Linear Grammars with Labels

Alain Lecomte, Houda Anoun

FG 2006:
The 11th conference on Formal Grammar
Malaga, Spain
July 29-30, 2006

Organizing Committee: Paola Monachesi
Gerald Penn Giorgio Satta
Shuly Wintner

CENTER FOR THE STUDY
OF LANGUAGE
AND INFORMATION

Contents

1

Linear Grammars with Labels

HOUDA ANOUN & ALAIN LECOMTE

Abstract

The purpose of this paper is to show that we can work in the spirit of Minimalist Grammars by means of an undirected deductive system called $\mathcal{LGL}$, enhanced with constraints on the use of assumptions. Lexical entries can be linked to sequences of controlled hypotheses which represent intermediary sites. These assumptions must be introduced in the derivation and then discharged in tandem by their proper entry which will therefore manage to find its final position: this allows to logically simulate *move* operation. Relevance of this formalism will be stressed by showing its ability to analyze difficult linguistic phenomena in a neat fashion.

Keywords LOGICAL GRAMMARS, MINIMALIST PROGRAM, SYNTAX/SEMANTICS INTERFACE, NON-LINEAR PHENOMENA

1.1 Introduction

Type Logical Grammars (Lambek (1958), Moortgat (1997)) and Minimalist Grammars (Chomsky (1995), Stabler (1997)) are two thriving theories dedicated to natural language analysis. Each one has its intrinsic assets. In fact, the first framework is computationally attractive as it works compositionally and gives the semantics for free. While the second one is based upon a reduced number of rules guaranteeing processing efficiency (Harkema (2000)). Despite their apparent differences, these theories share the same philosophy: they are both lexicalized and present universal sets of rules that allow to explain various linguistic phenomena in multitude of natural languages.

Our goal is to bridge the gap between Categorial and Minimalist Grammars by proposing a new logical formalism $\mathcal{LGL}$ (i.e. Linear Grammars with Labels) which captures Minimalist operations (i.e. *merge* and *move*) in a deductive setting. This match between logical framework and Minimalist Program proves to be fruitful as it gives a better understanding of the different

FG-2006.
Organizing Committee:, Paola Monachesi, Gerald Penn, Giorgio Satta, Shuly Wintner.

mechanisms involved in Minimalist derivations.
Lecomte, A. and Retoré, C. have already proposed a logical system that simulates Minimalist Grammars: Lecomte and Retore (2001). This latter system is built upon elimination rules for both the slashes and the tensor. The absence of any form of introduction rules leads to an efficient system. However, this restriction is not beneficial insofar as it violates the correspondence between syntactic types and semantic representations. In our new proposal, we want to keep a tranparent interface between syntax and semantics by reintroducing abstraction rules which are applied in a controlled fashion.

Like Abstract Grammars and Lambda-Grammars (de Groote (2001) and Muskens (2003)), $\mathcal{LGL}$ grammars are based upon an undirected logical system which has two interfaces (syntactic-phonetic, syntactic-semantics) owing to *Curry-Howard* correspondence. A syntactic derivation is then a deductive proof of a given sequent built using appropriate inference rules. Both phonetic form and semantic representation result from λ-terms combination which is carried out in parallel with the syntactic derivation, since each deductive rule encapsulates a computational step within the simply typed λ-calculus.

The originality of $\mathcal{LGL}$ stems from the refinement introduced in hypothetical reasoning. Our model aims at preserving the advantages of this technique (e.g. dealing with unbounded dependencies) while constraining its use in order to reduce the size of the search space. Thus, instead of considering freely accessible logical axioms, our system is equipped with finite sequences of consumable controlled hypotheses which are attached to certain lexical entries that are expected to move. Such linked hypotheses represent original sites occupied by their associated entry in the D-structure (i.e. before the displacement operation). They should be introduced during the derivation and then abstracted at the same time by their proper entry which will consequently reach its target. In the case of overt constituent movement, intermediary positions occupied by non-pronounced variables will be systematically replaced by phonetically-empty traces.
In this paper, we will prove that *move* is a metaphoric notion which can be rigorously formalized using Logic. Moreover, we will show how to capture complex linguistic phenomena (e.g. binding, discontinuity) within $\mathcal{LGL}$ thanks to the combination between Logic power and Minimalist Program ideas.

1.2 Bases of $\mathcal{LGL}$

1.2.1 Types & Terms

In this section, we survey the relevant bases inherent to $\mathcal{LGL}$.
Following earlier proposal by Curry, HB. in Curry (1961) and other more recent research work: de Groote (2001), Muskens (2003), our system distinguish between two fundamental levels of grammar. The first level is an

abstract language (tectogrammar) which encapsulates universal *principles*. The second level is a *concrete* one which may contain a range of components (e.g. phenogrammar, semantics) used to encode cross-linguistic variation (e.g. word order, lexical semantics).

