Generalizing a Proof-Theoretic Account of Scope Ambiguity

Sylvain Pogodalla

Generalizing a Proof-Theoretic Account of Scope Ambiguity

Sylvain Pogodalla (LORIA, INRIA Lorraine)

1 Semantic Ambiguity in Natural Language

When trying to build the semantic representation of a natural language expression, it may happen that a single expression produces many semantic representations. In this paper, we focus on scope ambiguities where a single syntactic analysis can yield many semantic representations. This kind of ambiguity can occur with quantified noun phrases (*every*, *some*, *most*, etc.) but also with adverbs and *how-many* questions, etc.

There are basically two ways to address scope ambiguities. One way is to build two syntactic structures (parse trees) from a single expression, then, from these syntactic structures, to functionally build two semantic representations. The other way is to build a single syntactic structure and to associate to the latter, in a non-functional way, two semantic representations. These two ways are represented by two frameworks: the type-logical framework, where ambiguity is modeled by the process (proof search), and the underspecification framework, where ambiguity is modeled by a (formal) language.

Our proposal aims at giving an account of scope ambiguity that does not rely on syntactic ambiguity nor on another intermediate language. It is based on the Abstract Categorial Grammar (ACG) framework [dG01].

2 Abstract Categorial Grammars

The main feature of an ACG is to 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. This general picture can of course be adapted to the need of the modeling. In order to be able to model non linearity (this is useful for semantics), we use an extension of the

ACG with both *linear* and *non-linear* implications but the principles follow [dG01]'s definitions.[1]

Definition 1. *Let A be a set of atomic types. The set $\mathcal{T}(A)$ of* implicative types *build upon A is defined with the following grammar:*

$$\mathcal{T}(A) ::= A | \mathcal{T}(A) \multimap \mathcal{T}(A) | \mathcal{T}(A) \to \mathcal{T}(A)$$

Definition 2. *A* higher-order signature Σ *is a triple $\Sigma = \langle A, C, \tau \rangle$ where A is a finite set of atomic types, C is a finite set of constants and $\tau : C \to \mathcal{T}(A)$ is a function assigning types to constants.*

Definition 3. *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 in a standard way, using the constants of C and the variable of X. The main point is that it introduces a linear abstraction $\lambda^0 x.t$ (if x occurs free in t exactly once) in addition to the usual intuitionistic abstraction ($\lambda x.t$ if x occurs free in t). There also are the usual notions of α-conversion and β-reduction.*

Given a higher-order signature Σ, the typing rules are given with an inference system whose judgments are of the following form: $\Gamma; \Delta \vdash_\Sigma t : \alpha$ where Γ is a finite set of non-linear variable typing declarations and Δ is a finite set of linear variable typing declarations. Both Γ and Δ may be empty. Here are the typing rules:

$$\frac{}{\Gamma; \vdash_\Sigma c : \tau(c)}\ \text{const.}$$

$$\frac{}{\Gamma; x : \alpha \vdash_\Sigma x : \alpha}\ \text{lin. var.} \qquad \frac{}{\Gamma, x : \alpha; \vdash_\Sigma x : \alpha}\ \text{var.}$$

$$\frac{\Gamma; \Delta, x : \alpha \vdash_\Sigma t : \beta}{\Gamma; \Delta \vdash_\Sigma \lambda^0 x.t : \alpha \multimap \beta}\ \text{l. abs.} \qquad \frac{\Gamma; \Delta_1 \vdash_\Sigma t : \alpha \multimap \beta \quad \Gamma; \Delta_2 \vdash_\Sigma u : \alpha}{\Gamma; \Delta_1, \Delta_2 \vdash_\Sigma (t\, u) : \beta}\ \text{l. app.}$$

$$\frac{\Gamma, x : \alpha; \Delta \vdash_\Sigma t : \beta}{\Gamma; \Delta \vdash_\Sigma \lambda x.t : \alpha \to \beta}\ \text{abs.} \qquad \frac{\Gamma; \Delta \vdash_\Sigma t : \alpha \to \beta \qquad \Gamma; \vdash_\Sigma u : \alpha}{\Gamma; \Delta \vdash_\Sigma (t\, u) : \beta}\ \text{app.}$$

Definition 4. *Let $\Sigma_1 = \langle A_1, C_1, \tau_1 \rangle$ and $\Sigma_2 = \langle A_2, C_2, \tau_2 \rangle$ be two higher-order signatures, a lexicon $\mathcal{L} = \langle F, G \rangle$ from Σ_1 to Σ_2 is such that:*

[1] Formal properties of this extension, as expressiveness and computational properties, are beyond the scope of this paper.

