Parsing with Polymorphic Categorial Grammars

Matteo Capelletti[1] and Fabio Tamburini[2]

[1] Lix, École Polytechnique - France
matteo.capelletti@elisanet.fi
[2] DSLO, University of Bologna - Italy
fabio.tamburini@unibo.it

Abstract. In this paper 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 these grammars.

1 Introduction

The simplest model of a categorial grammar is the so called Ajdukiewicz–Bar-Hillel calculus of [2] and [4]. Syntactic categories are formed from a given set of atoms as *functions* a/b and $b\backslash a$, with b and a categories. The intuitive meaning of a syntactic category of the form a/b (resp. $b\backslash a$) is that it looks for an *argument* of category b to its right (resp. left) to give a category of type a. The resulting grammar system is known to be context-free.

Contemporary categorial grammars in the style of Ajdukiewicz–Bar-Hillel grammars are called *combinatory categorial grammars*, see [25]. Such systems adopt other forms of composition rules which enable them to generate non–context-free languages, see [29; 28]. The other main tradition of categorial grammar, the type-logical grammars of [20; 18], stemming from the work of [15], 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. In this paper, we adopt a different strategy which consists in keeping the elementary rule component of Ajdukiewicz–Bar-Hillel grammar and in introducing *polymorphic* categories, that is syntactic categories that contain 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 ones. In respect to TLG, the polymorphic system we define is *polynomial*, as we will prove by providing a parsing algorithm. In respect to CCG (and most known TLG), our system is not affected by the so called *spurious ambiguity* problem, that is the problem of generating multiple, semantically equivalent, derivations.

Advances in Computational Linguistics.
Research in Computing Science 41, 2009, pp. 87-98

Received 07/11/08
Accepted 15/12/08
Final version 26/01/09

A system somewhat similar to our polymorphic categorial grammar was studied in the late 80s as a kind of feature theoretic categorial syntax by [26] and [30], although, to our knowledge, after these works it has been almost completely neglected. While our system, in its most general form, allows the encoding of feature structures into categorial grammar, our primary concern will be to discuss its generative power and its computational properties. We will present general polymorphic categorial grammars, and show that they are undecidable. Then we will define a restricted class of such grammars which is mildly context-sensitive. We will show this by giving examples of the languages that can and cannot be generated, and by providing a polynomial parser for such grammars.

In relation to other mildly context-sensitive grammar formalism, the categorial grammar framework is well known for its close correspondence between syntactic derivation and the construction of semantic representation, see [27]. For this reason the extensions of the basic calculi with more powerful devices allows for the construction of sophisticated parsing systems dealing in parallel with both the syntax and the semantics of languages.

2 Polymorphic Ajdukiewicz–Bar-Hillel Grammar

In this paper we will assume that categories are not only of the form a/b or $b\backslash a$, but also $a \otimes b$, this is not standard in the categorial literature, but such product categories are useful as we will see in the examples that follow. From now on, we use capital letters A, B and C and their variants as metavariables over formulas. As we said before, an expression of category A/B looks for an expression of category B to its right to give a compound expression of category A and dually for $B\backslash A$. Instead, an expression of category $A \otimes B$ is an expression whose syntactic information is given by a component of category A and another of category B. We assume that the slashes bind more tightly than the product. We define a *sequent* as a pair where the first element, the antecedent, is a list of category, while the second, the succedent, contains a single category, and we write it as $\Gamma \to A$.

The deductive system given in Figure 1, which we call $\mathrm{AB}^{\otimes}$, is a simple modification of the calculus of [14] to which it can easily be proved equivalent.

Identity Axioms:	$A \to A$
Product Axioms:	$A,\ B \to A \otimes B$
Shifting Rules:	$\dfrac{\Gamma \to C/A}{\Gamma, A \to C}\ (S_1)$ $\qquad$ $\dfrac{\Gamma \to A\backslash C}{A, \Gamma \to C}\ (S_2)$
Cut Rules:	$\dfrac{\Gamma, A \to C \quad \Delta \to A}{\Gamma, \Delta \to C}\ (C_1)$ $\qquad$ $\dfrac{\Gamma \to A \quad A, \Delta \to C}{\Gamma, \Delta \to C}\ (C_2)$

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

There are two ways in literature for extending the basic models of categorial grammars to generate non context-free languages. The first is the use of structural rules or other types of composition schemes. These approaches are characteristic of type-logical grammars, see [20; 18; 19], and combinatory categorial grammars (CCG), see [25; 3], and have been widely explored in the past. The second is based on the introduction of *polymorphism*. In this paper, 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 [17; 9]. 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 [15; 24; 8; 7].

