Categorial Minimalist Grammar: From Generative Syntax To Logical Form

Maxime Amblard, Alain Lecomte, Christian Retoré

Categorial minimalist grammars: from generative syntax to logical forms

Maxime Amblard, Alain Lecomte, Christian Retoré
Nancy Université, Université Paris 8, Université de Bordeaux

1. Convergence at first sight, and differences in a second time

From the early days of the minimalist program of Chomsky (1993), a convergence with categorial grammars was noticed by Epstein and Berwick (1995). The striking similarity lies in the merge operation, which looks like the application rule of the AB grammars. In both cases, word order is a consequence of the consumption rules but in categorial grammars being the head of a compound expression also derives from the categories while in a minimalist setting it can be defined independently from the resource consumption. The most striking difference is the absence of movement in categorial grammar: how could this notion be captured? If it can, how could minimality conditions like *shortest move* be formulated? In the other direction the main difference is the atomicity of minimalist features: *merge* is not recursive, there cannot be a demand of a category which itself demand another category.

Hence, after quite an optimistic wish of convergence, the task seems so tough that one may wonder why we are willing to do so? There are at least two reasons:

- Of course, the main reason is that categorial grammar easily yield the logical form (or semantic representation) following some of Montague's idea.
- A secondary reason is the formalisation of converging learning algorithms from positive structured examples, which can be defined for categorial grammars, but we shall not speak about it, although one of us did something in this direction Bonato and Retoré (2001).

Because the categorial tradition is anchored in a formal and logical apparatus, the convergence between generative grammar and categorial grammars benefited from the development of resource

logic and from the formalisation of the minimalist program by Stabler (1997) into tree grammars known as minimalist grammars as a special issue of Language and Computation shows (Retoré and Stabler (2004b)).

The kind of categorial grammar we are using relies on the outcomes of linear logic (Girard, 1987), which extends the Lambek calculus (Lambek, 1958) which itself is a logical completion of the AB-grammars (Bar-Hillel, 1953)when the fraction rules are viewed as *modus ponens*. Lambek calculus is quite a restricted logic: it is an intuitionistic logic, where every hypothesis is used once and exactly once, and where they cannot permute. Our representation uses a mixed system with both commutative and non commutative connectives (de Groote, 1996; Retoré, 2004; Amblard and Retoré, 2007). Still it is a linear subsystem of intutionistic logic, and this enables to derive semantic interpretation from the syntactic analysis.

There are several ways to represent rather faithfully Edward Stabler's minimalist grammars in categorial grammars and to endow them with an integrated treatment of compositional semantics. Some of them take place in the multimodal framework, like (Cornell, 1999; Vermaat, 2004), but here we focus on the one we introduced in the framework of linear logic Lecomte and Retoré (1999, 2001) that we call categorial minimalist grammars. We developed them with others (in particular Anoun (2007)), but in this presentation, we rather follow the one thoroughly presented in the PhD thesis of the first author,Amblard (2007).

We first recall some basic notions on minimalist grammars and on categorial grammars. Next we shortly introduce partially commutative linear logic, and our representation of minimalist grammars within this categorial system, the so-called categorial minimalist grammars. Next we briefly present $\lambda\mu$-DRT (Discourse Representation Theory) an extension of λ-DRT (compositional DRT) in the framework of $\lambda\mu$ calculus: it avoids type raising and derives different readings from a single semantic representation, in a setting which follows discourse structure. We run a complete example which illustrates the various structures and rules that are needed to derive a semantic representation from the categorial view of a transformational syntactic analysis.

Minimalist features

b	=	$\{\mathsf{C}, \mathsf{v}, \mathsf{V}, \mathsf{d}, \mathsf{t}, ...\}$	base syntactic categories
$^{=}\mathsf{b}$	=	$\{=x \mid x \in \mathsf{b}\}$	demand of a base category, selectors
$\mathsf{b}^{\Leftarrow}$	=	$\{x\Leftarrow \mid x \in \mathsf{b}\}$	demand of a base category, with specific placing of the head, i.e. head movement selectors
$-\mathsf{m}$	=	$\{-\mathsf{k}, -\mathsf{wh}, ...\}$	movement triggers, licensees
$+\mathsf{m}$	=	$\{+x \mid -x \in -\mathsf{m}\}$	movement target, licensors

Figure 1: Features and lexical entries in minimalist grammars

2. Minimalist grammars

There are actually several variants of minimalist grammars, due to Edward Stabler, whose guidelines are similar. We focus on the initial formulation of Stabler (1997), an important alternative being Stabler (1999) where operations that one may find unpleasant in a categorial setting like *head-movement*, are simulated by sequences of elementary operations.

As usual in a lexicalised grammar, the lexicon maps each lexical item from $\Phi = \{\mathit{Peter}, \mathit{sleeps}, \mathit{loves}, ...\}$ to a formal encoding of its syntactic behaviour. Here syntactic behaviour is depicted by a sequence of features (category, demand of a category, movement trigger, movement target, see figure 1) followed by the corresponding phonological form. This sequence controls the generating process, which makes use of independent generating functions, only depending on the features of the heads.

(1) *Minimalist lexical entries*
The sequence of features preceding the phonological form matches the following regular expression, whose ingredients are depicted in figure 1:

$$\mathit{Lex}(\mathit{word}) \in \mathcal{L} = (^{=}\mathsf{b} \cup \mathsf{b}^{\Leftarrow}(^{=}\mathsf{b} \cup \mathsf{b}^{\Leftarrow} \cup +\mathsf{m})^{*})^{*}\mathsf{b}(-\mathsf{m})^{*}$$

Such grammars operate on binary trees, the leaves of which are labelled with sequences of features, and the nodes of which are labelled by either "<" or ">". The *head* of a tree is defined recursively as follows: the head of a tree reduced to a leaf is itself. The head of $T_1 < T_2$ and of $T_2 > T_1$ is the head of T_1. Given a leaf ℓ of a tree T, there always exists a biggest subtree T' of T such that l is the head of T': T' is said to be the maximal projection of ℓ.

The *initial rule* says that a tree reduced to a leaf labelled by the lexical entry is itself a derivation tree.

The *merge* rule, is defined for two trees T_1 and T_2, one having a head $\tau_1 = =x\tau'_1\phi_1$ (the demand) and the other having a head $\tau_2 = x\tau'_2\phi_2$, where τ'_1, τ'_2 are sequences of features and ϕ_1, ϕ_2 are sequences of phonological features. Let us call T'_1 and T'_2 the trees obtained from T_1 and T_2 by suppressing from their respective heads the first features, that are respectively x and $=x$. If the demand comes from a lexical item, then the result of the merge operation is $T'_1 < T'_2$. Otherwise, the result is $T'_2 > T'_1$. In any case the head of the result comes from T_1 and is $\tau'_1\phi_1$.

The *head-movement with right adjunction* is actually a special merge and not a kind of movement. It is defined for two trees T_1 and T_2, one having a head $\tau_1 = x{\Leftarrow}\tau'_1\phi_1$ (the demand) and the other having a head $\tau_2 = x\tau'_2\phi_2$, where τ'_1, τ'_2 are sequences of features and ϕ_1, ϕ_2 are sequences of phonological features. Let us call T'_1 the tree obtained from T_1 by turning its head into $\tau_1 = \tau'_1\phi_1\phi_2$ i.e. the demand $x{\Leftarrow}$ has been suppressed and the phonological features are the ones of T_1's head followed by the one of the head of T_2. Let us call T'_2 the tree obtained from T_2 by turning its head into τ'_2 i.e. by suppressing both the x that has been consumed during the merge and the phonological features of the head that have been moved into T'_1. If the demand comes from a lexical item, then the result of the merge operation is $T'_1 < T'2$. Otherwise, the result is $T'_2 > T'_1$. In any case the head of the result comes from T_1 and is $\tau'_1\phi_1$. As far as strings are concerned, head-movement is not required: grammars with or without head-movement generate the same languages, the simulation of head-movement requires many standard merge and moves to be simulated, and yields different parse trees. For having shorter derivations and standard parse trees, we prefer to use head movement in this paper. There also exists head movement with left adjunction, which is defined symmetrically the phonological features moving before the phonological features of the selector head.

One can also define affix-hopping which is rather similar except that the selectee collects the phonological features of the head of the selector.

The *move* rule applies to a single tree T whose head is $+x\tau\phi$ i.e. starts with a movement target $+x \in +\mathsf{m}$, such that T has a subtree T_1 whose head starts with the corresponding movement trigger $-x \in -\mathsf{m}$. Then the result is $T'_1 > T'^{\circ}$ where $T'1$ is T_1 without the $-x$ feature and T'° is T minus the T_1 subtree and minus the $+x$ feature. Observe that the head of the result is the same as the initial one, except that its first feature $+x$ has been erased.[1]

(2) *Language generated by a minimalist grammar*
A convergent derivation generating the sentence $\phi_1 \cdots \phi_n \in \Phi^*$ ends up with a single non phonological feature C on the head, while the leafs read from left to right yield $\phi_1 \cdots \phi_n \in \Phi^*$.

An important requirement on minimalist grammars, both in the principles of Chomsky and in Stabler's formalisation, is the condition known as the *shortest move*. In Chomsky's formulation it says that whenever two subtrees have a $-x\tau$ head, the first one, that is the closes to the target, moves. In Stabler's view it says that whenever there are two candidates to movement, i.e. when there are two leaves starting with $-x$, the derivation crashes. Unless otherwise stated, with stick to Stabler's definition.

Minimalist grammars have rather interesting formal properties, some of which are worth quoting. Indeed, Harkema, Kobele, Michaelis, Morawietz, Mönnich, Retoré, Salvati, Stabler studied minimalist grammars from the viewpoint of formal language theory, the main focus being their generative capacity as sets of strings and as sets of trees, the complexity of parsing and learning strategies. We recall here some of the results:

- The generative capacity of minimalist grammars (with shortest move) is equivalent to that of context free linear rewriting system. Michaelis (2001a,b)

[1] It is possible to also consider weak movement, which leaves the $-x$ subtree at is pace, but suppress the semantic information, and makes a copy of the tree structure with only the semantic features and moves it as we did. As we do not speak about semantic features, we skipped this possible rule.