- $F : A_1 \to \mathcal{T}(A_2)$. *We also note* $F : \mathcal{T}(A_1) \to \mathcal{T}(A_2)$ *its homomorphic extension*[2]*;*
- $G : C_1 \to \Lambda(\Sigma_2)$. *We also note* $G : \Lambda(\Sigma_1) \to \Lambda(\Sigma_2)$ *its homomorphic extension*[3]*;*
- F *and* G *are such that for all* $c \in C_1$, $\vdash_{\Sigma_2} G(c) : F(\tau_1(c))$ *is provable.*

We also use $\mathcal{L}$ *instead of* F *or* G*.*

Definition 5. *An* abstract categorial grammar $\mathcal{G}$ *is defined by a quadruple* $\mathcal{G} = \langle \Sigma_1, \Sigma_2, \mathcal{L}, s \rangle$ *where* $\Sigma_1 = \langle A_1, C_1, \tau_1 \rangle$ *and* $\Sigma_2 = \langle A_2, C_2, \tau_2 \rangle$ *are two higher-order signatures,* $\mathcal{L} : \Sigma_1 \to \Sigma_2$ *is a lexicon and* $s \in \mathcal{T}(A_1)$ *is the* distinguished type *of the grammar.*

Definition 6. *Given an ACG* $\mathcal{G}$*, the* abstract language *is defined by* $\mathcal{A}(\mathcal{G}) = \{t \in \Lambda(\Sigma_1) \mid \vdash_{\Sigma_1} t : s \text{ is derivable}\}$*. The* object language *is defined by* $\mathcal{O}(\mathcal{G}) = \{u \in \Lambda(\Sigma_2) \mid \exists t \in \mathcal{A}(\mathcal{G}) \text{ s.t. } u = \mathcal{L}(t)\}$

Note that $\mathcal{L}$ binds the parse structures of $\mathcal{A}(\mathcal{G})$ to the concrete expressions of $\mathcal{O}(\mathcal{G})$. Depending on the choice of Σ_1, Σ_2 and $\mathcal{L}$, it can map for instance derivation trees of CFG, TAG or m-linear context-free rewriting systems and strings of the generated language [dGP04]. It can also apply to semantic formalisms [Pog04] and pragmatic modeling [dG06].

A crucial point is that ACG can be mixed in different ways: in a transversal way, were two ACG use the same abstract language, or in a compositional way, were the abstract language of an ACG is the object language of an other one (Figure 1 illustrates these compositions between three ACG). This paper exemplifies both these usages. It is also at the heart of our proposal.

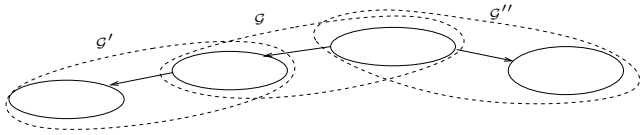

Figure 1: Ways of combining ACG

(1) Every man loves some woman

[2]Such that $F(\alpha \multimap \beta) = F(\alpha) \multimap F(\beta)$ and $F(\alpha \to \beta) = F(\alpha) \to F(\beta)$
[3]Such that $G(\lambda x.t) = \lambda x.G(t)$, etc.

Now, let us run into an example. Here is how scope ambiguities would be modeled in the standard way by categorial grammars. We first define the syntactic part, then the language of strings[4].

$$\Sigma_{\text{synt}} = \begin{cases} A_{\text{synt}} & = \{np, n, s\} \\ C_{\text{every}} & : n \multimap ((np \multimap s) \multimap s) \qquad C_{\text{man}} : n \\ C_{\text{some}} & : n \multimap ((np \multimap s) \multimap s) \qquad C_{\text{woman}} : n \\ C_{\text{love}} & : np \multimap np \multimap s \end{cases}$$

$$\Sigma_{\text{string}} = \begin{cases} A_{\text{string}} = \{\textsc{String}\} \\ \textit{every} : \textsc{String} \quad \textit{woman} : \textsc{String} \quad \textit{man} : \textsc{String} \\ \textit{loves} : \textsc{String} \quad \textit{some} : \textsc{String} \quad \epsilon : \textsc{String} \\ \qquad\qquad\qquad\qquad\qquad + : \textsc{String} \multimap \textsc{String} \end{cases}$$

And finally the lexicon.

$$\mathcal{L}^0_{\text{syntax}} = \begin{cases} n := \textsc{String} \quad np := \textsc{String} \quad s := \textsc{String} \\ C_{\text{every}} := \lambda^0 xR.R(\textit{every} + x) \qquad C_{\text{man}} := \textit{man} \\ C_{\text{some}} := \lambda^0 xR.R(\textit{some} + x) \qquad C_{\text{woman}} := \textit{woman} \\ C_{\text{love}} := \lambda^0 xy.x + \textit{loves} + y \end{cases}$$

