Using and Extending the ACG technology: Endowing Categorial Grammars with an Underspecified Semantic Representation

Sylvain Pogodalla

Using and Extending ACG technology: Endowing Categorial Grammars with an Underspecified Semantic Representation

Sylvain Pogodalla

LORIA - Campus Scientifique
BP239
F-54602 Vandœuvre-lès-Nancy

Abstract

Underlining the computational properties of Abstract Categorial Grammars, we show how to extend them while keeping some of their properties. In particular, we show how to enable the modeling of non-linear languages. We give as an example the encoding of an underspecified semantic representation language, and apply it to categorial grammars. So that we can model ambiguities of quantifier scopes only at the semantic level, and not at the syntactic level anymore.

Introduction

Instead of considering Abstract Categorial Grammars (ACGs) as a standard grammatical formalism, de Groote (2001) presents them as a grammatical framework in which other existing grammatical models may be encoded. This allows to make the latter formalisms take benefit of the ACG features, hence to have them communicate thanks to this common framework.

Different syntactical formalisms have been encoded into ACGs (Tree Adjoining Grammars (de Groote, 2002), m-Linear Context-Free Rewriting systems (de Groote and Pogodalla, 2003)), but ACGs basic definitions, based on linear λ-terms, may lack some expressive power, for instance to model semantic representation languages.

The aim of this paper is to underline the computational properties of ACGs (section 1) and draw from them some language extensions (section 2). Then in section 3, as an example, we model an underspecified semantic representation language (Bos, 1995; Blackburn and Bos, 2003) using the proposed extensions.

1 ACG Principles

ACGs generate two languages: an *abstract language* and an *object language*. Whereas the abstract language may appear as a set of grammatical or parse structures, the object language may appear as its realization, or the concrete language it generates. The two generated languages are sets of linear λ-terms.

An ACG $\mathscr{G}$ defines:

- two sets of typed linear λ-terms: Λ_1 (based on the typed constant set C_1) and Λ_2 (based on the typed constant set C_2);
- a morphism $\mathscr{L} : \Lambda_1 \rightarrow \Lambda_2$;
- a distinguished type S.

Then the abstract language $\mathcal{A}(\mathscr{G})$ and the object languages $\mathcal{O}(\mathscr{G})$ are defined as follows:

$$\mathcal{A}(\mathscr{G}) = \{t \in \Lambda_1 | t : S\} \qquad \mathcal{O}(\mathscr{G}) = \{t \in \Lambda_2 | \exists u \in \mathcal{A}(\mathscr{G})\ t = \mathscr{L}(u)\}$$

Note that $\mathscr{L}$ binds the parse structures in $\mathcal{A}(\mathscr{G})$ to the concrete expressions of $\mathcal{O}(\mathscr{G})$. Depending on the choice of Λ_1, Λ_2 and $\mathscr{L}$, it maps parse trees to contex free languages, proofs (as λ-terms) to strings like categorial grammars (de Groote, 2001), derivation trees to derived trees for TAGs (de Groote, 2002) or derivation trees to m-linear context free languages (de Groote and Pogodalla, 2003). Of course, this link between an abstract and a concrete structure concerns not only syntactical formalisms, but also semantic formalisms as the link between proofs (as λ-terms) and λ-terms to get Montague's semantics for categorial grammars in (de Groote, 2001) shows.

When considering such a grammatical formalism, two major questions arise: what is its expressive power and what is its computational complexity? Partial answers to the first question have already been given, and the aim of this paper is to complete this answer. But this has to rely on the answer to the second question. Let us first describe how to use ACGs in grammatical analysis.

Let $t \in \Lambda_2$ be an expression. It is a λ-term modeling for instance a string or a tree. To know if t is an expression of the language $\mathcal{O}(\mathscr{G})$, we have to find $u \in \mathcal{A}(\mathscr{G})$ such that $t = \mathscr{L}(u)$. Assume we have such a u then, replacing any constant c_i (we may have $c_i = c_j$) by x_i to get u', we have that $\mathscr{L}((\lambda x_1 \cdots x_n.u')c_1 \cdots c_n) = t$, or, because $\mathscr{L}$ is a morphism, we have $(\lambda x_1 \cdots x_n.\mathscr{L}(u'))\mathscr{L}(c_1) \cdots \mathscr{L}(c_n) = t$ with u' (hence $\mathscr{L}(u')$) with no constant in it.

Conversely, let $u' \in \Lambda_1$ such that u' has no constant and that the relation $(\lambda x_1 \cdots x_n.\mathscr{L}(u'))\mathscr{L}(c_1) \cdots \mathscr{L}(c_n) = t$ holds, then if $u = (\lambda x_1 \cdots x_n.u')c_1 \cdots c_n$,

we have $\mathscr{L}(u) = t$. Since if u' has no constant $u' = \mathscr{L}(u')$ (only the type may change), the problem of checking whether t is in $\mathcal{O}(\mathscr{G})$ reduces to solving the equation

$$(\lambda x_1 \cdots x_n.\mathscr{L}(u'))\mathscr{L}(c_1) \cdots \mathscr{L}(c_n) = t$$

that is solving a higher-order matching problem. So parsing with ACGs, that is going from the object language to the abstract language, is related to the complexity of higher-order matching.