- In the absence of shortest move, it has been showed that the membership problem is equivalent to the yet unknown decidability of provability of Multiplicative Exponential Linear Logic. Salvati (2007)

- As far as trees are concerned, we know that the minimalist derived trees, that are the ones depicting syntactic structures can be obtained from a regular tree grammar with a transduction applied thereafter. Mönnich *et al.* (2000); Kobele *et al.* (2007)

Although there are some semantic concerns in minimalist grammars as rewriting systems e.g. in Kobele (2006) what is a logical form and how it could be computed is not as clear as it is in the logical approach to categorial grammars. This is likely to be the most frustrating limit of minimalist grammars and the main reason for studying them from a categorial perspective.

For us, the main advantage of minimalist grammars (or in a broader perspective, generative grammar, transformational grammars, etc.) is their linguistic richness and soundness, which is attested by the relation they are able to draw between sentences (questions and related declarative sentences for instance) or between languages (the differences among features are a way to explain language variation).

3. Categorial grammars

A categorial grammar consists in a lexicon which associates with every lexical item a finite set of categories (propositional logical formulae). In the case of AB grammars categories are finitely generated by the connectives / and \ from a set of base categories: S (sentence), np (or dp for noun phrase or determiner phrase), n (common noun) etc. A sequence of words $w_1 \cdots w_n$ is in L(Lex) whenever $\forall i \exists t_i \in \text{Lex}(w_i)$ such that $t_1, \ldots, t_n \vdash S$ is derivable in a logical calculus. For AB grammars the calculus simply consists in the two famous residuation laws, which are elimination rules from the logical viewpoint, i.e. when / and \ are viewed as non commutative linear implications.

Alternatively, one could also label the formulae in the proofs as follows:

$$\frac{(w : \text{word, with } A \in \text{Lex}(w)\ )}{\vdash w : A}\ axiom$$

$$\frac{\vdash w_1 \dots w_n : A \quad \vdash m_1 \cdots m_p : A \setminus B}{\vdash w_1 \cdots w_n m_1 \cdots m_p : B}\ \setminus_e$$

$$\frac{\vdash t : B / A \quad \vdash u : A}{\vdash tu : B}\ /_e$$

This lead to a definition of the generated language as typable strings:

(3) *Language generated by an AB grammar*
(*) $w_1 \cdots w_n \in L(\text{Lex})$ whenever $\vdash w_1 \cdots w_n : S$.

The logical calculus we are to use for categorial minimalist grammars is close to the one above since we shall consider labels and only elimination rules. But it also resembles the Lambek calculus since we have more logical rules and one of them, namely product elimination, requires axioms and variables.[2]

One can wonder why moving out the mathematical paradise provided by usual categorial grammars, AB grammars or Lambek grammars, which are very neat and elegant, especially the Lambek calculus. The main reason is that the syntactic abilities of categorial grammars are limited. A formal way to say so is that they only describe context-free languages, which are commonly assumed to be insufficient to describe natural language syntax, but a more empirical one is their difficulty to describe various syntactic phenomena, like discontinuous constituent, or to relate declarative sentence and questions. [3] Therefore, one has to extend such systems, either by extending the logic by modalities and postulates, as Moortgat (1988) did, thus sticking to the parsing as deduction paradigm, or to have a

[2]Can one extends this definition of the generated language to Lambek grammars? Lambek calculus extends the above calculus with the introduction rules (or abstraction), and this requires to have context and variables (introduced by the axiom $x : A \vdash x : A$). Since variables are abstracted only from leftmost or rightmost positions, one has to carry over context with words to prohibit illicit abstractions, hence the lightness of the above system without contexts is lost.

[3]Although Lambek grammars only describe context free languages, if the grammar is not compiled into a context free grammar, parsing by proof search is NP complete.

logical basis providing the deep structure and an extra mechanism computing word order.

Despite their syntactic limitations, categorial formalisms nevertheless allow an appealing direct computation of (usual) logical formulae from a syntactic analysis. The deep reason, already noticed by the Ancients is that syntactic categories correspond to semantic types, see e.g. Baratin and Desbordes (1981): a sentence S is a proposition t, a definite np is an individual e, a common noun n or an intransitive verb $np \setminus S$ is a one place predicate, a transitive verb is a two place predicate etc. Without providing the details, since we are going to present a similar process for categorial minimalist grammars, let us explain how it works.

In order to compute the semantic formula associated with a sentence, we have:

- A lexicon that provides each word with a syntactic category c and a lambda term whose type is the semantic translation $t = c^*$.
- A syntactic analysis that is a proof in the Lambek calculus of S under the assumption $t_1, \ldots, t_n$, each t_i being a possible category for the word w_i.

The algorithm which computes the associated logical formula is quite simple:

(i) Convert the syntactic categories into semantic types as follows: $np^* = e, S^* = t, n^* = e \to t, (A \setminus B)^* = (B / A)^* = e \to t$: this yields a proof in intuitionistic logic of $S^* = t$ under the assumption $c_1^* = t_1, \ldots, c_n^* = t_n^*$. This proof is a proof that the lambda term hereby defined corresponds to a formulae (is of type t), whose free variables are of type t_i.

(ii) Replacing each variable with the corresponding term of type t_i provided by the lexicon provide a close term of type t, that is a logical formulae.

4. Categorial minimalist grammars

4.1. Labels encoding word order

Let us consider a set of variables V (for expressing hypothesis and encoding movement) and a (disjoint) set of phonological forms Φ (words, lexical items) The labels we are to use are slightly more complicated than sequences of words. A label $\vec{r}$ is a triple each component being a sequence of variables and phonological forms

$$\vec{r} = (r_{spec} \,|\, r_{head} \,|\, r_{comp}) = (r_s \,|\, r_h \,|\, r_c) \in \left[(V \cup \Phi)^*\right]^3$$

Intuitively, these three strings are the yields of the three subtrees respectively corresponding to the head (r_h), the specifier (r_s) and the complement (r_c). The following notations are convenient to denote the suppression of one of the components: $\vec{r}_{-h} = (r_s \,|\, \epsilon \,|\, r_c) \in [(V \cup \Phi)^*]^3$, $\vec{r}_{-s} = (\epsilon \,|\, r_h \,|\, r_c) \in [(V \cup \Phi)^*]^3$, $\vec{r}_{-c} = (r_s \,|\, r_h \,|\, \epsilon) \in [(V \cup \Phi)^*]^3$. With these notations $r_{-h} = r_s\, r_c \in (V \cup \Phi)^*$

Given a label $\vec{r} = (r_s \,|\, r_h \,|\, r_c)$ we denote by r the concatenation of its three components:

$$r = r_s\, r_h\, r_c \in (V \cup \Phi)^*$$

4.2. A restricted fragment of partially commutative logic

To represent minimalist grammars we make use of a very restricted fragment of partially commutative linear logic. This later system introduced by de Groote in de Groote (1996) and extended in Retoré (2004), is a superimposition of the Lambek calculus (intuitionistic non commutative multiplicative linear logic) and intuitionistic commutative multiplicative linear logic. The connectives are both the ones of the Lambek calculus ($\bullet, \backslash, /$) and the ones of (commutative)multiplicative linear logic ($\otimes, \multimap$).

We only consider a particular formulation of a restricted fragment of the calculus to encode minimalist grammars:

- We use natural deduction as in Amblard and Retoré (2007).
- We only use the commutative conjunction ($\otimes$) and the two non commutative implications ($/, \backslash$).
- We only use elimination rules.

$$\frac{\Delta \vdash z : A \otimes B \quad \Gamma[(x : A, y : B)] \vdash t : C}{\Gamma[\Delta] \vdash (\text{let } (x, y) = z \text{ in } t) : C}\,[\otimes_e]$$

Figure 2: Labels for product elimination rule.

- Contexts will simply be multisets of formulae. Usually, for such calculi, contexts are partially ordered multisets of formulae and a rule called *entropy* allows to relax such order. In the restricted calculus we use, we systematically use the full strength of *entropy* just after the eliminations of / and \ which usually introduce order between the assumptions. Hence contexts are simply multisets of formulae.

The product elimination rule is especially important, because it enables a coding of movement (or rather *copy theory* of Brody (1994)) from one hypothesis to the other one. It is a free adaptation of the product elimination rule to the proof expressions of Abramsky (1993) (see figure 2).

The formulae we use have a definition that follows rather precisely the sequences of features that are used in minimalist grammars, i.e. in the set $\mathcal{L}$ defined in 1:

(4) *Logical formulae in a categorial minimalist lexicon*

$$\begin{array}{lll} \mathcal{F} & ::= & \mathsf{x} \,/\, \mathsf{b} \mid \mathsf{x} \,/_{\Leftarrow}\, \mathsf{b} \mid \mathsf{c} \\ \mathsf{x} & ::= & \mathsf{b} \setminus \mathsf{x} \mid \mathsf{b} \,{}_{\Rightarrow}\!\setminus\, \mathsf{x} \mid \mathsf{m} \setminus \mathsf{x} \mid \mathsf{c} \\ \mathsf{c} & ::= & \mathsf{m} \otimes \mathsf{c} \mid \mathsf{b} \end{array}$$

The rules which control the labelling are presented in figure 4.2 — observe that labels do not control the deductive process but are derived from deductive process.

The rules of figure 4.2 will be used as a group of rules which corresponds to the main operations of minimalist grammars, namely *merge* and *move* given in (5) and (6).

We also need to distinguish different \, /-eliminations because head-movement with right adjunction is as far as logical types are concerned a *merge* but is has a different effect on word order. The two rules corresponding to lexical merge / head-movement with

$$\frac{\langle s, A\rangle \in Lex}{\vdash_G (\epsilon \mid s \mid \epsilon) : A}\,[Lex]$$

$$\frac{x \in V}{x : A \vdash_G (\epsilon \mid x \mid \epsilon) : A}\,[axiome]$$

$$\frac{\Gamma \vdash_G \vec{r}_1 : A \,/\, B \quad \Delta \vdash_G \vec{r}_2 : B}{\langle \Gamma; \Delta\rangle \vdash_G (r_{1s} \mid r_{1t} \mid r_{1c}\, r_2) : A}\,[/_e]$$

$$\frac{\Delta \vdash_G \vec{r}_2 : B \quad \Gamma \vdash_G \vec{r}_1 : B \setminus A}{\langle \Gamma; \Delta\rangle \vdash_G (r_2\, r_{1s} \mid r_{1h} \mid r_{1c}) : A}\,[\setminus_e]$$

$$\frac{\Gamma \vdash_G \vec{r}_1 : A \otimes B \quad \Delta[x : A, y : B] \vdash_G \vec{r}_2 : C}{\Delta[\Gamma] \vdash_G \vec{r}_2[r_1/x, \epsilon/y] : C}\,[\otimes_e]$$

$$\frac{\Gamma \vdash_G \vec{r} : A \quad \Gamma' \sqsubset \Gamma}{\Gamma' \vdash_G \vec{r} : A}\,[\sqsubset]$$

Figure 3: Labelled PCMLL deductions. As renaming allows, it is assumed that no variable is common to two labels from different premises: $Var(\vec{r}_1) \cap Var(\vec{r}_2) = \emptyset$. In the $\otimes_e$ rule, A is usually a movement feature.

right adjunction $/_{\Leftarrow}$ or the non lexical variant are provided in figure 7 This is the reason why we use the symbols $_{\Rightarrow}\backslash$ and $/_{\Leftarrow}$ corresponding to head-movement with right adjunction. Clearly, this is a weakness of the system, because as far as proofs are concerned it is weird to have different connectives with the same rules. We can answer to the possible protests that:

- As for minimalist grammars, head-movement is not really required since it can be simulated with regular movement, hence *merge* and *move* which are completely directed by the formulae
- Because the system does not contain the introduction rules, the system is unable to prove $A \setminus B \equiv A_{\Rightarrow}\backslash B$ nor $A / B \equiv A/_{\Leftarrow} B$ which would be quite a problem.