Let $\mathcal{G}^0_{\text{syntax}} = \langle \Sigma_{\text{synt}}, \Sigma_{\text{string}}, \mathcal{L}^0_{\text{syntax}}, s \rangle$ be an ACG. Does the sentence (1) belong to $\mathcal{O}(\mathcal{G}^0_{\text{syntax}})$? It amounts to find $t \in \mathcal{A}(\mathcal{G}^0_{\text{syntax}})$ such that $\mathcal{L}^0_{\text{syntax}}(t) = \textit{every} + \textit{man} + \textit{loves} + \textit{some} + \textit{woman}$[5]. There are two such terms: $t_1 = (C_{\text{every}} C_{\text{man}})(\lambda^0 x.(C_{\text{some}} C_{\text{woman}})(\lambda^0 y.C_{\text{love}}\, x\, y))$ and a second term t_2 such that $t_2 = (C_{\text{some}} C_{\text{woman}})(\lambda^0 y.(C_{\text{every}} C_{\text{man}})(\lambda^0 x.C_{\text{love}}\, x\, y))$. Indeed we have:

$$\begin{aligned} \mathcal{L}^0_{\text{syntax}}(t_1) &= (\lambda^0 R.R(\textit{every} + \textit{man})) \\ &\qquad (\lambda^0 x.(\lambda^0 Q.Q(\textit{some} + \textit{woman}))(\lambda^0 y.x + \textit{loves} + y)) \\ &= \textit{every} + \textit{man} + \textit{loves} + \textit{some} + \textit{woman} \\ \mathcal{L}^0_{\text{syntax}}(t_2) &= (\lambda^0 R.R(\textit{some} + \textit{woman})) \\ &\qquad (\lambda^0 y.(\lambda^0 Q.Q(\textit{every} + \textit{man}))(\lambda^0 x.x + \textit{loves} + y)) \\ &= \textit{every} + \textit{man} + \textit{loves} + \textit{some} + \textit{woman} \end{aligned}$$

It is clear that we essentially get there the standard modeling of the type-logical approaches (be it in the Lambek calculus style or with the $\Uparrow$ binder of [Moo91], see [Mor94, p. 153] or [Car97, p. 224]) that depends

[4] ϵ represents the empty string.

[5] In the very general case, this problem, known as *ACG parsing* is not decidable. However, some restrictions (as linearity, but other ones too that are not linear) make it decidable (and polynomial sometime). For such discussions, see [Sal05, Sal06, Yos06].

on the order in which abstractions occur: either y is first abstracted from $C_{\text{love}}\, x\, y$ and the quantified object applies to it, then x is abstracted and the quantified subject applies to it, or they are abstracted in the reverse order.

Now, we can look at the semantic part. It will rely on the same abstract language. However, we need a higher-order signature for the semantic representation. So we first define Σ_{sem} as

$$\Sigma_{\text{sem}} = \begin{cases} A_{\text{sem}} & = \{e, t\} \\ \forall & : (e \to t) \multimap t \quad \exists \quad : (e \to t) \multimap t \quad \textsf{man} \quad : e \multimap t \\ \Rightarrow & : t \multimap t \multimap t \quad \wedge \quad : t \multimap t \multimap t \quad \textsf{woman} \quad : e \multimap t \\ \neg & : t \multimap t \qquad\qquad\qquad\qquad\quad \textsf{love} \quad : e \multimap e \multimap t \end{cases}$$

Then the lexicon[6]:

$$\mathcal{L}_{\text{sem}} = \begin{cases} n := e \multimap t \quad np := e \quad s := t \\ C_{\text{every}} := \lambda^0 PQ.\forall x.(P\, x \Rightarrow Q\, x) \qquad C_{\text{man}} := \textsf{man} \\ C_{\text{some}} := \lambda^0 PQ.\exists x.(P\, x \wedge Q\, x) \qquad C_{\text{woman}} := \textsf{woman} \\ C_{\text{love}} := \textsf{love} \end{cases}$$

We can now define the ACG $\mathcal{G}_{\text{sem}} = \langle \Sigma_{\text{synt}}, \Sigma_{\text{sem}}, \mathcal{L}_{\text{sem}}, s \rangle$. And we have the following two readings:

$$\begin{aligned} \mathcal{L}_{\text{sem}}(t_1) &= (\lambda^0 Q.\forall x.\textsf{man}\, x \Rightarrow Qx) \\ &\qquad (\lambda^0 x.(\lambda^0 Q.\exists y.(\textsf{woman}\, y \wedge Q\, y))(\lambda y^0.\textsf{love}\, x\, y)) \\ &= \forall x.\textsf{man}\, x \Rightarrow \exists y.(\textsf{woman}\, y \wedge \textsf{love}\, x\, y) \end{aligned}$$

$$\begin{aligned} \mathcal{L}_{\text{sem}}(t_2) &= (\lambda^0 Q.\exists y.(\textsf{woman}\, x \wedge Qx)) \\ &\qquad (\lambda^0 y.(\lambda^0 Q.\forall x.\textsf{man}\, y \Rightarrow Q\, y)(\lambda^0 x.\textsf{love}\, x\, y)) \\ &= \exists y.(\textsf{woman}\, y \wedge \forall x.\textsf{man}\, x \Rightarrow \textsf{love}\, x\, y) \end{aligned}$$

