Polymorphic Categorial Grammars: expressivity and computational properties

Matteo Capelletti
matteo.capelletti@gmail.com

Fabio Tamburini
DSLO, University of Bologna - Italy
fabio.tamburini@unibo.it

Abstract

We investigate the use of polymorphic categorial grammars as a model for parsing natural language. We will show that, despite the undecidability of the general model, a subclass of polymorphic categorial grammars, which we call *linear*, is mildly context-sensitive and we propose a polynomial parsing algorithm for them. An interesting aspect of the resulting system is the absence of spurious ambiguity.

1 Introduction

The simplest model of a categorial grammar is based on the so called Ajdukiewicz–Bar-Hillel calculus of Ajdukiewicz [1935] and Bar-Hillel [1953], with only elimination rules for the slashes. Contemporary categorial grammars in the style of Ajdukiewicz–Bar-Hillel grammars are called *combinatory categorial grammars*, see Steedman [2000]. Such systems adopt other forms of composition rules which enable them to generate non–context-free languages, see Weir and Joshi [1988]; Vijay-Shanker and Weir [1994]. The other main tradition of categorial grammar, the type-logical grammars of Morrill [1994]; Moortgat [1997], stemming from the work of Lambek [1958], adopt special kinds of structural rules, that enable the system to generate non–context-free languages. Both approaches increase the generative power of the basic system by adding special kinds of rules.

Here we adopt a different strategy, which consists in keeping the elementary rule component of AB grammars and in introducing *polymorphic* categories, that is syntactic categories containing atomic variables ranging over categories. The inference process will be driven by unification, rather than by simple identity of formulas. We will see two kinds of polymorphic categorial grammars, one that is Turing complete and another, resulting from a restriction on the first, which is mildly context-sensitive. This second system, which is obviously the most interesting one for linguistics, has some important advantages with respect to the aforementioned categorial settings. In respect to TLG, the polymorphic system we define is *polynomial*, as we will prove by providing a parsing algorithm. In respect to CCG, our system is not affected by the so called *spurious ambiguity* problem, that is the problem of generating multiple, semantically equivalent, derivations.

The deductive system given in Figure 1, which we call $\mathrm{AB}^{\otimes}$, is a simple modification of the calculus of Kandulski [1988] to which it can easily be proved equivalent.

Identity Axioms:	$A \to A$
Product Axioms:	$A, B \to A \otimes B$
Cut Rule:	$\dfrac{\Gamma \to A \quad \Delta[A] \to C}{\Delta[\Gamma] \to C}\ (C)$
Shifting Rules:	$\dfrac{\Gamma \to C/A}{\Gamma, A \to C}\ (S_1) \qquad \dfrac{\Gamma \to A\backslash C}{A, \Gamma \to C}\ (S_2)$

Figure 1: Ajdukiewicz–Bar-Hillel calculus with product, $\mathrm{AB}^{\otimes}$.

This basic CG models can be extended to generate non context-free languages in at least two ways. The first uses structural rules, introduction rules and other types of composition schemes. These approaches are characteristic of TLG, see Morrill [1994]; Moortgat [1997]; Moot [2002], and CCG, see Steedman [2000];

Baldridge [2002], and have been widely explored in the past. The second is based on the introduction of *polymorphism*. Here, we study this second approach.

The formalism of polymorphic categorial grammar that we are going to present is inspired by the polymorphic theory of types, see Girard et al. [1989]; Barendregt [1992]. Types may contain type variables together with constants, and these variables may be (implicitly or explicitly) quantified over. The idea of polymorphism is very simple and natural. Rather than defining a class of `id` functions $\mathtt{id}_{Int} :: Int \to Int$, $\mathtt{id}_{Char} :: Char \to Char$ and so forth, the function `id` is defined for any type α, as $\mathtt{id} :: \forall\alpha.\alpha \to \alpha$ or $\mathtt{id} :: \alpha \to \alpha$ where α is implicitly universally quantified.

The same idea is very natural also in linguistics, where, for example, coordination particles such as 'and' and 'or' are typically polymorphic, as they coordinate expressions of *almost* any syntactic category. Thus one can find in the categorial grammar literature several examples of polymorphic assignments for these expressions Lambek [1958]; Steedman [1985]; Emms [1993]; Clark and Curran [2007].

