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 AB\(^\oplus\), is a simple modification of the calculus of Kandulsiki [1988] to which it can easily be proved equivalent.

<table>
<thead>
<tr>
<th>Identity Axioms:</th>
<th>( A \rightarrow A )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Product Axioms:</td>
<td>( A, B \rightarrow A \otimes B )</td>
</tr>
<tr>
<td></td>
<td>( \Gamma \rightarrow A \quad \Delta[A] \rightarrow C )</td>
</tr>
<tr>
<td></td>
<td>( \Delta[\Gamma] \rightarrow C ) (C)</td>
</tr>
<tr>
<td>Cut Rule:</td>
<td>( \Gamma \rightarrow C/A ) (S(_1))</td>
</tr>
<tr>
<td></td>
<td>( \Gamma, A \rightarrow C ) (S(_2))</td>
</tr>
</tbody>
</table>

Figure 1: Ajdukiewicz–Bar-Hillel calculus with product, AB\(^\oplus\).

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];
Example 2

We define a UAB polymorphism is very simple and natural. Rather than defining a class of id functions \( \text{id} \) functions \( \text{id}_{\text{Int}} \colon \text{Int} \rightarrow \text{Int} \), \( \text{id}_{\text{Char}} \colon \text{Char} \rightarrow \text{Char} \), and so forth, the function \( \text{id} \) is defined for any type \( \alpha \), as \( \text{id} \colon \forall \alpha. \alpha \rightarrow \alpha \) or \( \text{id} \colon \alpha \rightarrow \alpha \) where \( \alpha \) 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 Lambeek [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 UAB\(^\circ\) are defined as follows.

\[
\begin{align*}
\text{Atoms:} & \quad A := a, b, c, n, s, i, \ldots \\
\text{Variables:} & \quad \forall := \alpha, \beta, \gamma, \ldots \\
\text{Categories:} & \quad \mathcal{F} := A \mid \forall \mid \mathcal{F} \otimes \mathcal{F} \mid \mathcal{F} \setminus \mathcal{F} \mid \mathcal{F}/\mathcal{F}
\end{align*}
\]

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, UAB\(^\circ\) is defined in Figure 2.\(^2\)

![Figure 2: Unification Ajdukiewicz–Bar-Hillel calculus, UAB\(^\circ\)](image)

We give here some examples of non context-free languages generated by UAB\(^\circ\) grammars.

**Example 1** We define the UAB\(^\circ\) grammar for the language \( a^n b^n c^n \), \( n \geq 1 \). Let grammar \( G_1 \) consist of the following assignments:

\[
\begin{align*}
A & \colon= b \cdot c \\
B & \colon= b \\
C & \colon= (b / c) \cdot (c / (b \cdot (\alpha \cdot c))) \\
\end{align*}
\]

We derive the string ‘aabbc’. We write \( A \) for the formula \( (s/\alpha) \cdot (s / (b \cdot (\alpha \cdot c))) \). For readability, boxes are drawn around the words that anchor the axioms to the lexicon.

\[
\begin{align*}
\text{Example 2} & \quad \text{We define a UAB}^\circ \text{ grammar for ‘ww’, } w \in \{a, b\}^+.
\end{align*}
\]

---

\(^1\)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:

\[
\begin{align*}
\alpha & \::= a \\
\beta & \::= b \\
\gamma & \::= (s/\alpha)\backslash(s/(\alpha \otimes a)) \\
\delta & \::= (s/\alpha)\backslash(s/(\alpha \otimes b)) \\
\end{align*}
\]

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\textsuperscript{c} grammar for Dutch cross-serial dependencies. An example is the following subordinate clause, from Steedman [2000]:

\begin{quote}
I saw Cecilia help Henk feed the hippopotamuses.
\end{quote}

These constructs exhibit dependencies of the form ‘$ww$’, where the 4th words in the two halves are matched. An example lexicon generating the sentence in this example is the following\textsuperscript{1}:

\begin{quote}
\begin{align*}
\text{Ik, Cecilia, Henk, de nijlpaarden} & \rightarrow n \\
\text{zag} & \rightarrow ((n \otimes (n \otimes \alpha))\backslash(\alpha\backslash i)) \\
\text{helpen} & \rightarrow ((n \otimes \alpha)\backslash(\alpha\backslash i)) \\
\text{voeren} & \rightarrow n\backslash i \\
\end{align*}
\end{quote}

\begin{align*}
\text{zag} & \rightarrow \text{H} \rightarrow \text{H} \\
\text{Z} \rightarrow \text{Z} \\
\text{Z,} & \rightarrow (n \otimes (n \otimes \alpha))\backslash(\alpha\backslash i) \\
\text{helpen} & \rightarrow (n \otimes (n \otimes \alpha))\backslash(\alpha\backslash i) \\
\text{voeren} & \rightarrow (n \otimes (n \otimes \alpha))\backslash(\alpha\backslash i) \\
\end{align*}

