On the Complexity of Abstract Categorial Grammars

In this abstract we investigate the respective complexities of the membership and the universal membership problems for Abstract Categorial Grammars [dG01]. This problem has already been addressed in [YK05] and we present here some more precise results and some new ones.

Abstract Categorial Grammars [dG01] are appealing since they can represent many well-known formalisms [dGP04] while using a small set of primitives. They use the linear λ-calculus which is associated to the intuitionnistic implicative linear logic with proper axioms ($\mathbf{IILL}_{ax}$) via the Curry-Howard isomorphism. The formulae of $\mathbf{IILL}_{ax}$ are built from a finite set of *atoms* A and the binary connective $\multimap$. $\mathcal{I}_A$ denotes the set formulae of $\mathbf{IILL}_{ax}$ that can be built from A. In the linear λ-calculus, the proper axioms of $\mathbf{IILL}_{ax}$ are represented by constants having the corresponding type. Thus linear λ-terms are built on *higher-order signatures* like $\Sigma = (A, C, \tau)$, where A is the finite set of types on which formulae are built, C is the set of constants and τ is a function which associates a formula from $\mathcal{I}_A$ to each constant of C. We adopt the convention that the signature Σ is the triple (A, C, τ), and the signature Σ_i is the triple (A_i, C_i, τ_i) for any $i \in \mathbb{N}$. We assume that the reader is familiar to λ-calculus, the notions of free variables, $\beta\eta$-reduction *etc.*.. In the following we assume that we are given an infinite enumerable set of variables $\mathcal{V}$. Given a higher-order signature Σ the family $(\Lambda^\alpha_\Sigma)_{\alpha \in \mathcal{I}_A}$ of linear λ-terms built on Σ is defined as the smallest family verifying:

1. if $c \in C$ then $c \in \Lambda^{\tau(c)}_\Sigma$,
2. if $x \in \mathcal{V}$ and $\alpha \in \mathcal{I}_A$ then $x^\alpha \in \Lambda^\alpha_\Sigma$
3. if $t \in \Lambda^\alpha_\Sigma$ and $x^\beta \in FV(t)$[1] then $\lambda x^\beta.t \in \Lambda^{\beta \to \alpha}_\Sigma$
4. if $t_1 \in \Lambda^{\beta \to \alpha}_\Sigma$, $t_2 \in \Lambda^\beta_\Sigma$ and whenever $x^\gamma \in FV(t_1)$ (*resp.* $x^\gamma \in FV(t_2)$), for any $\gamma' \in \mathcal{I}_A$, $x^{\gamma'} \notin FV(t_2)$ (*resp.* $x^{\gamma'} \notin FV(t_1)$) then $t_1 t_2 \in \Lambda^\alpha_\Sigma$.

The set $\bigcup_{\alpha \in \mathcal{I}_A} \Lambda^\alpha_\Sigma$ is denoted by Λ_Σ.

Given two higher-order signatures Σ_1 and Σ_2 a homomorphism between Σ_1 and Σ_2 is a pair $\mathcal{H} = (f, g)$ such that f is a function from $\mathcal{I}_{A_1}$ to $\mathcal{I}_{A_2}$ such that $f(\beta \multimap \alpha) = f(\alpha) \multimap f(\beta)$ and g is a function from $\Lambda^\alpha_{\Sigma_1}$ to $\Lambda^{f(\alpha)}_{\Sigma_2}$ verifying:

[1] $FV(t)$ denotes the set of variables that are free in t.

1. if $c \in C_1$ then $g(c)$ is a closed term (*i.e.* $FV(f(c)) = \emptyset$) of $\Lambda_{\Sigma_2}^{f(\tau_1(c))}$,
2. if $x \in \mathcal{V}$ then $g(x^\alpha) = x^{f(\alpha)}$,
3. if $\lambda x^\beta.t \in \Lambda_{\Sigma_1}^\alpha$ then $g(\lambda x^\beta.t) = \lambda x^{f(\beta)}.g(t)$ and,
4. if $t_1 t_2 \in \Lambda_{\Sigma_1}^\alpha$ then $g(t_1 t_2) = g(t_1)g(t_2)$.