Another example of Ajdukiewicz–Bar-Hillel style categorial grammars adopting a form of polymorphism are the unification categorial grammars Uszkoreit [1986]; Zeevat [1988]; Heylen [1999], where polymorphism is used at the level of feature structures.

1.1 Unification Ajdukiewicz–Bar-Hillel grammars

Syntactic categories of $\mathrm{UAB}^{\otimes}$ are defined as follows.

$$\begin{array}{lll}
\textbf{Atoms:} & \mathcal{A} & ::= \; a, b, c, n, s, i \ldots \\
\textbf{Variables:} & \mathcal{V} & ::= \; \alpha, \beta, \gamma \ldots \\
\textbf{Categories:} & \mathcal{F} & ::= \; \mathcal{A} \mid \mathcal{V} \mid \mathcal{F} \otimes \mathcal{F} \mid \mathcal{F} \backslash \mathcal{F} \mid \mathcal{F}/\mathcal{F}
\end{array}$$

Unification of two categories A and B is defined in the obvious way and and the resulting substitution is denoted $A \approx B$.[1] The unification Ajdukiewicz–Bar-Hillel calculus, $\mathrm{UAB}^{\otimes}$ is defined in Figure 2.[2]

$$\begin{array}{ll}
\textbf{Identity Axioms}: & A \to A \\
\textbf{Product Axioms}: & A,\, B \to A \otimes B \\
\textbf{Cut Rule}: & \dfrac{\Gamma \to A \quad \Delta[B] \to C}{\Delta[\Gamma] \to C(A \approx B)}\ (C') \\
\textbf{Shifting Rules}: & \dfrac{\Gamma \to C/A}{\Gamma, A \to C}\ (S_1) \qquad \dfrac{\Gamma \to A\backslash C}{A, \Gamma \to C}\ (S_2)
\end{array}$$

Figure 2: Unification Ajdukiewicz–Bar-Hillel calculus, $\mathrm{UAB}^{\otimes}$

We give here some examples of non context-free languages generated by $\mathrm{UAB}^{\otimes}$ grammars.

Example 1 *We define the* $UAB^{\otimes}$ *grammar for the language* $a^n b^n c^n$, $n \geqslant 1$. *Let grammar* G_1 *consist of the following assignments:*

$$\begin{array}{ll}
a :: s/(b \otimes c) & b :: b \\
a :: (s/\alpha)\backslash(s/(b \otimes (\alpha \otimes c))) & c :: c
\end{array}$$

We derive the string 'aabbcc'. We write A *for the formula* $(s/\alpha)\backslash(s/(b\otimes(\alpha\otimes c)))$. *For readability, boxes are drawn around the words that anchor the axioms to the lexicon.*

$$\dfrac{\dfrac{\dfrac{\dfrac{\boxed{a}}{s/(b\otimes c) \to s/(b\otimes c)} \quad \dfrac{\dfrac{\boxed{a}}{A \to A}}{s/\alpha, A \to s/(b\otimes(\alpha\otimes c))}}{s/(b\otimes c), A \to s/(b\otimes((b\otimes c)\otimes c))}}{s/(b\otimes c), A, (b\otimes((b\otimes c)\otimes c)) \to s} \quad \dfrac{\dfrac{\boxed{b}}{b \to b} \quad \dfrac{\dfrac{\dfrac{\boxed{b}}{b \to b} \quad \dfrac{\boxed{c}}{c \to c}}{b, c \to b\otimes c} \quad \dfrac{\boxed{c}}{c \to c}}{b, c, c \to (b\otimes c)\otimes c}}{b, b, c, c \to b\otimes((b\otimes c)\otimes c)}}{s/(b\otimes c), A, b, b, c, c \to s}$$

Example 2 *We define a* $UAB^{\otimes}$ *grammar for 'ww',* $w \in \{a, b\}^{+}$.

[1] We use postfix notation for application of a substitution to a formula.

[2] Obviously, the rules involving unification are only defined if unification is defined.

Let grammar G_2 consist of the following assignments:

$$\begin{array}{ll} a :: a & b :: b \\ a :: s/a & b :: s/b \\ a :: (s/\alpha)\backslash(s/(\alpha \otimes a)) & b :: (s/\alpha)\backslash(s/(\alpha \otimes b)) \end{array}$$

It is easy to see that grammar G_2 generates exactly the language 'ww' with $w \in \{a, b\}^+$. As in the case of G_1, type variables are used as accumulators for long-distance dependencies.