Another example of Ajdukiewicz–Bar-Hillel style categorial grammars adopting a form of polymorphism are the unification categorial grammars of [26; 30; 10], where polymorphism is used at the level of feature structures.

In this section, we are going to explore Ajdukiewicz–Bar-Hillel categorial grammars *with product* ($AB^{\otimes}$, for short) extended with different forms of polymorphism. In the first place, we extend the notion of category as to include variables and define a sort of categorial grammar with ML style polymorphism, see [17; 12]. Variables are assumed to be implicitly universally quantified, with quantifiers in prenex position.

Consider the formula $(\alpha\backslash\alpha)/\alpha$ which can be assigned to the word 'and' in a lexicon. Applied to 'John' :: n, it will give the expression 'and John' :: $n\backslash n$, while applied to 'walks' :: $n\backslash s$, it will give the expression 'and walks' :: $(n\backslash s)\backslash(n\backslash s)$, and so forth. Thus the right argument α in $(\alpha\backslash\alpha)/\alpha$ gets instantiated in the application process and the result of such instantiation, a *substitution*, is applied to the value $\alpha\backslash\alpha$. Hence the process of type inference for this kind of polymorphic categorial grammars requires nothing more than type unification and substitution. We call the resulting system Unification Ajdukiewicz–Bar-Hillel grammars with product, UAB$^{\otimes}$ for short.

2.1 Unification Ajdukiewicz–Bar-Hillel Grammars

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

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

Unification of categories is defined in (1). We use $\sharp$ as a variable over $\{/, \backslash, \otimes\}$ connectives. The *substitution* of a formula A for a variable α in a formula B, denoted $B[\alpha := A]$, is defined as follows:

$$\alpha[\alpha := C] = C \qquad \alpha[\beta := C] = \alpha \text{ if } \alpha \neq \beta$$
$$A[\alpha := C] = A \text{ for } A \in \mathcal{A} \qquad (A \sharp B)[\alpha := C] = (A[\alpha := C] \sharp B[\alpha := C]).$$

We write $f \cdot g$ the composition of f and g, defined as $\lambda x.f\ (g\ x)$. Let $V(B)$ be the set of variables occurring in B, we define the *unification* of A and B, denoted $A \approx B$, by the following recursion, which is taken with minor modifications from [5].

$$\begin{array}{lcll} \alpha \approx B & = & [\alpha := B] & \text{if } \alpha \notin V(B) \\ & = & \texttt{Id} & \text{if } \alpha \equiv B \\ & = & \texttt{fail} & \text{otherwise} \\ A \sharp B \approx A' \sharp B' & = & (\sigma A \approx \sigma A') \cdot \sigma & \text{where } \sigma = B \approx B' \\ A \approx \alpha & = & \alpha \approx A & \end{array} \tag{1}$$

The unification Ajdukiewicz–Bar-Hillel calculus, $\text{UAB}^{\otimes}$ is defined in Figure 2.[3]

Identity Axioms:	$A \to A$
Product Axioms:	$A,\ B \to A \otimes B$
Shifting Rules:	$\dfrac{\Gamma \to C/A}{\Gamma, A \to C}\ (S_1)$ $\qquad$ $\dfrac{\Gamma \to A\backslash C}{A, \Gamma \to C}\ (S_2)$
Cut Rules:	$\dfrac{\Gamma, A \to C \quad \Delta \to B}{\Gamma, \Delta \to C(A \approx B)}\ (C_1')$ $\qquad$ $\dfrac{\Gamma \to B \quad A, \Delta \to C}{\Gamma, \Delta \to C(A \approx B)}\ (C_2')$

Fig. 2. Unification Ajdukiewicz–Bar-Hillel calculus, $\text{UAB}^{\otimes}$

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

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

$$a :: s/(b \otimes c) \qquad b :: b$$
$$a :: (s/\alpha)\backslash(s/(b \otimes (\alpha \otimes c))) \qquad c :: c$$

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.