It should be clear that whenever $t \in \Lambda_{\Sigma_1}^\alpha$ then $g(t) \in \Lambda_{\Sigma_2}^{f(\alpha)}$. We will write $\mathcal{H}(\alpha)$ and $\mathcal{H}(t)$ instead of $f(\alpha)$ and $g(t)$.

An ACG defined as a quadruple $\mathcal{G}(\Sigma_1, \Sigma_2, \mathcal{L}, S)$ where Σ_1 and Σ_2 are higher-order signatures, respectively the *abstract vocabulary* and the *object vocabulary*, $\mathcal{L}$ is a homomophism between Σ_1 and Σ_2, the *lexicon* and S is an element of A_1, the *accepting type*. Then $\mathcal{G}$ defines two languages:

1. the *abstract language*: $\mathcal{A}(\mathcal{G}) = \{M \in \Lambda_{\Sigma_1}^S | M \text{ is closed}\}$,
2. the *object language*: $\mathcal{O}(\mathcal{G}) = \{M \in \Lambda_{\Sigma_2} | \exists N \in \mathcal{A}(\mathcal{G}).\mathcal{L}(N) =_{\beta\eta} M\}$.

An ACG $\mathcal{G} = (\Sigma_1, \Sigma_2, \mathcal{L}, S)$ is said *lexicalized* if for all $c \in C_1$, $\mathcal{L}(c)$ contains at least the occurrence of a constant in C_2. In [YK05], both membership and universal membership of lexicalized ACGs are shown to be in NP.

ACGs are classified into a hierarchy which is based on the notion of *order of a type*. The order of an atomic type α is $ord(\alpha) = 1$ and $ord(\alpha \to \beta) = \max(ord(\alpha) + 1, ord(\beta))$. The definition of order is extended to higher-order signatures and $ord(\Sigma) = \max\{ord(\tau(c)) | c \in C\}$; and to homomophisms between signatures. The order of a homomophism $\mathcal{H}$ between Σ_1 and Σ_2 is $ord(\mathcal{H}) = \max\{ord(\mathcal{H}(\alpha)) | \alpha \in A_1\}$. Then the set $\mathbf{G}(n, m)$ is the set of ACGs $\mathcal{G} = (\Sigma_1, \Sigma_2, \mathcal{L}, S)$ such that $ord(\Sigma_1) \leq n$ and $ord(\mathcal{L}) \leq m$.

In what follows we show that the membership problem for the grammars of $\mathbf{G}(2, n)$ is polynomial. We also show that the universal membership problem is NP-complete for lexicalized grammars of $\mathbf{G}(2, 2)$ and we exhibit a lexicalized grammar of $\mathbf{G}(3, 1)$ whose language is NP-complete. These last results are an improvement over [YK05] who shows that the universal membership problem is NP-complete for lexicalized ACGs of $\mathbf{G}(4, 2)$ and exhibit a lexicalized ACG of $\mathbf{G}(4, 3)$ whose language is NP-complete. Furthermore, if P$\neq$NP, these results are optimal with respect to the hierarchy $\mathbf{G}(n, p)$. Indeed, since we show that the membership problem for grammars of $\mathbf{G}(2, n)$ is polynomial it is not possible (if P$\neq$NP) to find a grammar whose language is NP-complete in $\mathbf{G}(2, n)$; and, it is obvious that the universal membership is polynomial for grammars in $\mathbf{G}(2, 1)$.[2]

The proof that the membership problem for grammars of $\mathbf{G}(2, n)$ is polynomial is based on a result of [Sal06]. In that paper,the subterms of a λ-term u

[2] The normal forms of terms in the language, noted with de Bruijn convention, can easily be shown to be recognized by a bottom-up tree automaton whose size is linear with respect to the size of the grammar. Since normalizing a linear λ-term can be done in polynomial time, this gives the result.

are denoted by the pairs $(C[], t)$ (where $C[]$ is a context[3]) such that $C[t] = u$. Furthermore, these subterms are used as atomic types in order to type linear λ-terms. So, given u an element of Λ_Σ which is in long normal form, the family of sets $(\mathcal{D}_u^\alpha)_{\alpha \in \mathcal{I}_A}$ is defined as the smallest family verifying:

1. if $\alpha \in A$ then $\mathcal{D}_u^\alpha = \{(C[], t) \in \mathcal{S}_t | t \in \Lambda_\Sigma^\alpha\}$,
2. $\mathcal{D}_u^{\beta \multimap \alpha} = \mathcal{D}_u^\beta \times \mathcal{D}_u^\alpha$.