A typical example of non context-freeness of natural language are the so called cross serial dependencies, which can be found, for instance, in Dutch subordinate clauses.

Example 3 *We define a UAB$^\otimes$ grammar for Dutch cross-serial dependencies. An example is the following subordinate clause, from Steedman [2000]:*

Ik	*Cecilia*	*Henk*	*de*	*nijlpaarden*	*zag*	*helpen*	*voeren.*
I	Cecilia	Henk	the	hippopotamuses	saw	help	feed.

I saw Cecilia help Henk feed the hippopotamuses.

These constructs exhibit dependencies of the form 'ww', where the ith words in the two halves are matched. An example lexicon generating the sentence in this example is the following[3]*:*

$$\begin{array}{ll} \textit{Ik, Cecilia, Henk, de nijlpaarden} & :: n \\ \textit{zag} & :: ((n \otimes (n \otimes \alpha))\backslash c)/(\alpha\backslash i) \\ \textit{helpen} & :: ((n \otimes \alpha)\backslash i)/(\alpha\backslash i) \\ \textit{voeren} & :: n\backslash i \end{array}$$

$$\dfrac{\begin{array}{c}\boxed{N}\\ n,(n,(n,n)) \to n \otimes (n \otimes (n \otimes n))\end{array} \qquad \dfrac{\dfrac{\dfrac{\begin{array}{c}\boxed{zag}\\ Z \to Z\end{array}}{Z, \alpha\backslash i \to (n \otimes (n \otimes \alpha))\backslash c} \qquad \dfrac{\dfrac{\begin{array}{c}\boxed{helpen}\\ H \to H\end{array}}{H, \alpha\backslash i \to (n \otimes \alpha)\backslash i} \quad \begin{array}{c}\boxed{voeren}\\ n\backslash i \to n\backslash i\end{array}}{H, n\backslash i \to (n \otimes n)\backslash i}}{Z,(H, n\backslash i) \to (n \otimes (n \otimes (n \otimes n)))\backslash c}}{(n \otimes (n \otimes (n \otimes n))),(Z,(H,n\backslash i)) \to c}}{(n,(n,(n,n))),(Z,(H,n\backslash i)) \to c}$$

These examples show that the languages generated by UAB$^\otimes$ grammars properly include the context-free languages (since AB$^\otimes$ grammars are properly included in UAB$^\otimes$ grammars). It is also easy to show that if we allow null assignments, that is assignments of the form $\epsilon :: A$, where ϵ is the empty string, the UAB$^\otimes$ formalism becomes undecidable[4].

1.2 Constraining UAB$^\otimes$ grammars

One constraint that we can impose on UAB$^\otimes$ grammars to avoid undecidability is *linearity*. Roughly, we impose the restriction that any lexical type may contain at most one variable, occurring once in an argument position and once in value position. Thus, for instance, α/α, $(s/\alpha)\backslash(s/(\alpha \otimes a))$ are licit types, while $(\alpha\backslash\alpha)/\alpha$, $(s/(\alpha \otimes \beta))\backslash(s/((\alpha \otimes \beta) \otimes a))$ and $(s/(\alpha \otimes \alpha))\backslash(s/((\alpha \otimes \alpha) \otimes a))$ are not. More precisely we define *linear* categories as the types F_2 generated by the following context-free grammar.

$$\begin{array}{lll} \sharp & ::= & \otimes \mid / \mid \backslash \\ F_0 & ::= & A \mid F_0 \sharp F_0 \\ F_2 & ::= & F_1 \sharp' F_1 \mid F_0 \mid F_2 \sharp' F_0 \mid F_0 \sharp' F_2 \end{array} \qquad \begin{array}{lll} \sharp' & ::= & / \mid \backslash \\ F_1 & ::= & F_1 \sharp F_0 \mid F_0 \sharp F_1 \mid \alpha \\ & & \end{array} \qquad (1)$$

The interesting case in this definition are the F_2 formulas of the form A/B or $B\backslash A$, with A and B in F_1, the others being meant essentially to put these in context. Consider the case of A/B, then α occurs exactly once in A and in B, since a F_1 category contains the variable α by construction. By analogy with lambda terms,

[3] In the deduction, we write Z for the type of 'zag', H for that of 'helpen' and N for the string 'Ik, Cecilia, Henk, de nijlpaarden'.