(5) *The operation* merge *in a categorial setting*

$$\frac{\langle sem, A\rangle \in Lex}{\vdash_s em : A}\,[Lex]$$

$$\frac{\vdash g : A \,/\, B \quad \Delta \vdash u : B}{\Delta \vdash (g(u)) : A}\,[mg]$$

$$\frac{\Delta \vdash u : B \quad \Gamma \vdash g : B \setminus A}{\Delta, \Gamma \vdash (g(u)) : A}\,[mg]$$

(6) *The operation* move *in categorial setting*

$$\frac{\Gamma \vdash \vec{s} : A \otimes B \quad \Delta[u : A, v : B] \vdash \vec{r} : C}{\Delta[\Gamma] \vdash \vec{r}[u := s, v := \epsilon] : C}\,[mv]$$

(7) *Head-movement with right adjunction in a categorial setting*

$$\frac{\Gamma \vdash (r_s \,|\, r_h \,|\, r_c) : A /_{\Leftarrow} B \qquad \Delta \vdash (s_s \,|\, s_h \,|\, s_c) : B}{\Gamma, \Delta \vdash (r_s \,|\, r_h\, s_h \,|\, r_c\, s_s\, s_c) : A}\,[mghdr]$$

$$\frac{\Delta \vdash (s_s \,|\, s_h \,|\, s_c) : B \qquad \Gamma \vdash (r_s \,|\, r_h \,|\, r_c) : B \,_{\Rightarrow}\backslash\, A}{\Delta, \Gamma \vdash (s_s\, s_c\, r_s \,|\, r_h\, s_h \,|\, r_c) : A}\,[mghdr]$$

4.3. Grammars

As earlier said, the definition of the generated language resembles the one of categorial grammar given in 3, although here it encompasses a broader notion of label.

(8) *Language generated by a categorial minimalist grammars*
A grammar produces a sentence $\phi_1 \dots \phi_n \in \Phi^*$ whenever there exists a derivation of $\vdash \vec{r} : C$ with $r = \phi_1 \dots \phi_n$ (remember that r is the concatenation of the three components of $\vec{r}$). This definition entails among others that variables must disappear with product elimination rules encoding movement during the derivation process.

A natural question is whether minimalist grammars and categorial grammars generate the same strings and derived trees. The answer is yes, and the proof may be found in Amblard (2007),[4]. It relies on translations from one king of lexicon to the other:

(9) *The translation* $(\_)^\alpha$ *from a lexicon à la Stabler into a categorial minimalist lexicon*

$$
\begin{array}{lcl}
(\gamma -f)^\alpha & = & (f \otimes (\gamma)^\delta) \\
(=f\ \gamma)^\alpha & = & (\gamma)^\beta \ / \ f \\
(f \Leftarrow \gamma)^\alpha & = & (\gamma)^\beta \ /_\Leftarrow \ f \\
(f)^\alpha & = & f \\
(=f\ \gamma)^\beta & = & f \setminus (\gamma)^\beta \\
(f \Leftarrow \gamma)^\alpha & = & f \ {}_\Rightarrow\!\setminus (\gamma)^\beta \\
(+f\ \gamma)^\beta & = & f \setminus (\gamma)^\beta \\
(\gamma -f)^\beta & = & (f \otimes (\gamma)^\delta) \\
(f)^\beta & = & f \\
(\gamma -f)^\delta & = & (f \otimes (\gamma)^\delta) \\
(f)^\delta & = & f
\end{array}
$$

(10) *The translation* $(\_)^{\alpha'}$ *from a categorial minimalist lexicon into a lexicon à la Stabler*

[4]Actually Amblard (2007) does not include the head movemen case, but especially for this construct, the two kind grammars work just the same, *mutatis mutandi*.

$$
\begin{array}{lcl}
(F \;/\; f)^{\alpha'} & = & =f\ (F)^{\beta'} \\
(F \;/_{\Leftarrow}\; f)^{\alpha'} & = & f \Leftarrow (F)^{\beta'} \\
(f \otimes F)^{\alpha'} & = & (F)^{\delta'} - f \\
(f)^{\alpha'} & = & f \\
(f \setminus F)^{\beta'} & = & \begin{cases} =f\ (F)^{\beta'} \; si \; f \in p_1 \\ +f\ (F)^{\beta'} \; si \; f \in p_2 \end{cases} \\
(f \;{}_{\Rightarrow}\backslash\; F)^{\beta'} & = & f \Leftarrow (F)^{\beta'} \\
(f)^{\beta'} & = & f \\
(f \otimes F)^{\beta'} & = & (F)^{\delta'} - f \\
(f \otimes F)^{\delta'} & = & (F)^{\delta'} - f \\
(f)^{\delta'} & = & f
\end{array}
$$

(11) *Equivalence of categorial minimalist grammars and minimalist grammars*
If a minimalist grammar à la Stabler is turned into a categorial minimalist lexicon according to the rules in figure 9, then the generated sentences are the same and even parse trees are almost the same (provided the subtree undergoing movement is correctly placed). Conversely, if one turns a categorial minimalist lexicon (only containing formulae in CMG defined in figure 4) into a minimalist lexicon according to the rules in 10, then one also gets the same generated sentences and parse trees.

A remaining question is how constraints on derivations, like *shortest move*, could be formulated in a deductive setting. This is a weakness of the system: the only way to formulate these restrictions is to impose constraints on derivations, and this is not so appealing from a logical viewpoint since rule are usually context free, i.e. apply locally without referring to the history of the deduction — unless one goes to dependent types, which are complicated and not yet explored for linear calculi.

5. Syntax and semantics in categorial minimalist grammars

The purpose of viewing minimalist grammars as a categorial system is to have an automatic conversion of syntactic compositions into semantic composition. We firstly present the semantic

types and their relation to syntactic categories, then a variant of λ-DRT with μ constructs in order to obtain the different scope readings from a single semantic representation, and finally sum up the algorithm producing semantic analyses.

5.1. From syntactic categories to semantic types

The formulae that we infer as semantic representations are first order, because we are going to use DRT which better matches the discourse structure. Therefore we reify predicates, we use variables to express high order properties: we write $run(e, Rex) \wedge fast(e)$ instead of $fast(run(Rex))$. Consequently we have the following three base types

- **t** the standard Montague type, for truth values and propositions (that will be used as $\perp$ from the μ-calculus viewpoint).
- **e** the standard Montague type, for entities.
- **v** the type for reifying events and predicates in order to remain in first order logic.

The standard translation H of a syntactic category x into the corresponding semantic type $H(\mathsf{x})$ has to be slightly extended, because in addition to the syntactic connectives /, \ we have $\otimes$, together with an extra basic type for reifications.

(12) *Converting syntactic categories into semantic types*

$H(\mathsf{C})$	=	**t**
$H(\mathsf{v})$	=	$\mathbf{v} \rightarrow \mathbf{t}$
$H(\mathsf{t})$	=	$\mathbf{v} \rightarrow \mathbf{t}$
$H(\mathsf{V})$	=	$\mathbf{e} \rightarrow (\mathbf{v} \rightarrow \mathbf{t})$
$H(\mathsf{d})$	=	**e**
$H(\mathsf{n})$	=	$\mathbf{e} \rightarrow \mathbf{t}$
$H(\mathsf{x} \otimes \mathsf{y})$	=	$H(\mathsf{y})$ if $a \in \mathsf{m}$
	=	$H(\mathsf{x})$ if $a \notin \mathsf{m}$
$H(\mathsf{x} \backslash \mathsf{y})$	=	$H(\mathsf{x}) \rightarrow H(\mathsf{y})$
$H(\mathsf{y} / \mathsf{x})$	=	$H(\mathsf{x}) \rightarrow H(\mathsf{y})$
$H(\mathsf{x} \Rightarrow\backslash \mathsf{y})$	=	$H(\mathsf{x}) \rightarrow H(\mathsf{y})$
$H(\mathsf{y} /\Leftarrow \mathsf{x})$	=	$H(\mathsf{x}) \rightarrow H(\mathsf{y})$

This translation can be illustrated on the syntactic categories we are to use in the example of section 6.

$H(\mathsf{k} \otimes \mathsf{d})$	=	$\mathbf{e}$
$H(\mathsf{V} \,/\, \mathsf{d})$	=	$(\mathbf{v}\to\mathbf{t})\to\mathbf{e}\to\mathbf{v}\to\mathbf{t}$
$H((\mathsf{k}\backslash(\mathsf{d}\backslash\mathsf{v})) \,/_{\Leftarrow}\, \mathsf{V})$	=	$(\mathbf{e}\to\mathbf{v}\to\mathbf{t})\to\mathbf{e}\to\mathbf{e}\to\mathbf{v}\to\mathbf{t}$
$H((\mathsf{k} \setminus \mathsf{t}) \,/_{\Leftarrow}\, \mathsf{v})$	=	$(\mathbf{v} \to \mathbf{t}) \to \mathbf{e} \to \mathbf{v} \to \mathbf{t}$
$H(\mathsf{k} \otimes \mathsf{d} \,/\, \mathsf{n})$	=	$(\mathbf{e} \to \mathbf{t}) \to \mathbf{e}$
$H(\mathsf{C} \,/\, \mathsf{t})$	=	$(\mathbf{v} \to \mathbf{t}) \to \mathbf{t}$

5.2. $\lambda\mu$-calculus

Semantics can be computed in standard Montagovian terms and types, as did Amblard *et al.* (2003) or Lecomte and Retoré (2002) We extend a bit this process, by moving from λ-calculus to $\lambda\mu$ calculus, a λ calculus for depicting classical proofs (while standard λ-calculus correspond to proofs in intuitionistic propositional logic). The advantage is that quantifiers do not need to be type raised and that the different scopes are obtained from a single syntactic semantic analysis, by different $\lambda\mu$ computations, as initiated by de Groote (2001) and developed for categorial minimalist grammars in Amblard (2007) or Lecomte (2008).