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

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

To show that G_1 indeed generates the language $a^n b^n c^n$, $n \geqslant 1$, we proceed by induction on n. If $n = 1$, then 'abc' is generated by axioms $a/(b\otimes c) \to a/(b\otimes c)$, $b \to b$ and $c \to c$. Assume that G_1 generates $a^n b^n c^n$. Then G_1 assigns a^n the category s/A for some A and $b^n c^n$ the category A. We have $a :: (s/\alpha)\backslash(s/(b\otimes(\alpha\otimes c)))$. Hence, G_1 assigns a^{n+1} the category $s/(b\otimes(A\otimes c))$ and $b^{n+1}c^{n+1}$ the category $b\otimes(A\otimes c)$. We conclude that G_1 generates $a^{n+1}b^{n+1}c^{n+1}$.

Example 2 *We define a $UAB^{\otimes}$ grammar for 'ww', $w \in \{a, b\}^+$. Let grammar G_2 consist of the following assignments:*

$a :: a$	$b :: b$
$a :: s/a$	$b :: s/b$
$a :: (s/\alpha)\backslash(s/(\alpha\otimes a))$	$b :: (s/\alpha)\backslash(s/(\alpha\otimes b))$

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. Here we give an example deduction for the string 'aabaab', using A as for $(s/\alpha)\backslash(s/(\alpha\otimes a))$ and B for $(s/\alpha)\backslash(s/(\alpha\otimes b))$.

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

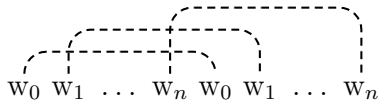
A typical example of non context-freeness of natural language are the so called cross serial dependencies, which can be found, for instance, in Dutch subordinate clauses.

Example 3 *We define a $UAB^{\otimes}$ grammar for Dutch cross-serial dependencies. An example is the following subordinate clause, from [25]:*

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.



A sample lexicon generating the sentence in this example is the following:

Ik, Cecilia, Henk, de nijlpaarden	$:: n$
zag	$:: ((n \otimes (n \otimes \alpha))\backslash c)/(\alpha\backslash i)$
helpen	$:: ((n \otimes \alpha)\backslash i)/(\alpha\backslash i)$
voeren	$:: n\backslash i$

With such a lexicon we obtain the following deduction for the subordinate clause (we write Z for the type of 'zag', H for that of 'helpen' and N for the string 'Ik Cecilia Henk de nijlpaarden').

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

These examples show that the languages generated by UAB$^{\otimes}$ grammars properly include the context-free languages (since AB$^{\otimes}$ grammars are instances of UAB$^{\otimes}$ grammars).

With regard to the generative power of UAB$^{\otimes}$ grammars, it can be proven 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. We can show this by translating in our categorial setting the argument of [13] to prove the Turing completeness of unification based attribute-value grammars. It is possible to encode a Turing machine M in a UAB$^{\otimes}$ grammar $G(M)$ such that $G(M)$ generates the string 'halt' if and only if M halts with a blank input tape; this is enough to conclude the undecidability of UAB$^{\otimes}$ grammars. Polymorphic null assignments, in fact, correspond quite neatly to lexical rules, as proposed in [6], leading to an undecidable formalism.

2.2 Constraining UAB⊗ Grammars

A constrain 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, 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.

$$\begin{array}{lll} \sharp & ::= & \otimes \mid / \mid \backslash \\ F_0 & ::= & A \mid F_0 \sharp F_0 \\ F_2 & ::= & F_1 \sharp' F_1 \mid F_0 \mid F_2 \sharp' F_0 \mid F_0 \sharp' F_2 \end{array} \qquad \begin{array}{lll} \sharp' & ::= & / \mid \backslash \\ F_1 & ::= & F_1 \sharp F_0 \mid F_0 \sharp F_1 \mid \alpha \end{array} \tag{2}$$

This definition of categories may deserve some comments. The interesting cases are the F_2 formulas of the form A/B or $B\backslash A$, with A and B in F_1 (the other cases are 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, 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 results from the UAB$^{\otimes}$ calculus in Figure 2 by replacing the Cut rules with the following *instantiation* rules.

$$\frac{\Delta \to A^B \quad \Gamma, A^\alpha \to C}{\Gamma, \Delta \to C[\alpha := B]} \qquad \frac{\Gamma \to A^B \quad A^\alpha, \Delta \to C}{\Gamma, \Delta \to C[\alpha := B]} \tag{3}$$

Observe that given a linear UAB$^{\otimes}$ grammar adopting the rules in 3, 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[4].