[4] One can easily adapt the construction of Johnson [1988] for proving the undecidability unification cased phrase-structure grammar formalisms, see Capelletti and Tamburini [2009a].

we can think of the occurrence of α in B as a *binder* (possibly a pattern-binder), and of the occurrence in A as the *bound* variable.

An UAB$^{\otimes}$ grammar is linear if all its lexical assignments are linear. Furthermore, in linear UAB$^{\otimes}$ grammar, we work by simple *variable instantiation*, rather than by a full-fledged unification algorithm. More precisely let us denote A^B a formula A with a distinguished occurrence of a subformula B. A^C is the formula obtained from A^B by replacing the occurrence of the subformula B with the formula C. The linear UAB$^{\otimes}$ calculus consists of all the rules of the UAB$^{\otimes}$ calculus in Figure 2 replacing the Cut rules with the following *instantiation* rule.

$$\frac{\Gamma \to A^B \quad \Delta[A^{\alpha}] \to C}{\Delta[\Gamma] \to C[\alpha := B]} \tag{2}$$

Observe that given a linear UAB$^{\otimes}$ grammar adopting the rule in 2, only linear types can occur in any of its deductions.

Observe also that the UAB$^{\otimes}$ grammars for $\mathrm{a}^n\mathrm{b}^n\mathrm{c}^n$ and ww languages as well as that for the Dutch cross serial patterns, are all linear. On the other hand, no linear UAB$^{\otimes}$ grammar can be given for the so called MIX or Bach language that is the language of the strings containing an equal number of a's, b's and c's[5].

As we have the proper inclusion of context-free languages and the realization of limited cross-serial dependencies, in order to have a *mildly context-sensitive* grammar formalism we shall prove that linear UAB$^{\otimes}$ grammars can be parsed in polynomial time. We do this in the next section by providing a parsing algorithm for linear UAB$^{\otimes}$ grammars.

2 Polynomial parsing with linear UAB$^{\otimes}$ grammars

Linear UAB$^{\otimes}$ grammars can be parsed in polynomial time by means of a simple extension of parsing algorithm for AB$^{\otimes}$ grammars given in Capelletti [2009], see Appendix A. Attention has to be paid to the way we implement the completion rules based on the cut rules in 2. Clearly the direct instantiation and substitution of the variable in the conclusion sequent will give an exponential growth of the number of items generated (in a similar way is it would happen by implementing naively the CCG composition rules, see Vijay-Shanker and Weir [1990]). Therefore we make use of an extra table to keep track of partial variable instantiations and postpone substitution as far as possible. This table, which we call *instantiation table*, is used for storing the 'partial' instantiations of variables. Let n be the length of the input string and Lex the input lexicon. Cells of the instantiation table are denoted $t_{(i,k,j)}$, where $0 \leqslant i < j \leqslant n$ and $0 \leqslant k \leqslant |Lex|$. We extend the construction of formulas with two kinds of variables, α_k and $\alpha_{(i,k,j)}$ where i, k and j are as before. The difference between the two kinds of variables is that α_k is an uninstantiated variable while $\alpha_{(i,k,j)}$ is a variable α_k which has been instantiated when an item $(i, \Lambda \to C, j)$ was generated, by the new instantiation rule given below. The algorithm assumes that different lexical entries contain different variables, that is for no k the variable α_k occurs in two distinct lexical assignments. The algorithm uses the following two new rules (of which we give only one oriented variant) together with those given for parsing with $\mathcal{AB}^{\otimes}$ in Appendix A.

$$\begin{array}{ll}
\text{Given items} & (i, \Delta \triangleright A^{\alpha_l}\, \Gamma \to C, k) \text{ and } (k, \Lambda \to A^B, j), \\
\text{generate the item} & (i, \Delta\, A^{\alpha_l} \triangleright \Gamma \to C[\alpha_l := \alpha_{(i,l,j)}], j) \\
\text{update the table} & t_{(i,l,j)} = t_{(i,l,j)} \cup \{B\} \\
& \\
\text{Given item} & (i, \Delta \triangleright \alpha_{(k,l,m)}\, \Gamma \to C, j) \text{ and } A \in t_{(k,l,m)} \\
\text{generate item} & (j, \triangleright A \to \alpha_{(k,l,m)}, j)
\end{array} \tag{3}$$

In Capelletti and Tamburini [2009a], we presented the detailed implementation of the parsing algorithm and proved its correctness. The complexity of the implementation given there is $O(|Lex|^2|\Sigma|^2 n^7)$, where $|Lex|$ and $|\Sigma|$ are the sizes of the lexicon and of the set of subformulas of the lexicon, respectively.