The μ calculus is strongly related to the formulation of classical logic in natural deduction using *reductio as absurdum*: instead of concluding $\neg\neg A = (A \to \bot) \to \bot$ from $\bot$ under the assumption $\neg A$ one concludes A. Provided one does not use the $\bot$ rule (*ex falso quod libet sequitur*), this can be done with any constant type for $\bot$. Here, as in de Groote (2001), we shall use Montague **t** type (truth values, propositions) for the $\bot$ of the glue language that is the $\lambda\mu$-calculus. As opposed to common intuitions, this is really harmless!

We will use a restricted calculus, whose typing rules are very easy: basic types are the aforementioned **t**, **e**, **v** and, as said above, **t** will be used as $\bot$. Types are defined as usual, with the arrow (intuitionistic implication) only.

For each type Z, we have a denumerable set of λ-variables of type Z.

We have μ variables with type $X \to \mathbf{t}$ for some type X.

There are three kinds of terms, Values V, Unamed Terms v and Named Terms c with $V \subset v$, which are defined as the smallest sets closed under the following operations:

- a λ-variable $x : X$ is a value and an unnamed term.
- if $u : U$ is an unnamed term and x a λ variable then $\lambda x.\ u : X \to U$ is a value and an unnamed term
- if $v_1 : X \to U$ and $v_2 : X$ are unnamed terms, $(v_1(v_2)) : U$ is an unnamed term
- if $v : X$ is an unnamed term and $\alpha : X \to \mathbf{t}$ a μ-variables, than $(\alpha(v)) : X$ is a named term
- if $v : X$ is a named term and $\alpha : X \to \mathbf{t}$ is a μ-variable then $(\mu\alpha.\ v) : X$ is an unnamed term.
- Any term of type $\mathbf{t}$ is a named term (this follows from $\bot = \mathbf{t}$).

The reduction patterns for $\lambda\mu$-calculus include the usual β conversion as well as reductions concerning μ variables and the μ-binder:

(13) *$\lambda\mu$ reductions*

$$
\begin{array}{llll}
(\beta) & ((\lambda x.\ t)(u)) & \rightsquigarrow_\beta & t[u/x] \\
(\mu) & ((\mu\alpha.\ T[(\alpha\ Q)])P) & \rightsquigarrow_\mu & \mu\alpha.\ T[(\alpha(Q\ P))] \\
(\mu') & (P(\mu\alpha.\ T[(\alpha\ Q)])) & \rightsquigarrow_{\mu'} & \mu\alpha.\ T[(\alpha(P\ Q))] \\
(\varsigma) & ((\mu\alpha.\ T[(\alpha\ Q)]) & \rightsquigarrow_\varsigma & T[Q] \qquad \text{only for } Q : \mathbf{t}
\end{array}
$$

Let us recall that $\lambda\mu$ calculus, with the given reductions, is *not confluent*. This will be useful to obtain the many readings of a given sentence, as first observed by de Groote (2001).

5.3. $\lambda\mu$-DRS

Instead of standard Montague semantics, we prefer to produce Discourse Representation Structures (DRS), whose dynamics better matches the one of sentences and discourse. Of course, given the wished correspondence with the categorial framework, we use a compositional presentation of DRT, like λ-DRT. But, because we also wish to have μ reductions for avoiding type raising and solving scope questions, we use a generalisation of λ-DRT that we call $\lambda\mu$-DRT.

Basically, we consider DRS with λ and μ variables (all μ variables will be of type $X \to \mathbf{t}$), using freely both λ and μ abstractions, and we refer to these structures as $\lambda\mu$-DRS (Discourse Representation Structure) In addition to the λ and μ variables and the variables used for movement which can be considered as free λ variables we need a set of discourse refrents V_D of type **e** (some could be of type **v**) denoted by $d_1, \cdots, d_n$.

The lexicon in addition to the syntactic categories maps each lexical item to a $\lambda\mu$-DRS. A $\lambda\mu$-DRS is typed term with three different kinds of variables V_λ (containing movement variable, which have nothing special except that they are not bound by any λ), V_μ and V_D and three corresponding kinds of binders the later one allowing dynamic binding. Variables that are μ or λ bound can be suppressed from the context, while discourse referents of V_D should remain in the context until some simplifications are performed, as their binding is sophisticated, see 14.

Types for these terms are defined as usual, from three base types $B = \{\mathbf{e}, \mathbf{v}, \mathbf{t}\}$ (entities, events, truth-values) by the arrow:

$$T = B \mid T \to T$$

We have constants, namely predicates of type $\upsilon \to (\upsilon \to (\upsilon \to \cdots \mathbf{t}))$ where υ is either **e** or **v** — there is no need to systematically force that a predicate only involves one event, itself. As usual, an intransitive verb has type $\mathbf{v} \to \mathbf{e} \to \mathbf{t}$, a transitive verbe $\mathbf{v} \to \mathbf{e} \to \mathbf{e} \to \mathbf{t}$, a ditransitive verbe $\mathbf{v} \to \mathbf{e} \to \mathbf{e} \to \mathbf{e} \to \mathbf{t}$, and a property $\mathbf{e} \to \mathbf{t}$.

We also have logical constants:

- $\wedge : \mathbf{t} \to \mathbf{t} \to \mathbf{t}$
- $\underline{\wedge} : \mathbf{t} \to \mathbf{t} \to \mathbf{t}$
- $\Rightarrow : \mathbf{t} \to \mathbf{t} \to \mathbf{t}$
- $\doteq : \mathbf{e} \to \mathbf{e} \to \mathbf{t}$

There is no constant for $\exists$ ($[d|F]$ is close to $\exists d\ F$) nor for $\forall$ ($[|[d|F] \Rightarrow [|G]]$ is close to $\forall d\ (F \Rightarrow G)$), since we thus obtain better scope properties. The variant of $\wedge$ named $\underline{\wedge}$ has a slightly different behaviour with respect to discourse variables. The operation $\underline{\wedge}$ concerns DRS as well as formulae, and is a kind of a fusion, similar to the merge operations explored by Muskens (1996).

Variable	Term	Result
$\alpha : (X \to t) \in V_\mu$	$u : X$	$\mu\alpha.\, u : X$
$x : X \in V_\lambda$	$u : U$	$(\lambda x.u) : X \to U$
$d : \mathbf{e} \in V_d$ $d : \mathbf{v} \in V_d$	$u : \mathbf{t}$	$[d\|u] : t$

We also have applications:

First Term	Second Term	Application Result
$\alpha : (X \to \mathbf{t})$	$u : X$	$(\alpha(u)) : \mathbf{t}$
$u : X \to Y$	$v : X$	$(u(v)) : Y$

Given a formula $F : \mathbf{t}$ we use the following shorthands:

- let $F : \mathbf{t}$ then $[|F] = F$ in the very same context. In particular $F \to G$ can be viewed as $[|F] \Rightarrow [|G]$.
- let $F : \mathbf{t}$ then $[d_1|[d_2[d_3|[\cdots[d_n[|F]]]]]] = [d_1 \cdots d_n|F]$ the context being unchanged.

A $\lambda\mu$-DRS is a typed term of this calculus. When it is of type $\mathbf{t}$, without μ-binder nor μ-variable, it is an ordinary λ DRS, and when, furthermore, there is neither λ binder nor λ variables, it is an ordinary DRS (provided formulae involved in implications are viewed as short hands for DRS without discourse variables). In order to formulate the reduction of $\underline{\wedge}$ which encodes dynamic binding, we need some standard notions on DRS and subDRSs.

A sub DRS is simply a subterm. A subDRS G of a DRS H is said to be a *positive* (resp. *negative*) subDRS of H written $\prec^+$ (resp. $\prec^-$) whenever:

- $G = H$
- $H = [d|K]$ and $G \prec^+ K$ (resp. $G \prec^- K$)).
- $H = K \wedge L$ or $H = K \,\underline{\wedge}\, L$ and $G \prec^+ K$ (resp. $G \prec^- K$)
- $H = K \Rightarrow L$ and $G \prec^+ K$ or $G \prec^- L$ (resp. $G \prec^+ L$ or $G \prec^- K$)

The reference markers associated to a DRS $[d_1 \cdots d_n|F]$ are the discourse refrents $d_1 \cdots d_n$. The accessibility relation is the closure of $(\prec) \cup (\nwarrow)$ where $F \nwarrow G$ whenever $F \to G$. The accessible

referent markers of a DRS are the referent markers of the accessible DRS.

As said above, $\underline{\wedge}$ operation is close to the DRS merge in the literature, but here it will be called *fusion* because we use *merge* as a term denoting a syntactic operation. Although it could be defined on the run, the reduction of $\underline{\wedge}$ is easier to define on a normal $\lambda\mu$-DRS, that is a DRS without λ abstraction nor μ abstractions. It can be reduced when it applies to a DRS D and a single formula F with free discourse variables $fdv(F)$. To view this reduction as a fusion (merge), firstly turn F into $[fdv(F)|F]$: then the resemblance should be clear.

(14) *Internalisation*
If there exist one largest $K = [k_1 \cdots k_s|G]$ positive subDRS of D whose accessible referent markers includes $fdv(F)$ then $D \underline{\wedge} F$ reduces to D with subDRS K replaced with $K = [k_1 \cdots k_s|G \wedge F]$. If there is no such positive subDRS, or if there is no largest one, no internalisation is possible.

5.4. The process of syntactic and semantic derivation

Assume we have a lexicon which maps a word to its syntactic category x (that can be inferred from a minimalist grammar) together with the corresponding $\lambda\mu$-DRS with the semantic type $H(x)$ defined in 5. Firstly, we should provide the semantic counter part of the syntactic rules. Merge does, as usual, correspond to λ application: the head is the semantic function which is applied to the semantics of the argument. Head movement with right adjunction, is just a merge from the semantic viewpoint — indeed the resource consumption is just the same, only word order is different from standard merge.