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.

3 Polynomial Parsing with Linear UAB⊗ Grammars

In this section we define a polynomial parsing algorithm for linear UAB$^{\otimes}$ grammars. It is based on a standard agenda-driven chart-parsing method and makes use of an external table, which we call *instantiation table*, for storing the 'partial' instantiations of variables Let n be the length of the input string and *Lex*

[4] 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 [11], see [16]. Hence for the MIX language, a UAB$^{\otimes}$ grammar needs to bind two distinct variables for each symbol, what violates linearity.

the input lexicon. Cells of the instantiation table are denoted $\mathcal{I}_{(i,k,j)}$, 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 spanning between i and j was generated. 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[5].

The parser operates on two kinds of items which we represent as

$$(i, \Delta \triangleright \Gamma \rightarrow A, j) \quad \text{and} \quad (i, \Gamma \triangleleft \Delta \rightarrow A, j)$$

where i and j are integers and $\Delta \triangleright \Gamma \rightarrow A$ and $\Gamma \triangleleft \Delta \rightarrow A$ are *sequents* in which exactly one occurrence of an auxiliary symbols (either $\triangleright$ or $\triangleleft$) appear. These symbols play a role similar to the dot in Earley parsing systems.

An item of the form $(i, \Delta \triangleright \Gamma \rightarrow A, j)$ asserts that $\Delta\Gamma \rightarrow A$ is derivable in linear UAB$^{\otimes}$ and that $w_{i+1} \dots w_j \Rightarrow^* \Delta$. Furthermore, the items have a predictive component.

- In case $A \equiv A' \otimes A''$ and $\Delta\Gamma \equiv A'\, A''$, it asserts that for some context Λ, $w_{l+1} \dots w_i\, A\, \Lambda \Rightarrow^* C$ with $0 \leqslant l < i$. This means that the item has been *predicted* from another item $(l, \Xi \triangleright A\, \Lambda \rightarrow C, i)$.
- In case $A \equiv \alpha_{(i,k,j)}$ and $\Delta\Gamma \equiv B$ for some formula $B \in \mathcal{I}_{(i,k,j)}$, it asserts that for some context Λ, $w_{l+1} \dots w_i\, A^{\alpha_{(i,k,j)}}\, \Lambda \Rightarrow^* C$ with $0 \leqslant l < i$. This means that the item has been *dereferenced* from another item $(l, \Xi \triangleright A^{\alpha_{(i,k,j)}}\, \Lambda \rightarrow C, i)$.

The dual conditions hold for $(i, \Gamma \triangleleft \Delta \rightarrow A, j)$. For simplicity, we write items of the form $(i, \triangleleft \Delta \rightarrow A, j)$ and $(i, \Delta \triangleright \rightarrow A, j)$ as $(i, \Delta \rightarrow A, j)$.

The proposed parsing algorithm for linear UAB$^{\otimes}$ grammars is showed in Figure 3.

The correctness of the algorithm can be proven by adapting the methods of [1; 23] for the CYK and Earley parsers and by observing that the triple (i, k, j) resulting from the instantiation of a variable is determined by the unique name k of the variable α_k and by the span (i, j) over which the instantiation has been determined. Observe that the Completion rules apply to a non-instantiated variable and therefore that instantiated variables behave like constants in the parsing process, whose value is determined by the Dereference rules.

To show that the resulting algorithm is polynomial, we follow the usual argument for agenda-driven chart-based parser evaluation, see for instance [22; 21]. The number of sequents that can occur in a cell of the chart for a given grammar with lexicon Lex is $O(n^2|Lex||\Sigma|)$ where Σ is the set of subformulas of the lexicon[6] and $|Lex|$ is the number of word-category pairs contained in the lexi-

[5] Clearly all these modifications do not affect linearity, and are motivated by correctness and efficiency reasons.

