Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 82–93, Chicago, IL.

Vijay-Shanker, K. 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, PA.

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. 1987. Characterizing structural descriptions produced by various grammatical formalisms. In *Proceedings of the 25th Annual Meeting of the Association for Computational Linguistics*, pages 104–111, Stanford, CA.

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*, pages 278–285, Buffalo, NY.

White, Michael, Robert A. J. Clark, and Johanna D. Moore. 2010. Generating tailored, comparative descriptions with contextually appropriate intonation. *Computational Linguistics*, 36(2):159–201.

Zhang, Yue and Stephen Clark. 2011. Shift-reduce CCG parsing. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics*, pages 683–692, Portland, OR.

Zhang, Yue and Stephen Clark. 2015. Discriminative syntax-based word ordering for text generation. *Computational Linguistics*, 41(3):503–538.