The elements of $\mathcal{D}_u^\alpha$ are then used to type terms of Λ_Σ^α; the rules used to type terms are the following:

$$\frac{d \in \mathcal{D}_u^\alpha}{u; x^\alpha : d \vdash x^\alpha : d}\text{Axiom} \qquad \frac{(C[], a) \in \mathcal{S}_u}{u;\ \vdash a : \theta(C[], a)}\text{Constant}$$

$$\frac{u; \Gamma, x^\alpha : d \vdash t : e}{u; \Gamma \vdash \lambda x^\alpha . t : d \multimap e}\lambda\text{−abst.} \qquad \frac{u; \Gamma_1 \vdash t_1 : d \multimap e \quad u; \Gamma_2 \vdash t_2 : d}{u; \Gamma_1, \Gamma_2 \vdash t_1 t_2 : e}\text{App.}$$

Given u an element of Λ_Σ^α in long normal form and $(C[], t) \in \mathcal{S}_u$, $\theta(C[], t)$ is defined as follows:

1. if $C[] = C'[[]t']$ then $\theta(C[], t) = \theta(C'[t[]], t') \multimap \theta(C'[], tt')$,
2. if $t = \lambda x.t'$ then $\theta(C[], t) = \theta(C[\lambda x.C_{t',x}[]], x) \multimap \theta(C[\lambda x.[]], t')$,
3. $\theta(C[], t) = (C[], t)$ otherwise.

It is then proved that for u closed and in long normal form, we have $u; \vdash v : \theta([], u)$ is derivable if and only if $v =_{\beta\eta} u$. Thus to prove that a term u of Λ_{Σ_2} is an element of $\mathcal{O}(\mathcal{G})$ it suffices to construct a term t of $\Lambda_{\Sigma_1}^S$ such that $u; \vdash \mathcal{L}(t) : \theta([], u)$. To this end we saturate a set $\mathcal{H}$ of pairs (α, d) of $(\{\alpha\} \times \mathcal{D}_u^\alpha)_{\alpha \in A_1}$. During one step, we transform the set $\mathcal{H}$ into a set $\mathcal{H}'$ in the following way:

1. if there is $c \in C_1$ such that $\tau_1(c) = \alpha$ with $\alpha \in A_1$ and for $d \in \mathcal{D}_u^{\mathcal{L}(\alpha)}$, $u; \vdash \mathcal{L}(c) : d$ is derivable then we let $\mathcal{H}' = \mathcal{H} \cup \{(\alpha, d)\}$
2. if there is $c \in C_1$ such that $\tau_1(c) = \alpha_1 \multimap \cdots \multimap \alpha_n \multimap \alpha_0$ with $\alpha_i \in A_1$ for all $i \in [0, n]$ and for all $i \in [1, n]$ $(d_i, \alpha_i) \in \mathcal{H}$ and $u; \vdash \mathcal{L}(c) : d_1 \multimap \cdots \multimap d_n \multimap d_0$, then we let $\mathcal{H}' = \mathcal{H} \cup \{(\alpha_0, d_0)\}$.

It is obvious that, with these rules, one may build a set containing the pair $(S, \theta([], u))$ if and only if there is $t \in \Lambda_{\Sigma_1}^S$ such that $u; \vdash \mathcal{L}(t) : \theta([], u)$ is derivable, *i.e.* such that $\mathcal{L}(t) =_{\beta\eta} u$ or $u \in \mathcal{O}(\mathcal{G})$. This algorithm can easily be implemented in polynomial time (parameters of the grammar being allowed to appear as exponents), since, the size of an element of $\mathcal{D}_u^{\mathcal{L}(\alpha)}$ is bounded by the product of the size of α and of the size of u. This finally shows that the membership problem for ACGs of $\mathbf{G}(2, n)$ is polynomial.