(15) *Semantic counterpart of* merge *and of* head-movement
In the *merge* and *hdr* rules of figures 5 and 7 let $H(s) : H(B)$ be the semantic term corresponding to $\vec{s} : B$ i and let $H(r) : H(A) \rightarrow H(B)$ be the one corresponding to $\vec{r} : A / B$ (or $A /_{\Leftarrow} B$ or $B \setminus A$ or $B \Rightarrow\backslash A$) (all the possible syntactic categories for $\vec{r}$ translated into the semantic type $H(A) \rightarrow H(B)$). Then the semantic term associated with the resulting category A is $H(r)(H(s)) : H(A)$ — the functional application of the

semantic term associated with the function-category to the semantic term associated with the argument.

The semantic counterpart of the *move* rule of figure 6 is more complex. As may be observed in the syntactic rule, *move* connects two places of respective categories A and B, in an elimination rule with main premise $A \otimes B$. These two places corresponds to two λ-variables u and v that by construction also appear in the semantic term. Items that can undergo movement, typically determiner phrases missing a case, or *wh* constituents, have a semantic term s associated with a particular discourse variable $d(s)$ introduced in the lexicon within the semantic term of the determiner. *Move* consists in replacing the first variable u with the semantic term s, and the second, v with the corresponding discourse variable $d(s)$. The internalisation process described above in 14 is precisely made to connect the formula in which $d(s)$ appears and the DRS that derives from s and which is meant to bound the discourse referent $d(s)$.

(16) *The semantic counterpart of* move

In the rule below $d(s)$ is the distinguished discourse variable associated with the movable semantic term s.

$$\frac{\Gamma \vdash H(s){:}A \otimes B \quad \Delta[u{:}H(A), v{:}H(B)] \vdash H(r)[u, v]{:}H(C)}{\Delta[\Gamma] \vdash H(r)[u{:=}H(s), v{:=}d(s)]{:}H(C)}\ [mv]$$

To sum up, how do we proceed to compute both the syntactic and the semantic structure associated to a sentence $\phi_1 \cdots \phi_n \in \Phi^*$?

(17) *Computing the parse structure and the logical forms in a categorial minimalist grammar*

(i) Firstly, we construct a proof with the rules *mv*, *mg* and *hdr* of $\vec{r} : \mathsf{C}$ with $r = \phi_1 \cdots \phi_n$.

(ii) Afterwards or in parallel steps, this provides a $\lambda\mu$-DRS of type t without free λ variables, nor with free μ variables, that is nearly what we were looking for: we only have to internalise the formulae concerning some discourse variable d that lie outside the corresponding box which bounds

d. This may seem problematic because it turns some free variables into bounded ones but as we proceed just after the derivation, before any renaming takes place, this is harmless. Also by construction, there cannot be conflict about in which box the formula must be moved.

$$[d, f|P(d, f)] \wedge Q(f) \rightsquigarrow [d, f|P(d, f) \wedge Q(f)]$$

Hence we end up with a $\lambda\mu$ DRS D of type t without free variables of any kind, and we may call D the semantic representation of the sentence $[\![\phi]\!]$.

(iii) Nevertheless $[\![\phi]\!]$ still contains μ variables and abstractions, which prevents us from interpreting directly $[\![\phi]\!]$ as a formulae. Performing β and μ reductions (μ, μ', ς) yields ordinary DRS that are usual the semantic representations of the sentence. The different scope readings result from the non confluence of μ reductions, as in de Groote (2001).

The next section provides detail on all these steps by running a complete example.

6. A complete example of a syntactic and semantic analysis

Here is a samble lexicon from which we will derive the two reading of the ambiguous sentence: *the children ate a pizza*.

lexical item	$\lambda\mu$**-DRS & semantic type**	**syntactic category**
the	$\lambda Q.\mu\delta.[\|[d\|(Q\ d)] \Rightarrow [(\delta\ d)]]$ $(\mathbf{e} \to \mathbf{t}) \to \mathbf{e}$ specific discourse variable: *d*	$\mathsf{k} \otimes \mathsf{d}\ /\ \mathsf{n}$
a	$\lambda Q.\mu\gamma.[p\|(Q\ p) \wedge (\gamma\ p)]$ $(\mathbf{e} \to \mathbf{t}) \to \mathbf{e}$ specific discourse variable: *p*	$(\mathsf{k} \otimes \mathsf{d})\ /\ \mathsf{n}$
children	$\lambda z.(\mathtt{child}\ z)$ $\mathbf{e} \to \mathbf{t}$	n
pizza	$\lambda z.(\mathtt{piz}\ z)$ $\mathbf{e} \to \mathbf{t}$	n
ate	$\lambda x\lambda y\lambda e.\mathtt{eat}(e, x, y)$ $\mathbf{e} \to \mathbf{e} \to \mathbf{v} \to \mathbf{t}$	$\mathsf{V}\ /\ \mathsf{d}$
modif	$\lambda R\lambda x_2\lambda y\lambda e.$ $R(y, e) \underline{\wedge}\ \mathtt{Pa}(e, u)$ $(\mathbf{e} \to \mathbf{v} \to \mathbf{t}) \to \mathbf{e} \to \mathbf{e} \to \mathbf{v} \to \mathbf{t}$	$(\mathsf{k}\backslash(\mathsf{d}\backslash\mathsf{v}))\ /_{\Leftarrow}\ \mathsf{V}$
infl	$\lambda Q\lambda y_2\lambda e.Q(e)$ $\wedge \mathtt{P}(e) \underline{\wedge}\ \mathtt{Ag}(e, y_2)$ $(\mathbf{v} \to \mathbf{t}) \to \mathbf{e} \to \mathbf{v} \to \mathbf{t}$	$(\mathsf{k} \setminus \mathsf{t})\ /_{\Leftarrow}\ \mathsf{v}$
comp	$\lambda Q.[e\|(Q(e))]$ $(\mathbf{v} \to \mathbf{t}) \to \mathbf{t}$	$\mathsf{C}\ /\ \mathsf{t}$

In order to complete both the syntactic and the semantic process, described in the previous section on this example, we are to use the following structure:

declaration of the syntactic variables

declaration of the corresponding semantic variables

$\vdash$ { label, triple of phonological forms; syntactic category; semantic type; DRS }

$$\frac{\vdash \begin{cases} (\varepsilon \mid \mathsf{a} \mid \varepsilon) \\ (\mathsf{k} \otimes \mathsf{d}) \,/\, \mathsf{n} \\ (\mathbf{e} \to \mathbf{t}) \to \mathbf{e} \\ \lambda Q.\mu\gamma.[p \mid Q(p) \wedge \gamma(p)] \end{cases} \qquad \vdash \begin{cases} (\varepsilon \mid \text{pizza} \mid \varepsilon) \\ \mathsf{n} \\ \mathbf{e} \to \mathbf{t} \\ \lambda z.\mathtt{piz}(z) \end{cases}}{\vdash \begin{cases} (\varepsilon \mid \mathsf{a} \mid \text{pizza}) \\ \mathsf{k} \otimes \mathsf{d} \\ \mathbf{e} \\ \mu\gamma. \\ [p \mid \mathtt{piz}(p) \wedge \gamma(p)] \end{cases}} \; mg$$

$$\frac{\vdash \begin{cases} (\varepsilon \mid \text{the} \mid \varepsilon) \\ (\mathsf{k} \otimes \mathsf{d}) \,/\, \mathsf{n} \\ (\mathbf{e} \to \mathbf{t}) \to \mathbf{e} \\ \lambda Q\mu\delta.[\,\mid [d \mid Q(d)] \Rightarrow [\delta(d)]] \end{cases} \qquad \vdash \begin{cases} (\varepsilon \mid \text{children} \mid \varepsilon) \\ \mathsf{n} \\ \mathbf{e} \to \mathbf{t} \\ \lambda z.\mathtt{child}(z) \end{cases}}{\vdash \begin{cases} (\varepsilon \mid \text{the} \mid \text{children}) \\ \mathsf{k} \otimes \mathsf{d} \\ \mathbf{e} \\ \mu\delta.[\,\mid [d \mid \mathtt{child}(d)] \\ \quad \Rightarrow [\,\mid \delta(d)]] \end{cases}} \; mg$$

Figure 4: Constructing the two dp's: Two steps of the derivation may be performed independently. They both consist in constructing a determiner phrase by a merge rule and semantically the determiner (some or all) applies to the predicate. Notice that the semantic of quantifiers include a μ-abstraction.

The first step is a lexical merge with a variable that will be used later on for movement. On the semantic side merge corresponds to the application of the verbal lambda term to the variable.

$$\frac{\vdash \begin{cases} (\varepsilon \mid \text{eat} \mid \varepsilon) \\ \mathsf{V} \,/\, \mathsf{d} \\ \mathbf{e} \to \mathbf{e} \to \mathbf{v} \to \mathbf{t} \\ \lambda x \lambda y \lambda e.\mathtt{eat}(e, x, y) \end{cases} \qquad \begin{matrix} u{:}\mathsf{d} \\ \underline{u}{:}\mathbf{e} \end{matrix} \vdash \begin{cases} (\varepsilon \mid u \mid \varepsilon) \\ \mathsf{d} \\ \mathbf{e} \\ \underline{u} \end{cases}}{\begin{matrix} u{:}\mathsf{d} \\ \underline{u}{:}\mathbf{e} \end{matrix} \vdash \begin{cases} (\varepsilon \mid \text{eat} \mid u) \\ \mathsf{V} \\ \mathbf{e} \to \mathbf{v} \to \mathbf{t} \\ \lambda y \lambda e. \\ \mathtt{eat}(e, \underline{u}, y) \end{cases}} \; mg$$