This example shows four things:

- how ACG transfer structures by way of sharing the abstract language (the string expression and the semantic representations share the same structures, namely t_1 and t_2);
- how type-logical approaches model scope ambiguity (with higher-order types and different possible orders for the derivation rules);
- that as soon as the semantic transfer from the syntactic structure is functional (here by the lexicon), semantic ambiguity can only occur if there is some syntactic ambiguity;

[6] We use the usual notation $\forall x.P$ instead of $\forall(\lambda x.P)$.

- that parsing requires both inverting a lexicon (here $\mathcal{L}^0_{\text{syntax}}$) and applying another one (here $\mathcal{L}_{\text{sem}}$).

This last remark is crucial for our proposal: in order to encode a non-functional relation, as the one exemplified between syntactic structure and semantic representation, we need to compose at least two ACG that share a same abstract language. Hence, we get a composition model like in Figure 1 where $\mathcal{G}'$ builds the syntactic structure from strings, where $\mathcal{G}''$ builds the semantic representation from a new kind of structure which is related to the syntactic structure by $\mathcal{G}$. The next section describes our proposal based on that idea.

3 Encoding a Non-Functional Relation

The first step is to design an ACG that will model the relation between parse structures and string expressions, and more precisely to define its abstract signature. The requirement that quantifiers do not entail ambiguity at that level imposes they have not a higher-order type any more. This is the main difference with the signature Σ_{synt} we previously defined. In the new signature Σ_{syntax}, c_{every} has now the expected type $n \multimap np$, which is not higher-order. Note we don't change the set of atomic types.

$$\Sigma_{\text{syntax}} = \begin{cases} A_{\text{synt}} & \\ c_{\text{every}} : n \multimap np & c_{\text{man}} : n \\ c_{\text{some}} : n \multimap np & c_{\text{woman}} : n \\ c_{\text{love}} : np \multimap np \multimap s & \end{cases}$$

$$\mathcal{L}_{\text{syntax}} = \begin{cases} n := \textsc{String} \quad np := \textsc{String} \quad s := \textsc{String} & \\ c_{\text{every}} := \lambda^0 x.every + x & c_{\text{man}} := man \\ c_{\text{some}} := \lambda^0 x.some + x & c_{\text{woman}} := woman \\ c_{\text{love}} := \lambda^0 xy.x + loves + y & \end{cases}$$

Let $\mathcal{G}_{\text{syntax}} = \langle \Sigma_{\text{syntax}}, \Sigma_{\text{string}}, \mathcal{L}_{\text{syntax}}, s \rangle$ be an ACG. Contrary to the previous example, there now is a unique $t_0 \in \mathcal{A}(\mathcal{G}_{\text{syntax}})$ such that $\mathcal{L}_{\text{syntax}}(t_0) = every{+}man{+}loves{+}some{+}woman$. And $t_0 = c_{\text{love}}(c_{\text{every}}c_{\text{man}})(c_{\text{some}}c_{\text{woman}})$.

In order to make ambiguity appear, we need another ACG $\mathcal{G}_{\text{amb}}$ whose object signature is the abstract signature of $\mathcal{G}_{\text{syntax}}$. As abstract signature for $\mathcal{G}_{\text{amb}}$, we simply use Σ_{synt}. The key part is in the following lexicon:

$$\mathcal{L}_{\text{amb}} = \begin{cases} n := n \quad np := np \quad s := s & \\ C_{\text{every}} := \lambda^0 xR.R(c_{\text{every}}x) & C_{\text{man}} := c_{\text{man}} \\ C_{\text{some}} := \lambda^0 xR.R(c_{\text{some}}x) & C_{\text{woman}} := c_{\text{woman}} \\ C_{\text{love}} := c_{\text{loves}} & \end{cases}$$

Note that only the constants dedicated to model quantifiers are changed (they get a higher-order type). With $\mathcal{G}_{\text{amb}} = \langle \Sigma_{\text{synt}}, \Sigma_{\text{syntax}}, \mathcal{L}_{\text{amb}}, s \rangle$, we have $t_0 \in \mathcal{O}(\mathcal{G}_{\text{amb}})$ because $\mathcal{L}_{\text{amb}}(t_1) = \mathcal{L}_{\text{amb}}(t_2) = t_0$.

With $\mathcal{G}_{\text{sem}}$ unchanged, we are now able to associate to the expression (1) a single syntactic structure (namely t_0) and two semantic representations (namely $\mathcal{L}_{\text{amb}}(t_1)$ and $\mathcal{L}_{\text{amb}}(t_2)$). Pushing the higher-order type requirement apart from the syntactic side allows us to avoid the use of a special type constructor, such as $\Uparrow$, hence the need for the corresponding introduction and elimination rules.