This justifies how careful one must be when trying to extend the expressive power of the object language, hence catching more λ-terms. Indeed, if higher-order matching is decidable until the fourth order case (Huet, 1976; Dowek, 1994; Padovani, 1996), from the sixth-order case, higher-order matching modulo β is undecidable (Loader, 2003) whereas matching modulo $\beta\eta$ is still open. The definition of ACGs by de Groote (2001) relies on the decidability and NP-completeness of higher-order linear matching (de Groote, 2000) to ensure parsing with ACG feasibility.

On the other hand, going from the abstract language to the object language (generation with ACGs) is just morphism application. As far as $\mathscr{L}$ is defined on the constants of C_1, the computation of the image of any term of Λ_1 is straightforward. Figure 1 shows an example where both parsing and generation with ACGs could be usefull, if two ACGs have the same abstract language. Note that both parsing and generation are required to cope with traditional parsing (if Λ_2 models strings and Λ_2' models any semantic representation language for instance) or generation.

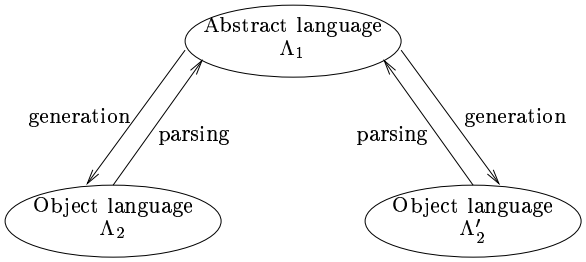


Fig. 1. Moving from an object language to another

Coping both with expressivity and computational complexity while using ACG technologies, we see that there are different (non-exclusive) options:

- doing ACG parsing, we have to remain in a fragment where higher-order matching is decidable;
- doing ACG generation is always easy.

In the following we propose an ACG using non-linear λ-terms, under restrictions we explain. It allows us to model the hole semantics language so that both parsing and generation are available. Then we plug it together with a

string language which model categorial grammars, so that the latter is endowed with an underspecified semantic representation.

2 A Non-Linear Extension

In this section, we extend the definitions of ACGs of de Groote (2001) in order to allow some kind of non linearity in λ-terms, while preserving the computational properties of ACGs. Basically, ACGs are built on intuitionnistic implicative linear logic with the linear implicative type $\multimap$. To reach the intuitionnistic implicative logic, we need the exponential connective ! of linear logic to translate the classical arrow $A \to B$ into $(!A) \multimap B$ (Girard, 1987; Danos and Cosmo, 1992).

Definition 1 (Implicative Types). Let A be a set of atomic types. The set $\mathscr{T}_{\mathscr{L}}(A)$ of *linear implicative types* is defined by the following grammar:

$$\mathscr{T}_{\mathscr{L}}(A) ::= A | \mathscr{T}_{\mathscr{L}}(A) \multimap \mathscr{T}_{\mathscr{L}}(A)$$

The set $\mathscr{T}(A)$ of *implicative types* is defined by the following grammar:

$$\mathscr{T}(A) ::= A | !\mathscr{T}(A) | \mathscr{T}(A) \multimap \mathscr{T}(A)$$

Using the usual right associativity, we note $\alpha \multimap \beta \multimap \gamma \multimap \delta$ for $(\alpha \multimap (\beta \multimap (\gamma \multimap \delta)))$

Definition 2 (Higher-Order Signature). $\Sigma = \langle A, C, \tau \rangle$ is a *higher-order signature* (resp. *higher-order linear signature*), where:

- A is a finite set of atomic types;
- C is a finite set of constants;
- $\tau : C \to \mathscr{T}(A)$ (resp. $\tau : C \to \mathscr{T}_{\mathscr{L}}(A)$) assigns an implicative type to each constant.

Definition 3 (Typed Terms). Let X be an infinite countable set of λ-variables. The set $\Lambda(\Sigma)$ of *λ-terms* built upon a higher-order signature $\Sigma = \langle A, C, \tau \rangle$ is inductively defined as follows:

- if $c \in C$ then $c : \tau(c) \in \Lambda(\Sigma)$;
- if $x \in X$ and $\alpha \in \mathscr{T}(A)$ then $x : \alpha \in \Lambda(\Sigma)$;
- if $x : !\alpha \in \Lambda(\Sigma), t : \beta \in \Lambda(\Sigma)$ and x occurs free at least once in t then $\lambda x.t : !\alpha \multimap \beta \in \Lambda(\Sigma)$;
- if $x : \alpha \in \Lambda(\Sigma), \alpha \neq !\alpha', t : \beta \in \Lambda(\Sigma)$ and x occurs free exactly once in t then $\lambda x.t : \alpha \multimap \beta \in \Lambda(\Sigma)$;

- if $t : \alpha \multimap \beta, u : \alpha \in \Lambda(\Sigma)$ and if $\alpha \neq !\alpha'$ or there is no free linear variable in u, then $(t\, u) \in \Lambda(\Sigma)$.

A term $t : \alpha \in \Lambda(\Sigma)$ is *linear* if $\alpha \in \mathscr{T}_{\mathscr{L}}(A)$.

We now define the extensio of ACGs with linear abstract terms but with general (with no empty abstraction) λ-terms as object terms.