[6] That is for each lexical entry w :: A, Σ contains all the subformulas of A. For example the formula $(a\backslash b)/c$ generates the subformulas $\{(a\backslash b)/c, a\backslash b, a, b, c\}$

Input: a string $w = w_1 \dots w_n$ and an $UAB^{\otimes}$ grammar G.
Output: Accept/reject.
Data Structures: an $n+1 \times n+1$ matrix $\mathcal{T}$, the *chart*, whose cells are sets of sequents, a set of items $\mathcal{N}$, the *agenda*, containing all the items to be processed and the instantiation table $\mathcal{I}$ as defined before.

Initialization:
Let $\mathcal{N} = \varnothing$ and $\mathcal{T}_{(i,j)} = \varnothing \quad \forall i, j$.
For $i = 1$ to n do
$\mathcal{N} = \mathcal{N} \cup \{ (i-1, A \to A, i) \mid w_i :: A \in Lex\}$
Main cycle:
While $\mathcal{N} \neq \varnothing$ do
remove one item $\nu = (i, \Gamma \to A, j)$ from $\mathcal{N}$.
If $\Gamma \to A \notin \mathcal{T}_{(i,j)}$, then
add $\Gamma \to A$ to chart $\mathcal{T}_{(i,j)}$
Shifting:
If $\nu = (i, \Gamma \to C/A, j)$, then add $(i, \Gamma \triangleright A \to C, j)$ to $\mathcal{N}$.
If $\nu = (i, \Gamma \to A\backslash C, j)$, then add $(i, A \triangleleft \Gamma \to C, j)$ to $\mathcal{N}$.
Prediction:
If $\nu = (i, \Gamma \triangleright A \otimes B\Delta \to C, j)$, then add $(j, \triangleright A, B \to A \otimes B, j)$ to $\mathcal{N}$.
If $\nu = (i, \Gamma A \otimes B \triangleleft \Delta \to C, j)$, then add $(i, A, B\triangleleft \to A \otimes B, i)$ to $\mathcal{N}$.
ϵ-Scanning:
If $\nu = (i, \Gamma \triangleright A\Delta \to C, j)$ and $\epsilon \Rightarrow^{+} A$, then add $(i, \Gamma A \triangleright \Delta \to C, j)$ to $\mathcal{N}$.
If $\nu = (i, \Gamma A \triangleleft \Delta \to C, j)$ and $\epsilon \Rightarrow^{+} A$, then add $(i, \Gamma \triangleleft A\Delta \to C, j)$ to $\mathcal{N}$.
Completion:
If $\nu = (i, \Gamma \triangleright A\Delta \to C, j)$, then for all $\Lambda \to A \in \mathcal{T}_{(j,k)}$, add $(i, \Gamma A \triangleright \Delta \to C, k)$ to $\mathcal{N}$.
If $\nu = (i, \Gamma A \triangleleft \Delta \to C, j)$, then for all $\Lambda \to A \in \mathcal{T}_{(k,i)}$, add $(k, \Gamma \triangleleft A\Delta \to C, j)$ to $\mathcal{N}$.
If $\nu = (i, \Lambda \to A, j)$, then $\begin{cases} \text{for all } \Gamma \triangleright A\Delta \to C \in \mathcal{T}_{(k,i)}, \text{ add } (k, \Gamma A \triangleright \Delta \to C, j) \text{ to } \mathcal{N}, \\ \text{for all } \Gamma A \triangleleft \Delta \to C \in \mathcal{T}_{(j,k)}, \text{ add } (i, \Gamma \triangleleft A\Delta \to C, k) \text{ to } \mathcal{N}. \end{cases}$
If $\nu = (i, \Gamma \triangleright A^{\alpha_l}\, \Delta \to C, j)$, then
for all $\Lambda \to A^B \in \mathcal{T}_{(j,k)}$
add $(i, \Gamma\, A^{\alpha_l} \triangleright \Delta \to C[\alpha_l := \alpha_{(i,l,k)}], k)$ to $\mathcal{N}$ and update $\mathcal{I}_{(i,l,k)} = \mathcal{I}_{(i,l,k)} \cup \{B\}$.
If $\nu = (i, \Gamma A^{\alpha_l} \triangleleft \Delta \to C, j)$, then
for all $\Lambda \to A^B \in \mathcal{T}_{(k,i)}$
add $(k, \Gamma \triangleleft A^{\alpha_l}\, \Delta \to C[\alpha_l := \alpha_{(k,l,j)}], j)$ to $\mathcal{N}$ and update $\mathcal{I}_{(k,l,j)} = \mathcal{I}_{(k,l,j)} \cup \{B\}$.
If $\nu = (i, \Lambda \to A^B, j)$, then
for all $\Gamma \triangleright A^{\alpha_l}\Delta \to C \in \mathcal{T}_{(k,i)}$
add $(k, \Gamma A^{\alpha_l} \triangleright \Delta \to C[\alpha_l := \alpha_{(k,l,j)}], j)$ to $\mathcal{N}$ and update $\mathcal{I}_{(k,l,j)} = \mathcal{I}_{(k,l,j)} \cup \{B\}$.
for all $\Gamma A^{\alpha_l} \triangleleft \Delta \to C \in \mathcal{T}_{(j,k)}$
add $(i, \Gamma \triangleleft A^{\alpha_l}\Delta \to C[\alpha_l := \alpha_{(i,l,k)}], k)$ to $\mathcal{N}$ and update $\mathcal{I}_{(i,l,k)} = \mathcal{I}_{(i,l,k)} \cup \{B\}$.
Dereference:
If $\nu = (i, \Gamma \triangleright \alpha_{(k,l,m)}\, \Delta \to C, j)$ and $A \in \mathcal{I}_{(k,l,m)}$ then add $(j, \triangleright A \to \alpha_{(k,l,m)}, j)$ to $\mathcal{N}$.
If $\nu = (i, \Gamma \alpha_{(k,l,m)} \triangleleft \Delta \to C, j)$ and $A \in \mathcal{I}_{(k,l,m)}$ then add $(j, A \triangleleft \to \alpha_{(k,l,m)}, j)$ to $\mathcal{N}$.
Termination: If, when $\mathcal{N} = \varnothing$, $\Gamma \to s \in \mathcal{T}_{(0,n)}$, for some Γ, then accept, else reject.