Conjunction of quantified and not quantified NPs, whose types differ only in Σ_{synt}, would yield the same type raising of the not quantified NP as in the standard type-logical approach. The next assignments exemplifies this for the expression:

(2) John and every kid ran

$$
\begin{array}{llllll}
C_{\text{run}} & : np \multimap s & C_{\text{kid}} & : n & C_{\text{John}} & : np \\
C_{\text{and}} & : ((np \multimap s) \multimap s) \multimap ((np \multimap s) \multimap s) \multimap (np \multimap s) \multimap s & & & & \\
c_{\text{run}} & : np \multimap s & c_{\text{kid}} & : n & c_{\text{John}} & : np \\
c_{\text{and}} & : np \multimap np \multimap np & & & &
\end{array}
$$

$$
\mathcal{L}_{\text{amb}} = \left\{ \begin{array}{llllll}
C_{\text{run}} & := c_{\text{run}} & C_{\text{kid}} & := c_{\text{kid}} & C_{\text{John}} & := c_{\text{John}} \\
C_{\text{and}} & := \lambda^0 PQR.P(\lambda^0 x.Q(\lambda^0 y.R(c_{\text{and}}\, x\, y)))
\end{array} \right.
$$

$$
\mathcal{L}_{\text{sem}} = \left\{ \begin{array}{llllll}
C_{\text{run}} & := \mathsf{run} & C_{\text{kid}} & := \mathsf{kid} & C_{\text{John}} & := \mathsf{j} \\
C_{\text{and}} & := \lambda^0 PQ.\lambda R.(PR) \wedge (QR)
\end{array} \right.
$$

$$
\mathcal{L}_{\text{syntax}} = \left\{ \begin{array}{llllll}
c_{\text{run}} & := ran & c_{\text{kid}} & := kid & c_{\text{John}} & := John \\
c_{\text{and}} & := \lambda^0 xy.x + and + y
\end{array} \right.
$$

The parse structure of (2) is $t_3 = c_{\text{run}}(c_{\text{and}} c_{\text{John}}(c_{\text{every}} c_{\text{kid}}))$ and its antecedent by $\mathcal{L}_{\text{amb}}$ is $t_4 = C_{\text{and}}(\lambda^0 P.P C_{\text{John}})(C_{\text{every}} C_{\text{kid}}) C_{\text{run}}$ which has the standard structure of terms that represent the conjunction of quantified noun phrases and not quantified noun phrases in type-logical approaches. Then we get the expected semantic representation from through $\mathcal{L}_{\text{sem}}$: $\mathcal{L}_{\text{sem}}(t_4) = (\mathsf{run}\, \mathsf{j}) \wedge (\forall x.\mathsf{kid}\, x \Rightarrow \mathsf{run}\, x)$

In the next sections, we show how to model some other phenomena:

(3) a. John saw a kid and so did Bill

b. John seeks a book

c. every kid didn't run

3.1 Conjunction and Verbal Ellipsis

Whereas we make some simplification on the syntactic side, the following extensions enable the analysis of (3a):

$$\begin{array}{llll} C_{\text{and so}} & : (np \multimap s) \multimap np \multimap np \multimap s & C_{\text{see}} & : np \multimap np \multimap s \\ c_{\text{and so}} & : (np \multimap s) \multimap np \multimap np \multimap s & c_{\text{see}} & : np \multimap np \multimap s \end{array}$$

$$\begin{array}{lll} \mathcal{L}_{\text{amb}} = & \{ & C_{\text{see}} := c_{\text{see}} \quad C_{\text{and so}} := c_{\text{and so}} \\ \mathcal{L}_{\text{sem}} = & \{ & C_{\text{and so}} := \lambda P.\lambda^0 xy.(Px) \wedge (Py) \quad C_{\text{see}} := \mathsf{see} \\ \mathcal{L}_{\text{syntax}} = & \{ & c_{\text{see}} := saw \quad c_{\text{and so}} := \lambda^0 Rxy.(Rx) + and\ so\ did + y \end{array}$$