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

1.2 Constraining UAB\textsuperscript{c} grammars

One constraint that we can impose on UAB\textsuperscript{c} 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, $\alpha/\alpha$, $(s/\alpha)\backslash(s/(\alpha \otimes a))$ are licit types, while $(\alpha/\alpha)/\alpha$, $(s/(\alpha \otimes \beta))\backslash(s/((\alpha \otimes \beta) \otimes a))$ 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{align*}
\sharp & ::= \otimes | / | \backslash \\
F_0 & ::= A \mid F_0\sharp F_0 \\
F_1 & ::= F_1\sharp F_1 \mid F_0\sharp F_0 \mid F_0\sharp F_1 \mid \alpha \\
F_2 & ::= F_1\sharp F_1 \mid F_0 \mid F_2\sharp F_0 \mid F_0\sharp F_2 \\
\end{align*}

The interesting case in this definition are the $F_2$ formulas of the form $A/B$ or $B/A$, with $A$ and $B$ in $F_1$, the others being meant essentially to put these in context. Consider the case of $A/B$, then $\alpha$ occurs exactly once in $A$ and in $B$, since a $F_1$ category contains the variable $\alpha$ by construction. By analogy with lambda terms,\textsuperscript{3}

\textsuperscript{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’.

\textsuperscript{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 $\alpha$ in $B$ as a binder (possibly a pattern-binder), and of the occurrence in $A$ as the bound variable.

An UAB\textsuperscript{\textregistered} grammar is linear if all its lexical assignments are linear. Furthermore, in linear UAB\textsuperscript{\textregistered} 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\textsuperscript{\textregistered} calculus consists of all the rules of the UAB\textsuperscript{\textregistered} calculus in Figure 2 replacing the Cut rules with the following instantiation rule.

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

(2)

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

Observe also that the UAB\textsuperscript{\textregistered} grammars for $a^nb^nc^n$ and $w^v$ languages as well as that for the Dutch cross serial patterns, are all linear. On the other hand, no linear UAB\textsuperscript{\textregistered} 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\textsuperscript{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\textsuperscript{\textregistered} grammars can be parsed in polynomial time. We do this in the next section by providing a parsing algorithm for linear UAB\textsuperscript{\textregistered} grammars.

## 2 Polynomial parsing with linear UAB\textsuperscript{\textregistered} grammars

Linear UAB\textsuperscript{\textregistered} grammars can be parsed in polynomial time by means of a simple extension of parsing algorithm for AB\textsuperscript{\textregistered} 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 $\text{Lex}$ the input lexicon. Cells of the instantiation table are denoted $t_{(i,k,j)}$, where $0 \leq i < j \leq n$ and $0 \leq k \leq |\text{Lex}|$. We extend the construction of formulas with two kinds of variables, $\alpha_k$ and $\alpha(i,k,j)$ where $i$, $k$ and $j$ are as before. The difference between the two kinds of variables is that $\alpha_k$ is an uninstantiated variable while $\alpha(i,k,j)$ is a variable $\alpha_k$ which has been instantiated when an item $(i, \Lambda \rightarrow C, j)$ 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 $\alpha_k$ 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 $\text{AB}^\text{\textregistered}$ in Appendix A.

**Given items** $(i, \Delta \triangleright A^\alpha \rightarrow \Gamma \rightarrow C, k) \text{ and } (k, \Lambda \rightarrow A^B, j)$,

**generate item** $(i, \Delta \triangleright A^\alpha \rightarrow \Gamma \rightarrow C[\alpha := \alpha(i,l,j)], j)$

**update the table** $t_{(i,l,j)} = t_{(i,l,j)} \cup \{B\}$

(3)

**Given item** $(i, \Delta \triangleright \alpha(i,k,m) \rightarrow \Gamma \rightarrow C, j)$ and $A \in t_{(k,l,m)}$

**generate item** $(j, \triangleright A \rightarrow \alpha(i,k,m), j)$

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 $O(|\text{Lex}|^2|\Sigma|^2n^7)$, where $|\text{Lex}|$ and $|\Sigma|$ are the sizes of the lexicon and of the set of subformulas of the lexicon, respectively.

\textsuperscript{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\textsuperscript{\textregistered} 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⊗ 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, (α \ α)/α. To what extent and with what consequences from the computational point of view, is an open subject of investigation.

A Parser

Let an AB⊗ grammar G and a string w₁...wₙ be given. The AB⊗ Mix deductive parser is the triple (I,A,R) presented in Figure 3. See Capelletti and Tamburini [2009a] for the O(|Lex|^2|Σ|^2n^7) implementation of this parsing algorithm.

References