Definition 4 (Lexicon). Let $\Sigma_1 = \langle A_1, C_1, \tau_1 \rangle$ be a higher-order *linear* signature, and $\Sigma_2 = \langle A_2, C_2, \tau_2 \rangle$ be a higher-order signature. A *lexicon* $\mathscr{L}$ from Σ_1 to Σ_2 is a pair $\mathscr{L} = \langle F, G \rangle$ such that:

- $F : A_1 \to \mathscr{T}(A_2)$ is a function interpreting the atomic types of Σ_1 as implicative types built upon A_2. We also call F its homomorphic extension over $\mathscr{T}_{\mathscr{L}}(A_1)$;
- $G : C_1 \to \Lambda(\Sigma_2)$ is a function interpreting the constant of Σ_1 as λ-terms built upon Σ_2, compatible with the typing relation: for any $c \in C_1$, $G(c) : F(\tau_1(c)) \in \Lambda(\Sigma_2)$. We also call G its homomorphic extension over $\Lambda(\Sigma_1)$.

Depending on the context, we note $\mathscr{L}(a)$ either for $F(a)$ or $G(a)$.

Definition 5 (Abstract Categorial Grammar). An *abstract Categorial Grammar* is a quadruple $\mathscr{G} = \langle \Sigma_1, \Sigma_2, \mathscr{L}, S \rangle$ where:

- Σ_1, called the *abstract vocabulary*, is a higher-order *linear* signature, and Σ_2, called the *object vocabulary*, is a higher-order signature;
- $\mathscr{L} : \Sigma_1 \to \Sigma_2$ is a lexicon from the abstract vocabulary to the object vocabulary;
- $S \in A_1$ is called the *distinguished type* of the grammar.

An abstract categorial grammar $\mathscr{G} = \langle \Sigma_1, \Sigma_2, \mathscr{L}, S \rangle$ is *lexicalized* whenever for any $c \in C_1$, there is $c' \in C_2$ such that c' is a subterm of $\mathscr{L}(c)$.

Definition 6 (Abstract and Object Languages). Let $\mathscr{G} = \langle \Sigma_1, \Sigma_2, \mathscr{L}, S \rangle$ be an abstract categorial grammar. The *abstract language* $\mathcal{A}(\mathscr{G})$ and the *object language* $\mathcal{O}(\mathscr{G})$ generated by $\mathscr{G}$ are defined as follows:

$$\mathcal{A}(\mathscr{G}) = \{t \in \Lambda_1 | t : S\} \qquad \mathcal{O}(\mathscr{G}) = \{t \in \Lambda_2 | \exists u \in \mathcal{A}(\mathscr{G})\ t = \mathscr{L}(u)\}$$

Theorem. Let $\mathscr{G}$ be a lexicalized abstract grammar as defined above. Then the parsing problem is decidable.

Proof. As we noted in section 1, the parsing problem is equivalent to solving the equation

$$(\lambda x_1 \cdots x_n . \mathscr{L}(t)) \mathscr{L}(c_1) \cdots \mathscr{L}(c_n) = u$$

for any u in $\Lambda(\Sigma_2)$.

Since there is no empty λ-abstraction in $\Lambda(()\Sigma_1)$ and $\Lambda(\Sigma_2)$ (every abstraction binds at least a free variable), there is no empty λ-abstraction in $\mathscr{L}(t)$ nor in any $\mathscr{L}(c_i)$. Since nothing disappear, any constant c' on the right hand side of the equation must appear at least once on the left hand side, and at most the number of times it appears on the right hand side. Moreover, every $\mathscr{L}(c_i)$ contributes with at least one constant ($\mathscr{G}$ is lexicalized). Thus, the set of possible combinations for $\mathscr{L}(c_1), \ldots, \mathscr{L}(c_n)$, hence for $c_1, \ldots, c_n$, is finite.

Moreover, t is a linear λ-term, so the problem can be seen as a proof-search problem (with additional constraints) in the multiplicative fragment of linear logic: with $c_i : \alpha_i$, we have $\alpha_i \in \mathscr{T}_{\mathscr{L}}(A_1)$ and it reduces to solve $x_i : \alpha_1, \cdots, x_n : \alpha_n \vdash t : S$ which is decidable. Pogodalla (2001), in another context, describes an algorithm to solve such problems. □

This extension may appear as a small extension of basic ACGs. Nevertheless, it's worth noting that :

- it preserves the computational properties of ACGs;
- it now allows to relate linear signatures (for instance for syntax) and non-linear signatures (for instance for semantics);
- it gives for free both parsing and generation (in the usual sense) decidability;
- since the abstract signature is linear, and the object one not necessarily linear, we lose some compositional properties of ACGs: the object vocabulary cannot appear as an abstract one in another ACG, but it can still appear as an object one in another ACG.

We are now in position of using these properties with an underspecified semantic representation language.

3 Example: Using the Hole Semantics

3.1 a Higher-Order Signature for the Hole Semantics

Bos (1995) proposes the hole semantics as a general framework to cope with underspecified representations. Given a logical language, it provides a syntax and a semantics to express underspecification, that is formulas expressing constraints over the formulas of the logical language. It uses the predicate logic "unplugged" as logical language as example of this approach. Its syntax is basically the same as predicate logic, except that, if atomic formulas remain the same, formulas are built from *holes* and *labels*, the latter being used as palce holder for logical formulas in the underspecified representation language.