Next step is a head-movement with right adjunction triggered by *modif*. Semantically it is also an application, which introduces the Pa thematic role:

$$\frac{\vdash \begin{cases} (\varepsilon \mid \varepsilon \mid \varepsilon) \\ (\mathsf{k}\backslash(\mathsf{d}\backslash\mathsf{v})) \,/_{\Leftarrow}\, \mathsf{V} \\ (\mathbf{e}{\to}\mathbf{v}{\to}\mathbf{t}){\to}\mathbf{e}{\to}\mathbf{e}{\to}\mathbf{v}{\to}\mathbf{t} \\ \lambda R \lambda x_2 \lambda y \lambda e. \\ R(y, e) \underline{\wedge}\, \mathtt{Pa}(e, x_2) \end{cases} \qquad \begin{matrix} u{:}\mathsf{d} \\ \underline{u}{:}\mathbf{e} \end{matrix} \vdash \begin{cases} (\varepsilon \mid \text{eat} \mid u) \\ \mathsf{V} \\ \mathbf{e} \to \mathbf{v} \to \mathbf{t} \\ \lambda y \lambda e. \\ \mathtt{eat}(e, \underline{u}, y) \end{cases}}{\begin{matrix} u{:}\mathsf{d} \\ \underline{u}{:}\mathbf{e} \end{matrix} \vdash \begin{cases} (\varepsilon \mid \text{eat} \mid u) \\ \mathsf{k}\backslash(\mathsf{d}\backslash\mathsf{v}) \\ \mathbf{e} \to \mathbf{e} \to \mathbf{v} \to \mathbf{t} \\ \lambda x_2 \lambda y \lambda e.\mathtt{eat}(e, \underline{u}, y) \\ \underline{\wedge}\mathtt{Pa}(e, x_2) \end{cases}} \; hdr$$

Next step introduces in a merge rule a variable for case that will be used for movement. On the semantic side, the variable gets the patient role.

$$\frac{\begin{array}{l}v{:}\mathsf{k}\\ \underline{v}{:}\mathbf{e}\end{array}\vdash\left\{\begin{array}{l}(\varepsilon\,|\,v\,|\,\varepsilon)\\ \mathsf{k}\\ \mathbf{e}\\ \underline{v}\end{array}\right.\qquad\begin{array}{l}u{:}\mathsf{d}\\ \underline{u}{:}\mathbf{e}\end{array}\vdash\left\{\begin{array}{l}(\varepsilon\,|\,\mathrm{eat}\,|\,u)\\ \mathsf{k}\backslash(\mathsf{d}\backslash\mathsf{v})\\ \mathbf{e}\to\mathbf{e}\to\mathbf{v}\to\mathbf{t}\\ \lambda x_2\lambda y\lambda e.\mathtt{eat}(e,\underline{u},y)\\ \underline{\wedge}\mathsf{Pa}(e,x_2)\end{array}\right.}{\begin{array}{l}v{:}\mathsf{k},\,u{:}\mathsf{d}\\ \underline{v}{:}\mathbf{e},\,\underline{u}{:}\mathbf{e}\end{array}\vdash\left\{\begin{array}{l}(v\,|\,\mathrm{eat}\,|\,u)\\ \mathsf{d}\backslash\mathsf{v}\\ \mathbf{e}\to\mathbf{v}\to\mathbf{t}\\ \lambda y\lambda e.\\ \mathtt{eat}(e,\underline{u},y)\\ \underline{\wedge}\mathsf{Pa}(e,\underline{v})\end{array}\right.}\,mg$$

A move inserts the dp a pizza (computed in figure 6) into the main derivation. Syntactically, the movement checks the case, and the phonological string replaces the variable. Semantically the $\lambda\mu$-DRS replaces the variable $\underline{u}$, and the specific discourse referent p replaces $\underline{v}$.

$$\frac{\vdash\left\{\begin{array}{l}(\varepsilon\,|\,\mathrm{a}\,|\,\mathrm{pizza})\\ \mathsf{k}\otimes\mathsf{d}\\ \mathbf{e}\\ \mu\gamma.\\ [p\,|\,\mathtt{piz}(p)\wedge\gamma(p)]\end{array}\right.\qquad\begin{array}{l}v{:}\mathsf{k},\,u{:}\mathsf{d}\\ \underline{v}{:}\mathbf{e},\,\underline{u}{:}\mathbf{e}\end{array}\vdash\left\{\begin{array}{l}(v\,|\,\mathrm{eat}\,|\,u)\\ \mathsf{d}\backslash\mathsf{v}\\ \mathbf{e}\to\mathbf{v}\to\mathbf{t}\\ \lambda y\lambda e.\\ \mathtt{eat}(e,\underline{u},y)\\ \underline{\wedge}\mathsf{Pa}(e,\underline{v})\end{array}\right.}{\vdash\left\{\begin{array}{l}(\mathrm{a\ pizza}\,|\,\mathrm{eat}\,|\,\varepsilon)\\ \mathsf{d}\backslash\mathsf{v}\\ \mathbf{e}\to\mathbf{v}\to\mathbf{t}\\ \lambda y\lambda e.\\ \mathtt{eat}(e,\mu\gamma.[p\,|\,\mathtt{piz}(p)\wedge\gamma(p)],y)\\ \underline{\wedge}\mathsf{Pa}(e,p)\end{array}\right.}\,mv$$

Next a variable corresponding to the subject (another dp) is introduced in a non-lexical merge-rule. Semantically, we just apply a λ-expression to the variable:

$$\frac{\begin{array}{l} w{:}\mathsf{d} \\ \underline{w}{:}\mathbf{e} \end{array} \vdash \left\{\begin{array}{l} (\varepsilon \mid w \mid \varepsilon) \\ \mathsf{d} \\ \mathbf{e} \\ \underline{w} \end{array}\right. \qquad \vdash \left\{\begin{array}{l} (\text{a pizza} \mid \text{eat} \mid \varepsilon) \\ \mathsf{d}\backslash\mathsf{v} \\ \mathbf{e} \to \mathbf{v} \to \mathbf{t} \\ \lambda y \lambda e. \\ \mathtt{eat}(e, \mu\gamma.[p \mid \mathtt{piz}(p) \wedge \gamma(p)], y) \\ \underline{\wedge}\mathtt{Pa}(e, p) \end{array}\right.}{\begin{array}{l} w{:}\mathsf{d} \\ \underline{w}{:}\mathbf{e} \end{array} \vdash \left\{\begin{array}{l} (w \text{ a pizza} \mid \text{eat} \mid \varepsilon) \\ \mathsf{v} \\ \mathbf{v} \to \mathbf{t} \\ \lambda e. \\ \mathtt{eat}(e, \mu\gamma.[p \mid \mathtt{piz}(p) \wedge \gamma(p)], \\ \underline{w})\underline{\wedge}\mathtt{Pa}(e, p) \end{array}\right.}\ mg$$

The verb is now ready to receive its inflection. Syntactically it is a head movement with right-adjunction that glues the inflection mark to the right of the verb (*eat_infl* should be understood as *ate*). Semantically, the λ-term associated with the inflection is applied to the term we derived so far.

$$\frac{\vdash \left\{\begin{array}{l} (\varepsilon \mid \mathit{infl} \mid \varepsilon) \\ (\mathsf{k} \setminus \mathsf{t}) /_{\Leftarrow} \mathsf{v} \\ (\mathbf{v}{\to}\mathbf{t}){\to}\mathbf{e}{\to}\mathbf{v}{\to}\mathbf{t} \\ \lambda Q \lambda y_2 \lambda e. \\ Q(e)\underline{\wedge}\mathsf{P}(e) \\ \wedge \mathit{agent}(e, y_2) \end{array}\right. \quad \begin{array}{l} w{:}\mathsf{d} \\ \underline{w}{:}\mathbf{e} \end{array} \vdash \left\{\begin{array}{l} (w \text{ a pizza} \mid \text{eat} \mid \varepsilon) \\ \mathsf{v} \\ \mathbf{v} \to \mathbf{t} \\ \lambda e. \\ \mathtt{eat}(e, \mu\gamma.[p \mid \mathtt{piz}(p) \wedge \gamma(p)], \\ \underline{w})\underline{\wedge}\mathtt{Pa}(e, p) \end{array}\right.}{\begin{array}{l} w{:}\mathsf{d} \\ \underline{w}{:}\mathbf{e} \end{array} \vdash \left\{\begin{array}{l} (\varepsilon \mid \text{ate} \mid w \text{ a pizza}) \\ \mathsf{k} \setminus \mathsf{t} \\ \mathbf{e} \to \mathbf{v} \to \mathbf{t} \\ \lambda y_2 \lambda e. \\ \mathtt{eat}(e, \mu\gamma.[p \mid \mathtt{piz}(p) \wedge \gamma(p)], \underline{w}) \\ \underline{\wedge}\mathtt{Pa}(e, p)\underline{\wedge}\mathsf{P}(e) \\ \underline{\wedge}\mathtt{Ag}(e, y_2) \end{array}\right.}\ hdr$$

A variable is introduced to enable the subsequent movement of the subject. Semantically, the application assigns the agent role to the subject.

$$\frac{\begin{array}{l} y{:}k \\ \underline{y}{:}\mathbf{e} \end{array} \vdash \left\{\begin{array}{l} (\varepsilon \mid y \mid \varepsilon) \\ \mathsf{k} \\ \mathbf{e} \\ \underline{y} \end{array}\right. \quad \begin{array}{l} w{:}\mathsf{d} \\ \underline{w}{:}\mathbf{e} \end{array} \vdash \left\{\begin{array}{l} (\varepsilon \mid \text{ate} \mid w \text{ a pizza}) \\ \mathsf{k} \setminus \mathsf{t} \\ \mathbf{e} \to \mathbf{v} \to \mathbf{t} \\ \lambda y_2 \lambda e. \\ \mathtt{eat}(e, \mu\gamma.[p \mid \mathtt{piz}(p) \wedge \gamma(p)], \underline{w}) \\ \underline{\wedge}\mathtt{Pa}(e, p) \underline{\wedge}\mathtt{P}(e) \\ \underline{\wedge}\mathtt{Ag}(e, y_2) \end{array}\right.}{\begin{array}{l} w{:}\mathsf{d}, \mathsf{y}{:}\mathsf{k} \\ \underline{w}{:}\mathbf{e}, \underline{y}{:}\mathbf{e} \end{array} \vdash \left\{\begin{array}{l} (y \mid \text{ate} \mid w \text{ a pizza}) \\ \mathsf{t} \\ \mathbf{v} \to \mathbf{t} \\ \lambda e. \\ \mathtt{eat}(e, \mu\gamma. \\ \quad [p \mid \mathtt{piz}(p) \wedge \gamma(p)], \\ \underline{w}) \\ \underline{\wedge}\mathtt{Pa}(e, p) \underline{\wedge}\mathtt{P}(e) \\ \underline{\wedge}\mathtt{Ag}(e, \underline{y}) \end{array}\right.}\ mg$$

The subject that is the dp we derived in figure 6, is inserted in the main derivation by a move. Replacing the variables attracts the subject to the left most position, and on the semantic side, the $\lambda\mu$-DRS associated with the dp replaces the variable $\underline{w}$, while the other semantic variable $\underline{y}$ is replaced with the specific discourse referent d.