[5]To see this, we observe that the context-free language of the strings containing an equal number of a's and b's is not *linear*, in the sense of Hopcroft and Ullman [1979], see Linz [1990]. Hence for the MIX language, a UAB$^{\otimes}$ grammar needs to bind two distinct variables for each symbol, what violates linearity.

3 Conclusion

We have investigated some linguistic and computational properties of unification based categorial grammars. We have seen that, like other unification based grammar formalisms, unrestricted UAB$^{\otimes}$ grammars are Turing complete. However, we have also seen that the constraint of *linearity* locates the system among the mildly context-sensitive formalisms. A pleasant aspect of the resulting system, particularly with respect to others CG-based mildly context-sensitive categorial formalisms is the absence of spurious ambiguity. This is a pleasant property that results from the simple non-associative composition schemes adopted in the *parsing* system (see Appendix A and Capelletti and Tamburini [2009b]), and not from special constraints imposed to the derivations, as in Eisner [1996].

We conclude by observing that the linearity constraint can also be relaxed. For instance, while preserving the condition that only one variable occurs in a formula, we can admit more than two occurrences of this variable. In this way, we can include the standard types for coordination, $(\alpha\backslash\alpha)/\alpha$.

This condition enlarges the class of generated languages, producing for instance w^i or $\mathrm{a}_1^i\mathrm{a}_2^i\ldots\mathrm{a}_n^i$. To what extent and with what consequences from the computational point of view, is an open subject of investigation.

A Parser

Let an AB$^{\otimes}$ grammar G and a string $w_1\ldots w_n$ be given. The $\mathcal{AB}^{\otimes}_{\mathrm{Mix}}$ deductive parser is the triple $(\mathcal{I},\mathcal{A},\mathcal{R})$ presented in Figure 3. See Capelletti and Tamburini [2009a] for the $O(|Lex|^2|\Sigma|^2n^7)$ implementation of this parsing algorithm.

$$\mathcal{I} = \begin{array}{c} \{\ (i,\Gamma\triangleright\Delta\to A,j)\mid\Gamma\Delta\to A\in\mathrm{AB}^{\otimes},\ 0\leqslant i\leqslant j\leqslant n\ \} \\ \cup \\ \{\ (i,\Gamma\triangleleft\Delta\to A,j)\mid\Gamma\Delta\to A\in\mathrm{AB}^{\otimes},\ 0\leqslant i\leqslant j\leqslant n\ \} \end{array}$$

$$\mathcal{A} = \{\ (i-1,A\to A,i)\mid w_i :: A\in Lex\ \}$$

$$\mathcal{R} = \left\{ \begin{array}{ll}
\left.\begin{array}{l}
\dfrac{(i,\Delta\triangleright A\,\Gamma\to C,\,j)}{(i,\Delta\,A\triangleright\Gamma\to C,\,j)}\ \epsilon\Rightarrow^{+}A \\[2ex]
\dfrac{(i,\Gamma\,A\triangleleft\Delta\to C,\,j)}{(i,\Gamma\triangleleft A\,\Delta\to C,\,j)}\ \epsilon\Rightarrow^{+}A
\end{array}\right\} & \textit{ϵ-Scanning} \\[4ex]
\left.\dfrac{(i,\Delta\to C/B,\,j)}{(i,\Delta\triangleright B\to C,\,j)}\quad\dfrac{(i,\Delta\to B\backslash C,\,j)}{(i,B\triangleleft\Delta\to C,\,j)}\right\} & \textit{Shifting} \\[3ex]
\left.\dfrac{(i,\Delta\triangleright A\otimes B\,\Gamma\to C,\,j)}{(j,\triangleright A\,B\ \to A\otimes B,\,j)}\quad\dfrac{(i,\Gamma\,A\otimes B\triangleleft\Delta\to C,\,j)}{(i,\ A\,B\triangleleft\to A\otimes B,\,i)}\right\} & \textit{$\otimes$-Prediction} \\[3ex]
\left.\begin{array}{l}
\dfrac{(i,\Delta\triangleright A\,\Gamma\to C,k)\quad(k,\Lambda\to A,j)}{(i,\Delta\,A\triangleright\Gamma\to C,j)} \\[2ex]
\dfrac{(i,\Lambda\to A,k)\quad(k,\Delta\,A\triangleleft\Gamma\to C,j)}{(i,\Delta\ \triangleleft A\,\Gamma\to C,j)}
\end{array}\right\} & \textit{Completion}
\end{array}\right.$$

Figure 3: The system $\mathcal{AB}^{\otimes}_{\mathrm{Mix}}$.