[3]That is to say that $C[]$ is a λ-term with a hole.

We now show that the universal membership problem for grammars of $\mathbf{G}(2,2)$ is NP-complete. We reduce this problem to the X3C problem which is known to be NP-complete [GJ79]. X3C problems have as input a pair (X, B) where $X = \{a_1; \ldots; a_{3n}\}$ is a set of $3n$ pairwise distinct elements and $B = \{B_1; \ldots; B_m\}$ is a set where $B_i = \{a_{i_1}; a_{i_2}; a_{i_3}\}$ with $1 \leq i_1 < i_2 < i_3 \leq 3n$. Solving an X3C problem amounts to find $C \subseteq B$ such that C is a partition of X. To prove the NP-hardness of the universal membership problem of lexicalized ACGs of $\mathbf{G}(2,2)$, for any instance of an X3C problem (X, B) we give an ACG $\mathcal{G}_{X,B}$ and a term $t_{X,B}$ such that $t_{X,B} \in \mathcal{O}(\mathcal{G}_{X,B})$ if and only if the X3C problem admits a solution. We let $\mathcal{G}_{X,B} = (\Sigma_1, \Sigma_2, \mathcal{L}, D_0)$ where $A_1 = \{D_0; \ldots; D_n\}$, $C_1 = \{E\} \cup \{E_{B_i,k_1,k_2,k_3,k} | B_i \in B \wedge 0 \leq k < n \wedge 1 \leq k_1 < k_2 < k_3 \leq k\}$, $\tau_1(E) = D_n$ and $\tau_1(E_{B_i,k_1,k_2,k_3,k}) = D_{k+1} \multimap D_k$. We also let $A_2 = \{\iota\}$, $C_2 = \{e\} \cup X$ with $g \notin X$, $\tau_2(a_i) = \iota$ for $1 \geq i \geq 3n$ and $\tau_2(e) = \underbrace{\iota \multimap \cdots \multimap \iota}_{3n} \multimap \iota$. We then let

1. $\mathcal{L}(D_k) = \underbrace{\iota \multimap \cdots \multimap \iota}_{3k} \multimap \iota$,
2. $\mathcal{L}(E) = \lambda x_1 \ldots x_{3n}.e x_1 \ldots x_{3n}$ and,
3. $\mathcal{L}(E_{B_i,k_1,k_2,k_3,k}) =$
   $\lambda g x_1 \ldots x_{k_1-1} x_{k_1+1} \ldots x_{k_2-1} x_{k_2+1} \ldots x_{k_3-1} x_{k_3+1} \ldots x_{3k}.$
   $g x_1 \ldots x_{k_1-1} a_{i_1} x_{k_1+1} \ldots x_{k_2-1} a_{i_2} x_{k_2+1} \ldots x_{k_3-1} a_{i_3} x_{k_3+1} \ldots x_{3k}$

It is then easy to prove that the term $e a_1 \ldots a_{3m}$ is in $\mathcal{O}(\mathcal{G}_{X,B})$ if and only if (X, B) admits a solution.

We now construct a lexicalized ACG of $\mathbf{G}(3,1)$ whose language is NP-complete. The language recognized by this grammar contains an encoding of the set of 3-PARTITION problems that admit a solution. A 3-PARTITION problem is a pair $(\{s_1; \ldots; s_{3m}\}, n)$ where n is an integer and for all $i \in [1, 3m]$ s_i is an integer verifying $\frac{n}{4} < s_i < \frac{n}{2}$. Such a problem is said to admit a solution if there is a partition $(S_i)_{i\in[1,m]}$ of $\{s_1; \ldots; s_{3m}\}$ such that for all $i \in [1, m]$ $\sum_{s\in S_i} s = n$. Remark that the S_i must exactly contain three elements. Determining whether a 3-PARTITION problem admits a solution is known to be NP-complete [GJ79]. We now build $\mathcal{G} = (\Sigma_1, \Sigma_2, \mathcal{L}, S)$ with the desired properties. We let $A_1 = \{B_1; B_2; B_3; C; D; E; L; S\}$, $C_1 = \{e; e'; nil; f_1; f_2; f_3; nil; cons; h\}$ with:

1. $\tau_1(e) = (B_1 \multimap B_2 \multimap B_3 \multimap C \multimap D) \multimap S$,
2. $\tau_1(e') = L \multimap S$,
3. $\tau_1(f_1) = \tau_1(f_2) = \tau_1(f_3) = (B_1 \multimap B_2 \multimap B_3 \multimap C \multimap D) \multimap (B_1 \multimap B_2 \multimap B_3 \multimap C \multimap D)$,
4. $\tau_1(cons) = E \multimap L \multimap L$,
5. $\tau_1(nil) = L$ and

6. $\tau_1(h) = (E \multimap E \multimap E \multimap E \multimap S) \multimap (B_1 \multimap B_2 \multimap B_3 \multimap C \multimap D)$.

We let $A_2 = \{*\}$, $C_2 = \{a, b, c, d, o\}$ with $\tau_2(a) = * \multimap * \multimap *$, $\tau_2(b) = \tau_2(c) = \tau_2(d) = * \multimap *$ an $\tau_2(o) = *$. Finally we define the lexicon as follows:

1. $\mathcal{L}(\alpha) = *$ for all $\alpha \in A_1$,
2. $\mathcal{L}(e) = \lambda f.f\, o\, o\, o\, o$
3. $\mathcal{L}(e') = \lambda x.d\, x$
4. $\mathcal{L}(f_1) = \lambda f\, x_1\, x_2\, x_3\, y.f\, (b\, x_2)\, x_2\, x_3\, (c\, y)$
5. $\mathcal{L}(f_2) = \lambda f\, x_1\, x_2\, x_3\, y.f\, x_1\, (b\, x_2)\, x_3\, (c\, y)$
6. $\mathcal{L}(f_3) = \lambda f\, x_1\, x_2\, x_3\, y.f\, x_1\, x_2\, (b\, x_3)\, (c\, y)$
7. $\mathcal{L}(cons) = \lambda x\, y.a\, x\, y$
8. $\mathcal{L}(nil) = o$
9. $\mathcal{L}(h) = \lambda f\, x_1\, x_2\, x_3\, y.f\, (d\, x_1)\, (d\, x_2)\, (d\, x_3)\, (d\, y)$

The idea behind the reduction is that the abstract constant *cons* codes for a list constructor while *nil* represents the empty list, the constant h takes a list where there are four places which are not specified and give them the type of four kinds of stacks, B_1, B_2, B_3 and C, the constant f_i pushes one b on the stack B_i and at the same time it pushes with a c on the stack C at the object level. The constants e closes the bottom of the stack with an o and the constant e' ends a list. Thus the grammar generates lists that contain integers of two kinds represented as monadic trees of b's or monadic trees of c's. The construction guaranties that the integers made of b's can be partitionned in triples $\{p_1; p_2; p_3\}$ which are put in bijection with the integers made of c's such that if n is the integer associated to $\{p_1; p_2; p_3\}$ we have $n = p_1 + p_2 + p_3$. Thus verifying that a certain 3-PARTITION problem $(\{s_1; \ldots; s_{3m}\}, n)$ has a solution amounts to check whether a list that contains each s_i represented with b's and m times the integer n represented with c's is an element of $\mathcal{O}(\mathcal{G})$.

References

[dG01] Philippe de Groote. Towards abstract categorial grammars. In Association for Computational Linguistic, editor, *Proceedings 39th Annual Meeting and 10th Conference of the European Chapter*, pages 148–155. Morgan Kaufmann Publishers, 2001.

[dGP04] Philippe de Groote and Sylvain Pogodalla. On the expressive power of abstract categorial grammars: Representing context-free formalisms. *Journal of Logic, Language and Information*, 13(4):421–438, 2004.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability – A Guide to the Theory of NP-Completeness*. Freeman, San Francisco, 1979.

[Sal06] Sylvain Salvati. Syntactic descriptions: a type system for solving matching equations in the linear λ-calculus. In *proceedings of the 17th International Conference on Rewriting Techniques and Applications*, 2006.

[YK05] Ryo Yoshinaka and Makoto Kanazawa. The complexity and generative capacity of lexicalized abstract categorial grammars. In *LACL*, pages 330–346, 2005.