$$\frac{\vdash \left\{\begin{array}{l} (\varepsilon \mid \text{the} \mid \text{children}) \\ \mathsf{k} \otimes \mathsf{d} \\ \mathbf{e} \\ \mu\delta.[\,\mid\mid [d \mid \mathtt{child}(d)] \\ \quad \Rightarrow [\,\mid \delta(d)]] \end{array}\right. \quad \begin{array}{l} w{:}\mathsf{d}, \mathsf{y}{:}\mathsf{k} \\ \underline{w}{:}\mathbf{e}, \underline{y}{:}\mathbf{e} \end{array} \vdash \left\{\begin{array}{l} (y \mid \text{ate} \mid w \text{ a pizza}) \\ \mathsf{t} \\ \mathbf{v} \to \mathbf{t} \\ \lambda e. \\ \mathtt{eat}(e, \mu\gamma. \\ \quad [p \mid \mathtt{piz}(p) \wedge \gamma(p)], \\ \underline{w}) \\ \underline{\wedge}\mathtt{Pa}(e, p) \underline{\wedge}\mathtt{P}(e) \\ \underline{\wedge}\mathtt{Ag}(e, \underline{y}) \end{array}\right.}{\vdash \left\{\begin{array}{l} (\text{the children} \mid \text{ate} \mid \text{a pizza}) \\ \mathsf{t} \\ \mathbf{v} \to \mathbf{t} \\ \lambda e.\mathtt{eat}(e, \mu\gamma.[p \mid \mathtt{piz}(p) \wedge \gamma(p)], \\ \quad \mu\delta.[\,\mid\mid [d \mid \mathtt{child}(d)] \Rightarrow [\,\mid \delta(d)]]) \\ \underline{\wedge}\mathtt{Pa}(e, p) \underline{\wedge}\mathtt{P}(e) \underline{\wedge}\mathtt{Ag}(e, d) \end{array}\right.}\ mv$$

Finally, the complementiser turns the whole sentence into a complement by a merge, and semantically reifies the whole formula.

$$\frac{\vdash \begin{cases} (\varepsilon \mid \varepsilon \mid \varepsilon) \\ \mathsf{C} \,/\, \mathsf{t} \\ (\mathbf{v} \to \mathbf{t}) \to \mathbf{t} \\ \lambda Q.[e \mid (Q(e))] \end{cases} \quad \vdash \begin{cases} (\text{the children} \mid \text{ate} \mid \text{a pizza}) \\ \mathsf{t} \\ \mathbf{v} \to \mathbf{t} \\ \lambda e.\mathtt{eat}(e, \mu\gamma.[p \mid \mathtt{piz}(p) \wedge \gamma(p)], \\ \quad \mu\delta.[\,\mid [d \mid \mathtt{child}(d)] \Rightarrow [\,\mid \delta(d)]]) \\ \underline{\wedge}\mathtt{Pa}(e,p)\underline{\wedge}\mathsf{P}(e)\underline{\wedge}\mathtt{Ag}(e,d) \end{cases}}{\vdash \begin{cases} (\varepsilon \mid \varepsilon \mid \text{the children ate a pizza}) \\ \mathsf{C} \\ \mathbf{t} \\ [e \mid \mathtt{eat}(e, \mu\gamma.[p \mid \mathtt{piz}(p) \wedge \gamma(p)], \\ \quad \mu\delta.[\,\mid [d \mid \mathtt{child}(d)] \Rightarrow [\,\mid \delta(d)]]) \\ \underline{\wedge}\mathtt{Pa}(e,p)\underline{\wedge}\mathsf{P}(e)\underline{\wedge}\mathtt{Ag}(e,d)] \end{cases}}\ mg$$

Thus the syntactic derivation yields the following $\lambda\mu$-DRS:

$[e|\mathtt{eat}(e, \mu\gamma.[p|\mathtt{piz}(p) \wedge \gamma(p)], \mu\delta.[|[d|\mathtt{child}(d)] \Rightarrow [|(\delta\ d)]])$
$\underline{\wedge}\mathtt{Pa}(e,p)\underline{\wedge}\mathsf{P}(e)\underline{\wedge}\mathtt{Ag}(e,d)]$

Yet we need to internalise the predicates, as explained in 14, i.e. to move inside the scope of quantifiers the formulae involving these variables (notice that this is harmless, because we just performed the derivation, and the variables do not have the same name by coincidence nor by renaming) and this leads to the following $\lambda\mu$-DRS:

$[e|\mathtt{eat}(e,$
$\mu\gamma.[p|\mathtt{piz}(p) \wedge \gamma(p) \wedge \mathtt{Pa}(e,p)],$
$\mu\delta.[|[d|\mathtt{child}(d)] \Rightarrow [|\delta(d) \wedge \mathtt{Ag}(e,d)]])$
$\wedge \mathsf{P}(e)]$

For computing this term, we can drop the final $\wedge\mathsf{P}(e)$ and the e box, since it is not going to interfere with the μ reductions. But we must be slightly cautious and write the $\lambda\mu$-DRS in the λ applicative style $(((\mathtt{eat}\, e)p)d)$ rather than predicate logic, to make μ reduction more visible.

$W = (((\mathtt{eat}\ e)(\mu\gamma.[p|(\mathtt{piz}\ p) \wedge (\gamma\ p) \wedge ((\mathtt{Pa}\ e)p)))(\mu\delta.Y))$
with
$Y = [|[d|(\mathtt{child}\ d)] \Rightarrow [|(\delta\ d) \wedge ((\mathtt{Ag}\ e)d)]]$

We now have to reduce acording to the reduction rules in 13 this $\lambda\mu$-term and this will lead to the two readings. The first one is that

the children shared a pizza, while the second one says that for every child there is a pizza that he ate.

(i) μ' reduction with $\alpha = \gamma$ (μ-variable), $V = (\texttt{eat}\ e)$ and $(\alpha\ T) = (\gamma\ p)$ replaced by $(V(\alpha'\ T)) = (\gamma'\ ((\texttt{eat}\ e)p))$ yields:

$W \rightsquigarrow (\mu\gamma'.[p|(\texttt{piz}\ p) \wedge (\gamma'\ ((\texttt{eat}\ e)p))\wedge((\texttt{Pa}\ e)p))(\mu\delta.Y)$

(ii) A μ reduction with $\alpha = \gamma'$ (μ-variables), $V = (\mu\delta.Y)$ and $(\alpha\ T) = (\gamma'\ ((\texttt{eat}\ e)p))$ replaced by $((\alpha'\ T)V) = ((\gamma''\ ((\texttt{eat}\ e)p))(\mu\delta.Y))$ yields:

$W \rightsquigarrow (\mu\gamma''.[p|(\texttt{piz}\ p)\wedge((\gamma''\ (((\texttt{eat}\ e)p)(\mu\delta.Y)))\wedge((\texttt{Pa}\ e)p))])$

(iii) Next we apply a ς-rule on γ'', yielding:

$W \rightsquigarrow [p|(\texttt{piz}\ p) \wedge \underbrace{(((\texttt{eat}\ e)p)(\mu\delta.Y))}_{X} \wedge((\texttt{Pa}\ e)p)]$

(iv) The underbraced subterm:

$X = (((\texttt{eat}\ e)p)(\mu\delta.Y))$ is actually:

$(((\texttt{eat}\ e)p)(\mu\delta.[|[d|(\texttt{child}\ d)]\Rightarrow[|(\delta\ d)\wedge((\texttt{Ag}\ e)d)]]))$ and it can undergo a μ'-reduction with $\alpha = \delta$ (μ-variable) and $V = ((\texttt{eat}\ e)p)$ in which $(\alpha\ T) = (\delta\ d)$ is replaced with $(\alpha'\ (V\ T)) = (\delta'\ '(((\texttt{eat}\ e)p)d))$, thus yielding:

$X \rightsquigarrow (\mu\delta'.[|[d|(\texttt{child}\ d)]\Rightarrow[|(\delta'\ (((\texttt{eat}\ e)p)d))\wedge((\texttt{Ag}\ e)d)]])$

(v) finally a simplification by the ς-rule yields:

$X \rightsquigarrow [|[d|(\texttt{child}\ d)]\Rightarrow[|(((\texttt{eat}\ e)p)d)\wedge((\texttt{Ag}\ e)d)]]$

If we insert the result of X into the result of W and add the final $\texttt{P}(e)$ we obtain: $[e|[p|(\texttt{piz}\ p) \wedge [|[d|(\texttt{child}\ d)]\Rightarrow$
$[|(((\texttt{eat}\ e)p)d)\wedge((\texttt{Ag}\ e)d)]]\wedge((\texttt{Pa}\ e)p)] \wedge \texttt{P}(e)]$
that is $\exists e.\texttt{P}(e) \wedge$
$\exists p\,(\texttt{piz}(p)\wedge\texttt{Pa}(e,p) \wedge \forall d\,(\texttt{child}(d) \Rightarrow (\texttt{eat}(e,p,d) \wedge \texttt{Ag}(e,d))))$

As $\lambda\mu$-reduction is not convergent, other readings may be obtained by applying different reductions when possible. For instance, after step 1 for which there is no choice, yielding
$W \rightsquigarrow (\mu\gamma'.Z)(\mu\delta.[|[d|(\texttt{child}\ d)]\Rightarrow[|(\delta\ d)\wedge((\texttt{Ag}\ e)d)]])$
with $Z = [p|(\texttt{piz}\ p) \wedge (\gamma'\ ((\texttt{eat}\ e)p))\wedge((\texttt{Pa}\ e)p)]$
instead of the μ reduction used in the first derivation, we can apply a μ' rule.