So that there are two languages: a semantic representation language (SRL), and an underspecified representation language (URL) (Blackburn and Bos, 2003). The latter using formulas of the former in its λ-term representation (Blackburn and Bos, 2003), and both being first order languages, we use boldface symbols (e.g, **All**, **And**, **Imp**, etc.) for the usual symbols in SRL, whereas we exactly use the usual logical symbols ($\exists$, $\wedge$), an infix predicate $\geq$ to specify the constraints and an infix operateur : for URL. The symbol $h \geq l$ somehow imposes the constraint for a formula that is associated to l to be a subformula of the one associated to h. $l : p$ indicates that a predicate p of SRL is labelled in URL by l. But let us have an example.

To the sentence *every man loves some woman* is associated the formula of URL

$$\begin{aligned}\exists h_0 h_1 h_2 l_1 l_2 l_3 l_4 l_5 l_6 l_7 xy(l_1 : \mathbf{All}(x, l_2) \wedge l_2 : \mathbf{Imp}(l_3, h_1) \wedge l_3 : \mathbf{man}(x) \\ \wedge l_4 : \mathbf{Some}(y, l_5) \wedge l_5 : \mathbf{And}(l_6, h_2) \wedge l_6 : \mathbf{woman}(y) \\ \wedge l_7 : \mathbf{love}(x, y) \quad \wedge h_1 \geq l_7 \wedge h_2 \geq l_7 \wedge h_0 \geq l_1 \wedge h_0 \geq l_4)\end{aligned}$$

As suggested by Blackburn and Bos (2003), it can preferably be drawn as in figure 2, representing the $\geq$ relation with dashed lines. We can state the constraints it represents as: $\mathbf{love}(x, y)$ must be a subformula of l_2 in $\mathbf{All}(x, l_2)$, but also a subformula of l_5 in $\mathbf{Some}(y, l_5)$. It represents two formulas of SRL that both satisfy the constraints. In the first formula, l_5 (hence l_4) is a subfornula of l_2 too. In the second one, l_2 (hence l_1) is a subformula of l_3:

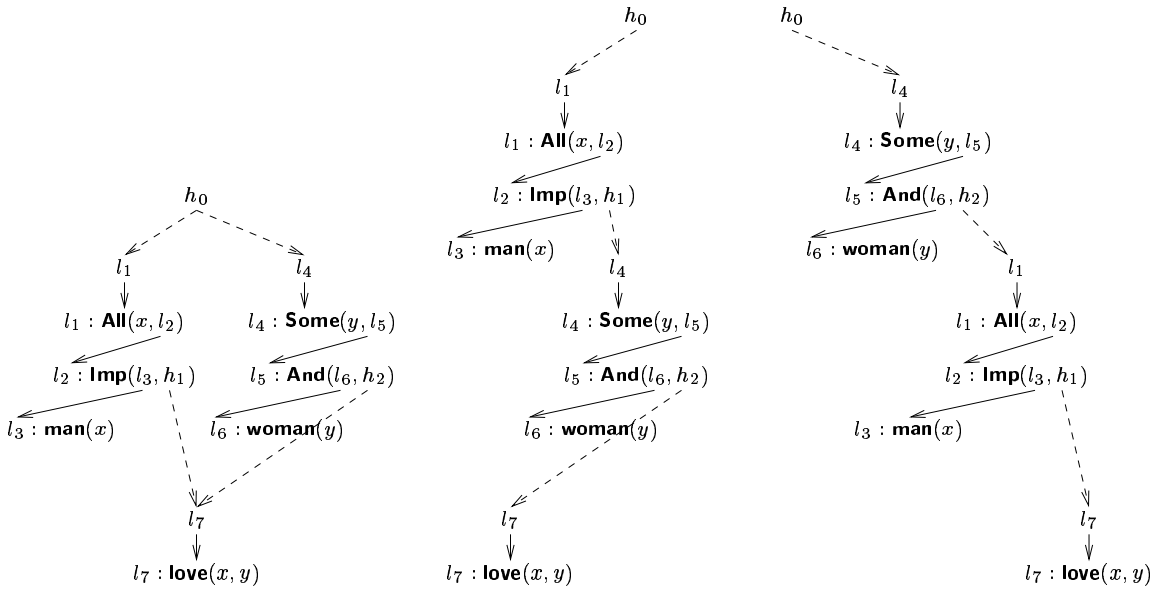
$$\mathbf{All}(x, \mathbf{Imp}(\mathbf{man}(x), \mathbf{Some}(y, \mathbf{And}(\mathbf{woman}(y), \mathbf{love}(x, y))))) \quad (1)$$
$$\mathbf{Some}(y, \mathbf{And}(\mathbf{woman}(y), \mathbf{All}(x, \mathbf{Imp}(\mathbf{man}(x), \mathbf{love}(x, y))))) \quad (2)$$

or, expressed in the usual predicate logic language:

$$\forall x(\mathbf{man}(x) \Rightarrow \exists y(\mathbf{woman}(y) \wedge \mathbf{love}(x, y))) \quad (1)'$$
$$\exists y(\mathbf{woman}(y) \wedge \forall x(\mathbf{man}(x) \Rightarrow \mathbf{love}(x, y))) \quad (2)'$$