Fig. 3. Recognition algorithm for linear UAB$^{\otimes}$grammars.

con. Then, the time is dominated by the Completion rules, that gives a global asymptotic complexity of $O(n^5|Lex||\Sigma|)$.

4 Conclusion

In this paper 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* defines a linguistically interesting class of categorial grammars. Most notably, it locates the system among the mildly context-sensitive formalisms. Another pleasant aspect of the resulting system, at least with respect to any other CG-based mildly context-sensitive categorial formalism (be it CCG or type-logical grammar) is the absence of spurious ambiguity.

The work in [8] presents a decision procedure for a kind of polymorphic categorial grammar allowing at most two instances of the same variable in argument position and one in value (e.g. $(X\backslash X)/X$). However, with such kind of polymorphism we can generate languages that are beyond the mildly context-sensitives (for instance, we can easily generate indexed languages such as $\{www|w \in \{a,b\}^+\}$). This extension, and its implications on recognition complexity, are currently being investigated.

Despite the fact that in this paper we have been concerned only with a recognition algorithms, a proper *parsing* algorithm that provides all the parses of the input, handling also semantic information, can be easily provided. In that case, the correspondence between syntax and semantics typical of categorial grammars and the absence of spurious ambiguity of our systems become important ingredients of the resulting natural language parser.

We wish to conclude this section 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 allow more than two occurrences of this variable. Then bounded languages such as w^i or $\mathrm{a}_1^i\mathrm{a}_2^i\ldots\mathrm{a}_n^i$, which are generalizations of 'ww' and $\mathrm{a}^i\mathrm{b}^i\mathrm{c}^i$, can easily be generated and recognized in polynomial time.