(ii) A μ'-rule with $\alpha = \delta$, and $V = (\mu\gamma'.Z)\ (\alpha\ T) = (\delta\ d)$ replaced with $(\alpha'\ (V\ T)) = (\delta'\ ((\mu\gamma'.Z)d))$ yields:

$W \rightsquigarrow \mu\delta'.[\|[d|(\texttt{child}\ d)]\Rightarrow[|(\delta'\ ((\mu\gamma'.\ Z)d))\wedge((\texttt{Ag}\ e)d)]]$

(iii) next we can apply a simplification ς rule for $\mu\delta'$ yielding:

$W \rightsquigarrow [\|[d|(\texttt{child}\ d)]\Rightarrow[|\underbrace{((\mu\gamma'.Z)d)}_{S}\wedge((\texttt{Ag}\ e)d)]]$

- the underbraced subterm $S = ((\mu\gamma'.Z)d)$ that is

  $S = ((\mu\gamma'.[p|(\texttt{piz}\ p) \wedge (\gamma'\ ((\texttt{eat}\ e)p))\wedge((\texttt{Pa}\ e)p)])d)$

  can undergo a μ-reduction with $\alpha = \gamma', \beta = \gamma''$ (μ-variables) and $V = d$ in which $(\alpha T) = (\gamma'\ ((\texttt{eat}\ e)p))$ replaced with $(\alpha(TV)) = (\gamma'\ (((\texttt{eat}\ e)p)d))$ yielding:

  $S \rightsquigarrow (\mu\gamma''.[p|(\texttt{piz}\ p) \wedge ((\gamma''\ ((\texttt{eat}\ e)p))d)\wedge((\texttt{Pa}\ e)p)])$

(iv) next we can apply to S a $\mu\gamma''$ simplification, yielding:

$S \rightsquigarrow [p|(\texttt{piz}\ p) \wedge (((\texttt{eat}\ e)p)d)\wedge((\texttt{Pa}\ e)p)]$

If we insert the reduct of S into what we reduced W to, we get:
$W \rightsquigarrow$
$[\|[d|(\texttt{child}\ d)]$
$\Rightarrow[\|[p|(\texttt{piz}\ p) \wedge (((\texttt{eat}\ e)p)d)\wedge((\texttt{Pa}\ e)p)]\wedge((\texttt{Ag}\ e)d)]]$

If we add the $[e|\texttt{P}(e) \wedge \cdots$ that has been left out, we obtain: $\exists e\ \texttt{P}(e) \wedge \forall d\texttt{child}(d) \Rightarrow \texttt{Ag}(e,d) \wedge \exists p(\texttt{piz}(p) \wedge \texttt{eat}(e,p,d) \wedge \texttt{Pa}(e,p))$

7. Perspectives

The purpose of this paper was to describe a state of our work on categorial representation of generative grammars, which enables the automated computation of semantic representation, here viewed as DRSs. Nevertheless, some intriguing questions remain.

On the syntactic side, what would be a proof theoretical version of shortest move? As such conditions involve the history of proofs, that are proof terms, dependant types should provide a way to formulate such condition, but for linear logic they have not yet been much studied. Still on the syntactic side, representation of minimalist grammars with remnant movement (which leaves out head-movement) for which the correspondence is tighter by our work

should be developed on particular phenomena, like VP rolls from Hungarian: as derivation intends to be quite lengthy, possibly some predetermined sequences or rules should be used, if they are proof-theoretically meaningful.

On the semantic side, some aspects deserve further study and improvement. The internalisation process, although harmless because it is performed just after parsing, is not to be recommended. Indeed, it depends on the name of the variable and modifies variable binding. Correlated to this issue, rules that depend on the name of a bound variable, the one associated with a movable constituent, should be avoided.

We nevertheless hope that this movement inside the categorial community will make colleagues happy, and especially Jim Lambek.

Works Cited

1. Abramsky, Samson. 1993. Computational interpretations of linear logic. *Theoretical Computer Science* 111(1-2): 3 – 57.
2. Amblard, Maxime. 2007. *Calculs de représentations sémantiques et syntaxe générative : les grammaires minimalistes catégorielles*. Ph.D. thesis, Université Sciences et Technologies - Bordeaux I.
3. Amblard, Maxime, Alain Lecomte, and Christian Retoré. 2003. Syntax and Semantics interacting in a minimalist theory. In *Pospects and Advances in the Syntax/Semantics Interface*, edited by Denys Duchier, 17–22. Nancy.
4. Amblard, Maxime and Christian Retoré. 2007. Natural Deduction and Normalisation For Partially Commutative Linear Logic and Lambek Calculus with product. In *Computation and Logic in the Real World (Computing in Europe 2007)*, edited by S. Barry Cooper, Thomas F. Kent, Benedikt Löwe, and Andrea Sorbi, *Quaderni del Dipartimento di Scienze Matematiche e Informatiche* Roberto Magari, vol. ID487, 28–35. Università degli Studi di Siena.
5. Anoun, Houda. 2007. *Approche logique des grammaires pour les langues naturelles*. Ph.D. thesis, Université Sciences et Technologies - Bordeaux I.

6. Bar-Hillel, Yehoshua. 1953. A quasi arithmetical notation for syntactic description. *Language* 29: 47–58.
7. Baratin, Marc and Françoise Desbordes. 1981. *L'analyse linguistique dans l'antiquité classique – I Les théories*. Klincksieck.
8. Bonato, Roberto and Christian Retoré. 2001. Learning rigid Lambek grammars and minimalist grammars from structured sentences. In *Proceedings of the third workshop on Learning Language in Logic, LLL 01*, edited by Luboš Popelìnskỳ and Miloslev Nepil, no. FI-MU-RS-2001-08 in FI MU Report series, 23–34. Strabourg: Faculty of Informatics – Masaryk University.
9. Brody, Michael. 1994. *Lexico-logical form: a radically minimalist theory*. No. 27 in Linguistic Inquiry Monographs. Cambridge, Massachusetts: M.I.T. Press.
10. Chomsky, Noam. 1993. A minimalist program for linguistic theory. In *The view from building 20*, edited by K. Hale and S.J. Keyser, 1–52. Cambridge, MA: MIT Press.
11. Cornell, Thomas. 1999. Derivational and Representational views of minimalist transformational grammar. In *Logical aspects of computational linguistics: Second International Conference, LACL '97, Nancy, France, September 22–24, 1997; selected papers*, edited by Alain Lecomte, François Lamarche, and Guy Perrier, vol. 1582, 92–111. Springer-Verlag.
12. Epstein, Samuel and Robert Berwick. 1995. On the Convergence of 'Minimalist' Syntax and Categorial Grammar. In *Algebraic Methods in Language Processing*, edited by A. Nijholt, G. Scollo, and R. Steetkamp. Universiteit Twente.
13. Girard, Jean-Yves. 1987. Linear Logic. *Theoretical Computer Science* 50(1): 1–102.
14. de Groote, Philippe. 1996. Partially commutative linear logic: sequent calculus and phase semantics. In *Third Roma Workshop: Proofs and Linguistics Categories – Applications of Logic to the analysis and implementation of Natural Language*, edited by Vito Michele Abrusci and Claudia Casadio, 199–208. Bologna:CLUEB.
15. —. 2001. Type raising, continuations, and classical logic. In *Thirteenth Amsterdam Colloquium*, edited by R. van Rooy and M. Stokhof, 97–101. Institute for Logic, Language and Computation, Universiteit van Amsterdam.

16. Kobele, Gregory. 2006. *Generating Copies: An investigation into Structural Identity in Language and Grammar*. Ph.D. thesis, UCLA.
17. Kobele, Gregory, Christian Retoré, and Sylvain Salvati. 2007. An automata-theoretic approach to minimalism. In *Model Theoretic Syntax at 10 -ESSLLI 2007 Workshop*, edited by Foundation of Logic Language and Information.
18. Lambek, Joachim. 1958. The mathematics of sentence structure. *American mathematical monthly* 154–170.
19. Lecomte, Alain. 2008. Semantics in Minimalist Categorial Grammar. In *Formal Grammar 2008*. CSLI Publications.
20. Lecomte, Alain and Christian Retoré. 1999. Towards a Minimal Logic for Minimalist Grammars: a Transformational Use of Lambek Calculus. In *Formal Grammar, FG'99*, 83–92. FoLLI.
21. —. 2001. Extending Lambek grammars: a logical account of minimalist grammars. In *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics, ACL 2001*, 354–361. Toulouse: ACL.
22. —. 2002. Bi-grammars for the syntax-semantics interface. In *Proceedings Formal Grammar 2002*, edited by Gehrard Jäger, Paola Monachesi, Gerald Penn, and Shuly Winter, 77–90. FoLLI.
23. Michaelis, Jens. 2001a. Derivational minimalism is mildly context sensitive. In *Logical Aspects of Computational Linguistics, LACL'98, selected papers*, edited by Michael Moortgat, no. 2014 in LNCS/LNAI, 179–198. Springer-Verlag.
24. —. 2001b. Transforming linear context-free rewriting systems into minimalist grammars. In *Logical Aspects of Computational Linguistics, LACL'2001*, edited by Philippe de Groote, Glyn Morrill, and Christian Retoré, no. 2099 in LNCS/LNAI, 228–244. Springer-Verlag.
25. Mönnich, Uwe, Jens Michaelis, and Frank Morawietz. 2000. Derivational Minimalism in Two Regular and Logical Steps. In *TAG+5*.
26. Moortgat, Michael. 1988. *Categorial Investigations*. Dordrecht: Foris.
27. Muskens, Reinhard. 1996. Combining Montague Semantics and Discourse Representation. *Linguistics and Philosophy* 19: 143–186.

28. Retoré, C. and E. Stabler, eds. 2004a. *Special Issue on Resource Logics and Minimalist Grammars, Journal of Research on Language and Computation*, vol. 2(1). Kluwer.
29. Retoré, Christian. 2004. A description of the non-sequential execution of Petri nets in partially commutative linear logic. In *Logic Colloquium 99*, edited by Jan van Eijck, Vincent van Oostrom, and Albert Visser, Lecture Notes in Logic, 152–181. ASL and A. K. Peters. Complete version INRIA RR-4288 (2001) http://www.inria.fr.
30. Retoré, Christian and Edward Stabler. 2004b. Generative Grammar in Resource Logics. In Retoré and Stabler (2004a), 3–25.
31. Salvati, Sylvain. 2007. Minimalist Grammars in the Light of Logic. Work presented at the colloquium in honour of Alain Lecomte.
32. Stabler, Edward. 1997. Derivational Minimalism. In *Logical Aspects of Computational Linguistics, LACL'96*, edited by Christian Retoré, *LNCS/LNAI*, vol. 1328, 68–95. Springer-Verlag.
33. —. 1999. Remnant movement and structural complexity. In *Constraints and Resources in Natural Language Syntax and Sema ntics*, edited by Gosse Bouma, Erhard Hinrichs, Geert-Jan M. Kruijff, and Richard Oehrle, 299–326. CSLI. Distributed by Cambridge University Press.
34. Vermaat, Willemijn. 2004. The Minimalist Move Operation in a Deductive Perspective. In Retoré and Stabler (2004a), 69–85.