We want to underline the difference between URL and SRL because our concern in this paper is not to build and manage SRL formulas, but only URL formulas, that is underspecified representations. So, the object language of the ACG we are designing is URL. Let us now define the higher-order signature $\Sigma_{\text{url}} = \langle A_{\text{url}}, C_{\text{url}}, \tau_{\text{url}} \rangle$:

- $A_{\text{url}} = \{e, h, l, p, t\}$ where e stands for entities, h for holes, l for labels, p for predicate of the logical language and t for truth values;
- $C_{\text{url}} = \{\geq, :, \exists_l, \exists_e, \exists_h, \wedge, \mathbf{Imp}, \mathbf{And}, \mathbf{Some}, \mathbf{All}\} \cup P$ where P is set set of the predicate symbols of the logical language (in the examples, we use $P = \{\mathbf{man}, \mathbf{woman}, \mathbf{love} \ldots\}$);



(a) Underspecified representation

(b) First model

(c) Second model

Fig. 2. Semantic representation of *every man loves some woman*

- τ_{url} is defined in table 1.

Note we have three existential quantifiers $\exists_l$, $\exists_h$ and $\exists_e$, but we usually note them only $\exists$. Moreover, to keep the usual logical notation we write $\exists x\, P$ instead of $\exists(\lambda x.P)$ where x is a free variable of P.

We are now in position to use this higher-order signature in an ACG.

3.2 Example: An ACG that endowes Categorial Grammars with an Underspecified Semantics

We need first to define the abstract vocabulary. We rely on the example de Groote (2001) gives to accomodate categorial grammars with ACGs. First is the abstract vocabulary $\Sigma_{\text{cg}} = \langle A_{\text{cg}}, C_{\text{cg}}, \tau_{\text{cg}} \rangle$:

- $A_{\text{cg}} = \{N, NP, S\}$;
- $C_{\text{cg}} = \{c_{John}, c_{loves}, c_{claims}, c_{man}, c_{woman}, c_{some}, c_{every}, c_{seems}\}$;

$\geq$	$: h \to l \to t$	specifies the underspecification constraints
$:$	$: l \to p \multimap t$	labels the logical predicates
$\wedge$	$: t \multimap t \multimap t$	conjunct of descriptions
$\exists_l$	$: (l \to t) \multimap t$	existential quantifier on labels
$\exists_h$	$: (l \to t) \multimap t$	existential quantifier on holes
$\exists_e$	$: (e \to t) \multimap t$	existential quantifier on entities
And, **Imp**	$: l \to h \to p$	conjonction and implication in the embedded logical language
man, **woman**	$: e \to p$	predicates in the embedded logical language
claims	$: e \to p$	predicate in the embedded logical language
john	$: (e \to t) \to t$	predicate in the embedded logical language
loves	$: e \to e \to p$	predicate in the embedded logical language
F, **S**	$: h \to p$	predicates in the embedded logical language

Table 1
Definition of τ_{url}

- τ_{cg} :

$$
\begin{aligned}
c_{John} &: NP & c_{loves} &: NP \multimap NP \multimap S \\
c_{every} &: N \multimap NP & c_{some} &: N \multimap N \\
c_{claims} &: S \multimap NP \multimap S & c_{man} &: N \\
c_{woman} &: N & c_{seems} &: (NP \multimap S) \multimap NP \multimap S \\
c_{and} &: NP \multimap NP \multimap NP & c_{usually} &: (NP \multimap S) \multimap (NP \multimap S)
\end{aligned}
$$

Then, on the syntactical side, we have the following signature:

$$\Sigma_{\text{str}} = \langle \{\text{str}\}, \{\epsilon, /John/, /loves/, +, \ldots\}, \tau_{\text{str}} \rangle$$

with $\tau_{\text{str}}(+) = \text{str} \multimap \text{str}$ the concatenation operation on strings and ϵ the empty string, and for any other constant c, $\tau_{\text{str}}(c) = \text{str}$, and the expected following lexicon:

$$
\begin{aligned}
\mathscr{L}_{\text{syn}}(John) &= /John/ & \mathscr{L}_{\text{syn}}(loves) &= \lambda xy.y + /loves/ + x \\
\mathscr{L}_{\text{syn}}(every) &= \lambda x./every/ + x & \mathscr{L}_{\text{syn}}(some) &= \lambda x./some/ + x \\
\mathscr{L}_{\text{syn}}(claims) &= \lambda xy.y + /claims/ + x & \mathscr{L}_{\text{syn}}(man) &= /man/ \\
\mathscr{L}_{\text{syn}}(woman) &= /woman/ & \mathscr{L}_{\text{syn}}(seems) &= \lambda rx.x + /seems/ + r/, \epsilon
\end{aligned}
$$

As in the figure 1, we have defined a first ACG to deal with the syntactic analysis. For instance, we have

$$\mathscr{L}_{\text{syn}}(c_{loves}(c_{some}c_{woman})(c_{every}c_{man})) = (/every/+/man/)+/loves/+(/some/+/woman/)$$

so $t = c_{loves}(c_{some}c_{woman})(c_{every}c_{man})$ is the abstract term corresponding to the expression *every man loves some woman.*

But let us now keep drawing the figure adding another ACG to deal with the semantic side.