References

Ajdukiewicz, K. (1935). Die syntaktische Konnexität. *Studia Philosophica*, 1:1–27.

Baldridge, J. (2002). *Lexically Specified Derivational Control in Combinatory Categorial Grammar*. PhD thesis, University of Edinburgh.

Bar-Hillel, Y. (1953). A quasi-arithmetical notation for syntactic description. *Language*, 29:47–58.

Barendregt, H. (1992). Lambda calculus with types. In Abramsky, S., Gabbay, D. M., and Maibaum, T. S. E., editors, *Handbook of Logic in Computer Science*, volume 2, pages 117–309. Oxford University Press.

Capelletti, M. (2009). Parsing with non-associative Lambek grammars. In Penn, G. and Fowler, T., editors, *Parsing with Categorial Grammars*. Esslli proceedings.

Capelletti, M. and Tamburini, F. (2009a). Parsing with polymorphic categorial grammars. *Advances in Computational Linguistics. Research in Computing Science*, 41:87–98.

Capelletti, M. and Tamburini, F. (2009b). Parsing with three models of categorial grammar. Under review: Computational Linguistics.

Clark, S. and Curran, J. R. (2007). Wide-coverage efficient statistical parsing with ccg and log-linear models. *Computational Linguistic*, 33(4):493–552.

Eisner, J. (1996). Efficient normal-form parsing for combinatory categorial grammar. In *Proceedings of the 34th annual meeting on Association for Computational Linguistics*, pages 79–86, Morristown, NJ, USA. Association for Computational Linguistics.

Emms, M. (1993). Parsing with polymorphism. In *EACL*, pages 120–129.

Girard, J.-Y., Taylor, P., and Lafont, Y. (1989). *Proofs and Types*. Cambridge University Press.

Heylen, D. (1999). *Types and Sorts. Resource logic for feature checking*. PhD thesis, UiL-OTS, Utrecht.

Hopcroft, J. E. and Ullman, J. D. (1979). *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley.

Johnson, M. (1988). *Attribute-Value Logic and the Theory of Grammar*, volume 16 of *CSLI Lecture Notes*. CSLI, Stanford, California.

Kandulski, M. (1988). The equivalence of nonassociative Lambek categorial grammars and context-free grammars. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 34:41–52.

Lambek, J. (1958). The mathematic of sentence structure. *American Mathematical Monthly*, 65(3):154–170.

Linz, P. (1990). *An introduction to formal languages and automata*. D. C. Heath and Company, Lexington, MA, USA.

Moortgat, M. (1997). Categorial type logics. In van Benthem, J. and ter Meulen, A., editors, *Handbook of Logic and Language*, pages 93–177. Elsevier, Amsterdam.

Moot, R. (2002). *Proof Nets for Linguistic Analysis*. PhD thesis, UiL-OTS, Utrecht.

Morrill, G. (1994). *Type Logical Grammar: Categorial Logic of Signs*. Kluwer, Dordrecht.

Steedman, M. (1985). Dependency and coordination in the grammar of dutch and english. *Language*, 61(3):523–568.

Steedman, M. (2000). *The Syntactic Process*. The MIT Press.

Uszkoreit, H. (1986). Categorial unification grammars. In *COLING*, pages 187–194.

Vijay-Shanker, K. and Weir, D. J. (1990). Polynomial time parsing of combinatory categorial grammars. In *ACL*, pages 1–8.

Vijay-Shanker, K. and Weir, D. J. (1994). The equivalence of four extensions of context-free grammars. *Mathematical Systems Theory*, 27.

Weir, D. J. and Joshi, A. K. (1988). Combinatory categorial grammars: Generative power and relationship to linear context-free rewriting systems. In *Meeting of the Association for Computational Linguistics*, pages 278–285.

Zeevat, H. (1988). Combining categorial grammar and unification. In Reyle, U. and Rohrer, C., editors, *Natural Language Parsing and Linguistic Theories*, pages 202–229. D. Reidel, Dordrecht.