References

1. A. Aho and J. Ullman. *The Theory of Parsing, Translation and Compiling*, volume 1: Parsing. Prentice-Hall, INC., 1972.
2. K. Ajdukiewicz. Die syntaktische Konnexität. *Studia Philosophica*, 1:1–27, 1935.
3. J. Baldridge. *Lexically Specified Derivational Control in Combinatory Categorial Grammar*. PhD thesis, University of Edinburgh, 2002.
4. Y. Bar-Hillel. A quasi-arithmetical notation for syntactic description. *Language*, 29:47–58, 1953.
5. H. Barendregt. Lambda calculus with types. In S. Abramsky, Dov M. Gabbay, and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 2, pages 117–309. Oxford University Press, 1992.
6. B. Carpenter. The generative power of categorial grammars and head-driven phrase structure grammars with lexical rules. *Computational Linguistics*, 17(3):301–313, 1991.
7. S. Clark and J. R. Curran. Wide-coverage efficient statistical parsing with ccg and log-linear models. *Computational Linguistic*, 33(4):493–552, 2007.
8. M. Emms. Parsing with polymorphism. In *EACL*, pages 120–129, 1993.
9. J.-Y. Girard, P. Taylor, and Y. Lafont. *Proofs and Types*. Cambridge University Press, 1989.
10. D. Heylen. *Types and Sorts. Resource logic for feature checking*. PhD thesis, UiL-OTS, Utrecht, 1999.
11. J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, 1979.
12. G. Huet. A uniform approach to type theory. In *Logical foundations of functional programming*, pages 337–397. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1990.
13. M. Johnson. *Attribute-Value Logic and the Theory of Grammar*, volume 16 of *CSLI Lecture Notes*. CSLI, Stanford, California, 1988.
14. M. Kandulski. The equivalence of nonassociative Lambek categorial grammars and context-free grammars. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 34:41–52, 1988.
15. J. Lambek. The mathematic of sentence structure. *American Mathematical Monthly*, 65(3):154–170, 1958.
16. P. Linz. *An introduction to formal languages and automata*. D. C. Heath and Company, Lexington, MA, USA, 1990.
17. R. Milner. A theory of type polymorphism in programming. *Journal of Computer and System Sciences*, 17:348–375, 1978.
18. M. Moortgat. Categorial type logics. In J. van Benthem and A. ter Meulen, editors, *Handbook of Logic and Language*, pages 93–177. Elsevier, Amsterdam, 1997.
19. R. Moot. *Proof Nets for Linguistic Analysis*. PhD thesis, UiL-OTS, Utrecht, 2002.

20. G. Morrill. *Type Logical Grammar: Categorial Logic of Signs*. Kluwer, Dordrecht, 1994.
21. M.-J. Nederhof and G. Satta. Tabular parsing. In C. Martin-Vide, V. Mitrana, and G. Paun, editors, *Formal Languages and Applications, Studies in Fuzziness and Soft Computing 148*, pages 529–549. Springer, 2004.
22. G. Satta and O. Stock. Bidirectional context-free grammar parsing for natural language processing. *AIJ: Artificial Intelligence*, 69, 1994.
23. K. Sikkel. Parsing schemata and correctness of parsing algorithms. *Theoretical Computer Science*, 199, 1998.
24. M. Steedman. Dependency and coordination in the grammar of dutch and english. *Language*, 61(3):523–568, 1985.
25. M. Steedman. *The Syntactic Process*. The MIT Press, 2000.
26. Hans Uszkoreit. Categorial unification grammars. In *COLING*, pages 187–194, 1986.
27. J. van Benthem. *Essays in Logical Semantics*. Reidel, Dordrecht, 1986.
28. K. Vijay-Shanker and D. J. Weir. The equivalence of four extensions of context-free grammars. *Mathematical Systems Theory*, 27, 1994.
29. D. J. Weir and A. K. Joshi. Combinatory categorial grammars: Generative power and relationship to linear context-free rewriting systems. In *Meeting of the Association for Computational Linguistics*, pages 278–285, 1988.
30. H. Zeevat. Combining categorial grammar and unification. In U. Reyle and C. Rohrer, editors, *Natural Language Parsing and Linguistic Theories*, pages 202–229. D. Reidel, Dordrecht, 1988.