We can define $\mathscr{L}_{\text{sem}} : \Sigma_{\text{cg}} \to \Sigma_{\text{url}}$ with the following:

$$\mathscr{L}_{\text{sem}}(NP) = (e \to h \to l \to t) \to (h \to l \to t) \qquad \mathscr{L}_{\text{sem}}(S) = h \to l \to t$$
$$\mathscr{L}_{\text{sem}}(N) = e \to h \to l \to t$$

$$\begin{aligned}
\mathscr{L}_{\text{sem}}(c_{John}) &= \lambda P.P\ \textbf{john}\\
\mathscr{L}_{\text{sem}}(c_{loves}) &= \lambda os.s(\lambda x.o(\lambda yhl.h \geq l \wedge l : \textbf{loves}(x,y)))\\
\mathscr{L}_{\text{sem}}(c_{every}) &= \lambda xyhl.\exists h_1 l_1 l_2 l_3 v_1 (h \geq l_2 \wedge l_2 : \textbf{All}(v_1, l_3) \wedge l_3 : \textbf{Imp}(l_1, h_1)\\
&\quad \wedge h_1 \geq l \wedge x\, v_1\, h\, l_1 \wedge y\, v_1\, h\, l)\\
\mathscr{L}_{\text{sem}}(c_{some}) &= \lambda xyhl.\exists h'_1 l'_1 l'_2 l'_3 v'_1 (h \geq l'_2 \wedge l'_2 : \textbf{Some}(v'_1, l'_3) \wedge l'_3 : \textbf{And}(l'_1, h'_1)\\
&\quad \wedge h'_1 \geq l \wedge x\, v'_1\, h\, l'_1 \wedge y\, v'_1\, h\, l)\\
\mathscr{L}_{\text{sem}}(c_{man}) &= \lambda vhl.h \geq l \wedge l : \textbf{man}(v)\\
\mathscr{L}_{\text{sem}}(c_{woman}) &= \lambda vhl.h \geq l \wedge l : \textbf{woman}(v)\\
\mathscr{L}_{\text{sem}}(c_{and}) &= \lambda PQ.\lambda rhl.P\, r\, h\, l \wedge Q\, r\, h\, l
\end{aligned}$$

Let $\mathscr{G}_{\text{sem}} = \langle \Sigma_{\text{cg}}, \Sigma_{\text{url}}, \mathscr{L}_{\text{sem}}, S \rangle$. We have that

$$\begin{aligned}
\mathscr{L}_{\text{sem}}(c_{some}c_{woman}) &= \lambda yhl.\exists h'_1 l'_1 l'_2 l'_3 v'_1 (h \geq l'_2 \wedge l'_2 : \textbf{Some}(v'_1, l'_3) \wedge l'_3 : \textbf{And}(l'_1, h'_1)\\
&\quad \wedge h'_1 \geq l \wedge h \geq l'_1 \wedge l'_1 : \textbf{woman}(v'_1) \wedge y\, v'_1\, h\, l)\\
\mathscr{L}_{\text{sem}}(c_{every}c_{man}) &= \lambda yhl.\exists h_1 l_1 l_2 l_3 v_1 (h \geq l_2 \wedge l_2 : \textbf{All}(v_1, l_3) \wedge l_3 : \textbf{Imp}(l_1, h_1)\\
&\quad \wedge h_1 \geq l \wedge h \geq l_1 \wedge l_1 : \textbf{man}(v_1) \wedge y\, v_1\, h\, l)\\
\mathscr{L}_{\text{sem}}(c_{loves}(c_{some}c_{woman})) &= \lambda s.s(\lambda xhl.\exists h'_1 l'_1 l'_2 l'_3 v'_1 (h \geq l'_2 \wedge l'_2 : \textbf{Some}(v'_1, l'_3) \wedge l'_3 : \textbf{And}(l'_1, h'_1)\\
&\quad \wedge h'_1 \geq l \wedge h \geq l'_1 \wedge l'_1 : \textbf{woman}(v'_1) \wedge h \geq l \wedge l : \textbf{loves}(x, v'1)))
\end{aligned}$$

Then, with the previous definition of t

$$\begin{aligned}
\mathscr{L}_{\text{sem}}(t) = \lambda hl.&\exists h_1 l_1 l_2 l_3 v_1 (h \geq l_2 \wedge l_2 : \textbf{All}(v_1, l_3) \wedge l_3 : \textbf{Imp}(l_1, h_1)\\
&\wedge h_1 \geq l \wedge h \geq l_1 \wedge l_1 : \textbf{man}(v_1)\\
&\wedge \exists h'_1 l'_1 l'_2 l'_3 v'_1 (h \geq l'_2 \wedge l'_2 : \textbf{Some}(v'_1, l'_3) \wedge l'_3 : \textbf{And}(l'_1, h'_1)\\
&\wedge h'_1 \geq l \wedge h \geq l'_1 \wedge l'_1 : \textbf{woman}(v'_1) \wedge h \geq l \wedge l : \textbf{loves}(v_1, v'1)))
\end{aligned}$$

The latter formula correspond to the one expected and represented in figure 2(a) (modulo variable renaming). So the abstract term t can be send on

every man loves some woman on the syntactic side, thanks to $\mathscr{L}_{\text{syn}}$, and on the representation above on the semantic side, thanks to $\mathscr{L}_{\text{sem}}$. Moreover, with this approach, the semantical requirement of type raising do not interact anymore with the syntactic role of determiners.