With these lexicon, $t_5 = c_{\text{and so}}(\lambda^0 x.c_{\text{see}} x(c_{\text{a}} c_{\text{kid}}))c_{\text{John}} c_{\text{Bill}}$ is a unique parse structure for (3a). But with $t_6 = C_{\text{and so}}(\lambda^0 x.C_{\text{a}} C_{\text{kid}}(\lambda^0 y.C_{\text{see}} xy))C_{\text{John}} C_{\text{Bill}}$ and $t_7 = C_{\text{a}} C_{\text{kid}}(\lambda^0 y.C_{\text{and so}}(\lambda x.C_{\text{see}} xy)C_{\text{John}} C_{\text{Bill}})$ we have that $\mathcal{L}_{\text{amb}}(t_6) = \mathcal{L}_{\text{amb}}(t_7) = t_5$, hence two semantic readings:

$$\begin{array}{ll} \mathcal{L}_{\text{sem}}(t_6) & = (\exists x.(\mathsf{kid}\, x) \wedge (\mathsf{see}\, \mathsf{j}\, x)) \wedge (\exists x.(\mathsf{kid}\, x) \wedge (\mathsf{see}\, \mathsf{b}\, x)) \\ \mathcal{L}_{\text{sem}}(t_7) & = \exists x.(\mathsf{kid}\, x) \wedge ((\mathsf{see}\, \mathsf{j}\, x) \wedge (\mathsf{see}\, \mathsf{b}\, x)) \end{array}$$

3.2 *de re* and *de dicto* Readings

This section shows how to model the *de re* and the *de dicto* readings of (3b).

$$\begin{array}{llll} C_{\text{seek}} & : np \multimap ((np \multimap s) \multimap s) \multimap s & C_{\text{book}} & : n \\ c_{\text{see}} & : np \multimap np \multimap s & c_{\text{book}} & : n \end{array}$$

$$\begin{array}{lll} \mathcal{L}_{\text{amb}} = & \{ & C_{\text{seek}} := \lambda^0 xP.P(\lambda^0 y.c_{\text{seek}}\, x\, y) \quad C_{\text{book}} := c_{\text{book}} \\ \mathcal{L}_{\text{sem}} = & \{ & C_{\text{seek}} := \lambda^0 xo.\mathsf{try}\, x\, (\lambda^0 z.o(\lambda^0 y.\mathsf{find}\, z\, y)) \quad C_{\text{book}} := \mathsf{book} \\ \mathcal{L}_{\text{syntax}} = & \{ & c_{\text{seek}} := \lambda^0 xy.x + seeks + y \quad c_{\text{book}} := book \end{array}$$

With these lexicon, (3b) has a unique parse structure $t_8 = c_{\text{seek}} c_{\text{John}}(c_{\text{a}} c_{\text{book}})$. Let $t_9 = C_{\text{seek}} C_{\text{John}}(C_{\text{a}} C_{\text{book}})$, $t_{10} = (C_{\text{a}} C_{\text{book}})(\lambda^0 y.C_{\text{seek}} C_{\text{John}}\, (\lambda^0 Q.Q\, y))$. We have that $\mathcal{L}_{\text{amb}}(t_9) = \mathcal{L}_{\text{amb}}(t_{10}) = t_8$, hence two semantic readings:

$$\begin{array}{ll} \mathcal{L}_{\text{sem}}(t_9) & = \mathsf{try}\, \mathsf{j}\, (\lambda^0 x.\exists y.(\mathsf{book}\, y) \wedge (\mathsf{find}\, x\, y)) \\ \mathcal{L}_{\text{sem}}(t_{10}) & = \exists y.(\mathsf{book}\, y) \wedge (\mathsf{try}\, \mathsf{j}\, (\lambda^0 x.\mathsf{find}\, x\, y)) \end{array}$$

3.3 Quantification and Negation

Our last example shows how to parse (3c).

$$\begin{array}{ll} C_{\text{didnt}} & : (((np \multimap s) \multimap s) \multimap s) \multimap ((np \multimap s) \multimap s) \multimap s \\ c_{\text{didnt}} & : (np \multimap s) \multimap np \multimap s \end{array}$$

$$\begin{array}{lll} \mathcal{L}_{\text{amb}} = & \{ & C_{\text{didnt}} := \lambda^0 PR.R(\lambda^0 x.P(\lambda^0 Q.c_{\text{didnt}}\, Q\, x)) \\ \mathcal{L}_{\text{sem}} = & \{ & C_{\text{didnt}} := \lambda^0 PQ.\neg(P\, Q) \\ \mathcal{L}_{\text{syntax}} = & \{ & c_{\text{didnt}} := \lambda^0 Rx.x + \textit{didn't} + (R\, \epsilon) \end{array}$$

These lexicons give (3c) a unique parse structure $t_{11} = c_{\text{didnt}}\, c_{\text{run}}(c_{\text{every}} c_{\text{kid}})$. But with $t_{12} = (C_{\text{every}} C_{\text{kid}})(\lambda^0 y.C_{\text{didnt}}(\lambda^0 Q.Q\, C_{\text{run}})(\lambda^0 P.P\, y))$ and $t_{13} = C_{\text{didnt}}(\lambda^0 Q.Q\, C_{\text{run}})(C_{\text{every}} C_{\text{kid}})$ we have that $\mathcal{L}_{\text{amb}}(t_{12}) = \mathcal{L}_{\text{amb}}(t_{13}) = t_{11}$, hence two semantic readings:

$$\begin{array}{ll} \mathcal{L}_{\text{sem}}(t_{12}) & = \forall x.C_{\text{man}}\, x \Rightarrow \neg(C_{\text{run}}\, x) \\ \mathcal{L}_{\text{sem}}(t_1 3) & = \neg(\forall x.C_{\text{kid}}\, x \Rightarrow C_{\text{run}}\, x) \end{array}$$

3.4 Current Limitations

There are some cases where it is not possible to extract quantifiers out of the relative clauses. For instance, (4) has no reading where the universal quantifier has scope over the existential one. As for now, we don't know how to express these constraints in the ACG formalism. Type-logical formalisms deal with these phenomena in using substructural logics and structural control [Mor94].