It remains true with conjonction.For instance we can compute:

$$\begin{aligned}\mathscr{L}_{\text{sem}}(c_{and}c_{John}(c_{some}c_{man})) &= (\lambda PQ.\lambda rhl.P\,r\,h\,l \wedge Q\,r\,h\,l)(\lambda P.P\ \mathbf{j})(\lambda yhl.\\ &\quad \exists h_1' l_1' l_2' l_3' v_1' (h \geq l_2' \wedge l_2' : \mathbf{Some}(v_1', l_3') \wedge l_3' : \mathbf{And}(l_1', h_1')\\ &\quad \wedge h_1' \geq l \wedge h \geq l_1' \wedge l_1' : \mathbf{man}(v_1') \wedge y\,v_1'\,h\,l))\\ &= (\lambda Q.\lambda rhl.r\ \mathbf{j}\ h\,l \wedge Q\,r\,h\,l)(\lambda yhl.\\ &\quad \exists h_1' l_1' l_2' l_3' v_1' (h \geq l_2' \wedge l_2' : \mathbf{Some}(v_1', l_3') \wedge l_3' : \mathbf{And}(l_1', h_1')\\ &\quad \wedge h_1' \geq l \wedge h \geq l_1' \wedge l_1' : \mathbf{man}(v_1') \wedge y\,v_1'\,h\,l))\\ &= \lambda rhl.r\ \mathbf{j}\ h\,l \wedge \exists h_1' l_1' l_2' l_3' v_1' (h \geq l_2' \wedge l_2' : \mathbf{Some}(v_1', l_3')\\ &\quad \wedge l_3' : \mathbf{And}(l_1', h_1') \wedge h_1' \geq l \wedge h \geq l_1' \wedge l_1' : \mathbf{man}(v_1') \wedge r\,v_1'\,h\,l)\end{aligned}$$

Then

$$\begin{aligned}\mathscr{L}_{\text{sem}}(c_{loves}c_{Mary}(c_{and}c_{John}(c_{some}c_{man}))) &= \lambda hl.h \geq j \wedge l : loves(\mathbf{j}, \mathbf{m}) \wedge \exists h_1' l_1' l_2' l_3' v_1' (h \geq l_2'\\ &\quad \wedge l_2' : \mathbf{Some}(v_1', l_3') \wedge l_3' : \mathbf{And}(l_1', h_1')\\ &\quad \wedge h_1' \geq l \wedge h \geq l_1' \wedge l_1' : \mathbf{man}(v_1') \wedge h \geq l\\ &\quad \wedge l : \mathbf{loves}(v'1, \mathbf{m})))\end{aligned}$$

We can also extend the lexicon to deal with control and raising verbs:

$$\begin{aligned}\mathscr{L}_{\text{sem}}(c_{claims}) &= \lambda rs.s(\lambda xhl.\exists l_1 h_1 (h \geq l_1 \wedge l_1 : \mathbf{claims}(x, h_1) \wedge r h_1 l))\\ \mathscr{L}_{\text{sem}}(c_{seems}) &= \lambda rs.s(\lambda xhl.\exists h_1 l_1 (r\,x\,h\,l \wedge h \geq l_1 \wedge l_1 : \mathbf{seems}(h_1) \wedge h_1 \geq l)\\ \mathscr{L}_{\text{sem}}(c_{tries}) &= \lambda rs.s(\lambda xhl.\exists h_1 l_1 (r\,x\,h\,l \wedge h \geq l_1 \wedge l_1 : \mathbf{tries}(x, h_1) \wedge h_1 \geq l)\end{aligned}$$

Or intersective or non subsective adjectives:

$$\begin{aligned}\mathscr{L}_{\text{sem}}(c_{black}) &= \lambda n.\lambda xhl.n\,x\,h\,l \wedge l : \mathbf{black}(x)\\ \mathscr{L}_{\text{sem}}(c_{former}) &= \lambda n.\lambda xhl.\exists h_1 l_1 (h \geq l_1 \wedge l_1 : \mathbf{former}(h_1) \wedge n\,x\,h_1\,l)\end{aligned}$$

More interesting is the case of adverbs. They also can create ambiguities, such as in *John usually loves some woman*,depending on the scope of *usually* and the scope of *some woman*. It is the same in *John alledgedly usually loves some woman*, except that *alledgedly* always scopes over *usually*.

$$\begin{aligned}\mathscr{L}_{\text{sem}}(c_{usually}) &= \lambda r.s.\lambda hl.\exists h_1 l_1 (h \geq l \wedge l : \mathbf{usually}(h_1) \wedge h_1 \geq l_1 \wedge r\,s\,h\,l_1)\\ \mathscr{L}_{\text{sem}}(c_{alledgedly}) &= \lambda r.s.\lambda hl.\exists h_1 l_1 (h \geq l \wedge l : \mathbf{alledgedly}(h_1) \wedge h_1 \geq l_1 \wedge r\,s\,h\,l_1)\\ \mathscr{L}_{\text{sem}}(c_{intentionally}) &= \lambda r.s.\lambda hl.\exists h_1 l_1 (h \geq l_1 \wedge l_1 : \mathbf{intentionally}(h_1) \wedge h_1 \geq l \wedge r\,s\,h\,l)\end{aligned}$$

We have for instance:

$$\begin{aligned}\mathscr{L}_{\text{sem}}(c_{usually}(c_{loves}(c_{some}c_{woman})) = \lambda shl.\exists h_1 l_1 (h \geq l \wedge l : \textbf{usually}(h_1) \wedge h_1 \geq l_1 \\ \wedge s(\lambda x h l.\exists h'_1 l'_1 l'_2 l'_3 v'_1 (h \geq l'_2 \wedge l'_2 : \textbf{Some}(v'_1, l'_3) \\ \wedge \textbf{And}(l'_1, h'_1) \wedge h'_1 \geq l \wedge h \geq l'_1 \wedge l'_1 : \textbf{woman}(v'_1) \\ \wedge h \geq l \wedge l : \textbf{loves}(y, v'_1)))hl_1)\end{aligned}$$

It appears that **loves** has to be both a subformula of **usually** and a subformula of **Some**. This will remain, even if the l label to **usually** imposes that another adverb, such as **alledgedly**, to make it a subformula. So that the ambiguities with the existential quantifier are preserved, but **usually** always remains in the scope of any other adverb.

On the other hand, an adverb such as **intentionnaly** would preserve all the combinations, not specifying that it has to be a subformula of any other adverb: its label l_1 is specified in the lexical entry.

Conclusion

We have presented the different parameters that give ACGs their computational properties. We have seen that we can manage expressivity and reversibility, depending on the aim we have. For instance, since lexicon application is always easy, an object language from which no parsing has to be made can be very broad. On the other hand, whenever we want to keep parsing from the object language to the abstract language, we have to be careful with the language we use.

It helped us to define an extension that keeps the reversibility property of ACGs. This extension is very important when dealing with semantic representation languages, because it can manage non-linearity of terms. As an illustration, we described an ACG that can endow catagorial grammars with an underspecified semantic representation: the semantic ambiguities are not anymore modeled at the syntactic level, but at the semantic one. On the other hand, because the abstract and the object vocabularies now differ, we lose some kind of compositionality of ACGs.

Finally, because of the compositional aspects of ACGs, this extension can be used with other syntactical formalisms that have been encoded with ACGs, in order to provide them with an explicit compositional semantics, as done in (Pogodalla, 2004) for TAGs. Of course, it is not limited to underspecified semantics, and other semantic representation languages could be encoded since we now provide non-linearity.

References

Blackburn, P., Bos, J., 2003. Computational semantics for natural language. http://www.iccs.informatics.ed.ac.uk/ jbos/comsem/book1.html, course Notes for NASSLLI 2003.

Bos, J., 1995. Predicate logic unplugged. In: Proceedings of the Tenth Amsterdam Colloquium.

Danos, V., Cosmo, R. D., 1992. The linear logic primer. http://www.pps.jussieu.fr/ dicosmo/CourseNotes/LinLog/, an introductory course on Linear Logic.

de Groote, P., 2000. Linear higher-order matching is np-complete. In: Bachmair, L. (Ed.), Rewriting Techniques and Applications, RTA'00. Vol. 1833 of LNCS. Springer, pp. 127–140.

de Groote, P., 2001. Towards abstract categorial grammars. In: Association for Computational Linguistics, 39th Annual Meeting and 10th Conference of the European Chapter, Proceedings of the Conference. pp. 148–155.

de Groote, P., 2002. Tree-adjoining grammars as abstract categorial grammars. In: TAG+6, Proceedings of the sixth International Workshop on Tree Adjoining Grammars and Related Frameworks. Università di Venezia, pp. 145–150.

de Groote, P., Pogodalla, S., 2003. m-linear context-free rewriting systems as abstract categorial grammars. In: Oehrle, R., Rogers, J. (Eds.), MOL 8, proceedings of the eighth Mathematics of Language Conference.

Dowek, G., 1994. Third order matching is decidable. Annals of Pure and Applied Logic 69 (2–3), 135–155.

Girard, J.-Y., 1987. Linear logic. Theoretical Computer Science 50, 1–102.

Huet, G. P., 1976. Résolution d'équations dans les langages d'ordre 1, 2, ..., ω. Ph.D. thesis, Université Paris, 7.

Loader, R., 2003. Higher order β matching is undecidable. Logic Journal of the IGPL 11 (1), 51–68.

Padovani, V., 1996. Filtrage d'ordre supérieur. Ph.D. thesis, Université de Paris 7.

Pogodalla, S., 2001. Réseaux de preuve et génération pour les grammaires de types logiques. Ph.D. thesis, Institut National Polytechnique de Lorraine. URL http://www.loria.fr/publications/2001/A01-T-422/A01-T-422.ps

Pogodalla, S., May 2004. Computing semantic representation: Towards ACG abstract terms as derivation trees. In: Proceedings of the Seventh International Workshop on Tree Adjoining Grammar and Related Formalisms (TAG+7).