(4) a man that every woman finds walks

The use of implicative linear logic, more precisely the first order fragment, has been proposed to model some of these phenomena [MP01]. Extensions of the ACG type system in the same spirit have to be explored. In such a framework, we could also distinguish restrictions to extractions coming from syntax (with the type of the constants of Σ_{syntax}) and the restrictions coming from semantics (with the type of the constants of Σ_{synt}).

4 Comparison with Other Approaches

4.1 Scoping Constructor

Because the underlying type systems in the case of ACG and other type-logical formalisms may be quite different (associative and commutative *vs.* not associative, not commutative with structural rules), direct comparison between the expressive power of these approaches is not possible. However, we can compare ACG with a commutative and associative type-logical formalism (then there is only one implication, the linear implication). In that case we can rephrase the inference rules for this system with the scoping constructor. Types are augmented with this constructor:

$$\mathcal{TL}(A) ::= A | \mathcal{TL}(A) \multimap \mathcal{TL}(A) | \mathcal{TL}(A) \Uparrow \mathcal{TL}(A)$$

$$\frac{}{x : \alpha \vdash_{\mathrm{TL}} x : \alpha}\ (\text{ax.})$$

$$\frac{\Gamma, x : \alpha \vdash_{\mathrm{TL}} t : \beta}{\Gamma \vdash_{\mathrm{TL}} \lambda^0 x.t : \alpha \multimap \beta}\ (\text{abs.}) \qquad \frac{\Gamma_1 \vdash_{\mathrm{TL}} t : \alpha \multimap \beta \qquad \Gamma_2 \vdash_{\mathrm{TL}} u : \alpha}{\Gamma_1, \Gamma_2 \vdash_{\mathrm{TL}} (t\, u) : \beta}\ (\text{app.})$$

$$\frac{\Gamma \vdash_{\mathrm{TL}} t : \beta \Uparrow \alpha \qquad \Delta, x : \beta \vdash_{\mathrm{TL}} u : \alpha}{\Gamma, \Delta \vdash_{\mathrm{TL}} t(\lambda x.u) : \alpha}\ (\mathrm{E}_{\Uparrow}) \qquad \frac{\Gamma \vdash_{\mathrm{TL}} t : \beta}{\Gamma \vdash_{\mathrm{TL}} \lambda x.(x\, t) : \beta \Uparrow \alpha}\ (\mathrm{I}_{\Uparrow})$$

We define a translation from $\mathcal{TL}(A)$ to $\mathcal{T}(A)$ as follows:

- if $a \in A$ then $a^{\mathrm{syn}} = a$ and $a^{\mathrm{sem}} = a$
- if $a = \alpha \multimap \beta$ then $a^{\mathrm{syn}} = \alpha^{\mathrm{syn}} \multimap \beta^{\mathrm{syn}}$ and $a^{\mathrm{sem}} = \alpha^{\mathrm{sem}} \multimap \beta^{\mathrm{sem}}$. In the latter formula, $\multimap$ is called a main connective
- if $a = \alpha \Uparrow \beta$ then $a^{\mathrm{syn}} = \alpha^{\mathrm{syn}}$ and $a^{\mathrm{sem}} = (\alpha^{\mathrm{sem}} \multimap \beta^{\mathrm{sem}}) \multimap \beta^{\mathrm{sem}}$ and there is no main connective
- if $\Gamma = a_1, \ldots, a_n$, $\Gamma^{\mathrm{syn}} = {a_1}^{\mathrm{syn}}, \ldots, {a_n}^{\mathrm{syn}}$ and $\Gamma^{\mathrm{sem}} = {a_1}^{\mathrm{sem}}, \ldots, {a_n}^{\mathrm{sem}}$

Theorem 1. *Let $\Gamma \vdash_{TL} t : A$ be a TL sequent. Then $\Gamma \vdash_{TL} t : A$ is provable if and only if $\Gamma^{syn} \vdash A^{syn}$ is provable, $\Gamma^{sem} \vdash u : A^{sem}$ is provable, $u = \lambda^0 x.x\, v$ when the last rule of the proof of $\Gamma^{sem} \vdash u : A^{sem}$ does not introduce a main connective, and $t =_\alpha u$.*

We only sketch the proof here.

Proof. By induction on the proofs. The sufficient condition is rather straightforward. For the necessary condition, the point is that in general in linear logic, $\Gamma, A \multimap B \vdash B$ is provable does not imply that $\Gamma \vdash A$ is provable (take for instance $\Gamma = (A \multimap B) \multimap B$). The condition on the λ-term (it is of the form $\lambda x.x\, v$) makes $\Gamma \vdash v : A$ provable. Then the induction hypothesis can apply. The main connective notion also helps to choose when an application is an actual application or $\mathrm{E}_{\Uparrow}$ on the type-logical side. □

This basically means that the two proposals are able to model the same phenomena, albeit in different ways. We don't make explicit here the role of the lexicon (and how to define it) so that it ensures the relation between the syntactic and the semantic proofs (namely that the image of the semantic proof is the syntactic one). But note that the $\cdot^{\mathrm{syn}}$ and $\cdot^{\mathrm{sem}}$ translations are the one we use in our example to define Σ_{syntax} and Σ_{synt}.

4.2 Glue Semantics

Glue Semantics (GS) [Dal99] was introduced to compute semantic representations from LFG structures, but it has also been applied to other formalisms (HPSG [AC01] and TAG [FvG01]). The principle is to associate to a parse structure a logical formula (of intuitionistic linear logic) which has to be proven. As in our approach and in the type-logical approach, this yields possibly many proofs which are then directly turned, via the Curry-Howard isomorphism, into possibly many semantic interpretations.

But contrary to the type-logical approach, syntactic structures do not directly translate into semantic structures but rather translate into a sequent to prove. This clearly relates to the way the syntactic structure defines conditions on the semantic structure in our proposal (let u be the parse structure, prove $t \in \mathcal{A}(\mathcal{G}_{\text{sem}})$ and $\mathcal{L}_{\text{sem}}(t) = u$).

5 Conclusion

This paper shows how to encode the non-functional relation between syntactic structures and semantic representations in a proof-theoretic setting. This differs from the standard type-logical approach, where semantic ambiguity requires syntactic ambiguity, and from the underspecification framework, where ambiguity is expressed by a specific language. The ACG framework in which it takes place also provides a modularity in the way constraints can be described either on the syntactic level or in the semantic level. Moreover, this proposal gives hints on how to extend the type system of ACG to take such constraints into account. Because ACG can model different grammatical formalisms, we think it can help to share insights from different formalisms and to take pragmatic models into account.

References

[AC01] Ash Asudeh and Richard Crouch. Glue for hpsg. In *Proceedings 8th Int. Conference on Head-Driven Phrase Structure Grammar*, 2001.

[Car97] Bob Carpenter. *Type-Logical Semantics*. The MIT Press, 1997.

[Dal99] Mary Dalrymple, editor. *Semantics and Syntax in Lexical Functional Grammar: The Resource Logic Approach*. MIT Press, 1999.

[dG01] Philippe de Groote. Towards abstract categorial grammars. In *Proceedings of ACL*, pages 148–155, 2001.

[dG06] Philippe de Groote. Towards a motagovian account of dynamics. In *Proceedings of Semantics and Linguistic Theory XVI*, 2006.

[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.

[FvG01] Anette Frank and Josef van Genabith. Glue tag: Linear logic based semantics construction for LTAG. In Miriam Butt and Tracy Holloway King, editors, *Proceedings of the LFG '01 Conference*, Online Proceedings. CSLI Publications, 2001. `http://cslipublications.stanford.edu/LFG/6/lfg01.html`.

[Moo91] Michael Moortgat. Generalized quantifiers and discontinuous type constructors. In W. Sijtsma and A. van Horck, editors, *Discontinuous constituency*. De Gruyter, 1991.

[Mor94] Glyn V. Morrill. *Type Logical Grammar Categorial Logic of Signs*. Kluwer Academin Publishers, 1994.

[MP01] Richard Moot and Mario Piazza. Linguistic applications of first order intuitionistic linear logic. *Journal of Logic, Language and Information*, 10:211–232, 2001.

[Pog04] Sylvain Pogodalla. 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)*, pages 64–71, May 2004.

[Sal05] Sylvain Salvati. *Problmes de filtrage et problmes d'analyse pour les grammaires catgorielles abstraites*. PhD thesis, Institut National Polytechnique de Lorraine, 2005.

[Sal06] Sylvain Salvati. Syntactic descriptions: A type system for solving matching equations in the linear λ-calculus. In Frank Pfenning, editor, *Proceedings of RTA 2006*, volume 4098 of *Lecture Notes in Computer Science*, 2006.

[Yos06] Ryo Yoshinaka. Linearization of affine abstract categorial grammars. In *proceedings of Fromal Grammar 2006*, 2006.