Our core logic operates on abstract syntactic types which are inductively defined as follows:

$$\mathcal{T}(\mathcal{A}) := \mathcal{A} \mid \mathcal{T} \multimap \mathcal{T} \mid !\mathcal{T}$$

$\mathcal{A}$ is a finite set of atomic types that contains usual primitives of minimalist grammars (e.g. **c** (sentence), $\mathbf{d}_{acc}$ (noun phrase with accusative case), $\mathbf{d}_{nom}$ (noun phrase with nominative case). Composite types are built using the linear implication $\multimap$ and the exponential operator ! introduced in Girard (1987).

Our framework supports a two-dimensional concrete level dealing respectively with phonetics and semantics. Therefore, we consider two kinds of concrete types, namely Φ-types ($\mathcal{T}_\Phi$) and λ-types ($\mathcal{T}_\lambda$) whose definitions are the following:

$$\mathcal{T}_\Phi := s \mid \mathcal{T}_\Phi \multimap \mathcal{T}_\Phi$$

$$\mathcal{T}_\lambda := e \mid t \mid \mathcal{T}_\lambda \to \mathcal{T}_\lambda$$

The set $\mathcal{T}_\Phi$ is composed of only one atomic type **s** which represents phonetic structures (structured trees), whereas $\mathcal{T}_\lambda$ contains two primitives **e** (individuals) and **t** (truth values). Notice that composite Φ-types are built upon linear implication $\multimap$, whereas composite λ-types use intuitionistic implication $\to$. Both phonetic and semantic representation of expressions are easily defined owing to λ-calculus, thus leading to two sets of terms, namely Φ-terms Λ_Φ and λ-terms Λ_λ. Let Σ be a finite set of phonetic constants and C a finite set of semantic constants. Let $\mathcal{V}_\Phi$ (resp. $\mathcal{V}_\lambda$) be an infinite countable set of typed phonetic (resp. semantic) variables. The set $\Lambda_\Phi(\Sigma)$ of well-typed linear Φ-terms is inductively defined as follows:

1. $\epsilon \in \Lambda_\Phi(\Sigma)$ and ϵ is of type **s**[1]
2. if $\phi \in \Sigma$ then $\phi \in \Lambda_\Phi(\Sigma)$ and ϕ is of type **s**
3. if $(x_\Phi : t_\Phi) \in \mathcal{V}_\Phi$ then $x_\Phi \in \Lambda_\Phi(\Sigma)$
4. if s_1 and s_2 are Φ-terms of type **s** then $s_1 \bullet s_2 \in \Lambda_\Phi(\Sigma)$ and it is of type **s** ($\bullet$ operator is used to combine phonetic structures, it is neither associative nor commutative)
5. if ϕ_1 and ϕ_2 are Φ-terms of types t_1 and $t_1 \multimap t_2$ with no common free variable then $(\phi_1\ \phi_2) \in \Lambda_\Phi(\Sigma)$ and is of type t_2
6. if x_Φ is a variable of type t_1, ϕ_1 a Φ-term of type t_2 and x_Φ occurs free exactly once in ϕ_1 then $(\lambda x.\ \phi_1) \in \Lambda_\Phi(\Sigma)$ and has type $t_1 \multimap t_2$

[1] ϵ represents a phonetically empty element used for traces

$\Lambda_\Phi(\Sigma)$ is provided with the usual relation of β-reduction $\overset{\beta}{\Rightarrow}$ enhanced with two additional rewriting rules: $\phi_1 \bullet \epsilon \overset{\beta}{\Rightarrow} \phi_1$ and $\epsilon \bullet \phi_1 \overset{\beta}{\Rightarrow} \phi_1$.
On the other hand, the set $\Lambda_\lambda(C)$ of λ-terms is defined using a simply typed λ-calculus with two basic operations, namely intuitionistic application and abstraction.

Finally, let $\tau_{\lambda at}$ be a function which assigns a λ-type to each atomic abstract type (we assume for instance that: $\tau_{\lambda at}(\mathbf{c})=\mathbf{t}$, $\tau_{\lambda at}(\mathbf{n})= \mathbf{e}\rightarrow\mathbf{t}$, $\tau_{\lambda at}(\mathbf{d}_{case})= \mathbf{e}$). Two homomorphisms τ_Φ and τ_λ are defined to link abstract types to concrete types as follows:

τ_Φ	τ_λ
$\forall$ t$\in \mathcal{A}$, τ_Φ(t)=**s**	$\forall$ t$\in \mathcal{A}$, τ_λ(t)= $\tau_{\lambda at}$(t)
$\tau_\Phi(t_1 \multimap t_2)= \tau_\Phi(t_1) \multimap \tau_\Phi(t_2)$	$\tau_\lambda(t_1 \multimap t_2)= \tau_\lambda(t_1) \rightarrow \tau_\lambda(t_2)$
$\tau_\Phi(!\ t_1)=\tau_\Phi(t_1)$	$\tau_\lambda(!\ t_1)= \tau_\lambda(t_1)$

1.2.2 Lexical Entries & Controlled Hypotheses

We now introduce the notion of 2-dimensional signs which are the basic units managed by our system. Such signs are of the following form $(l_\Phi, l_\lambda) : ty$, where:

- $ty \in \mathcal{T}(\mathcal{A})$ (abstract type)
- $l_\Phi \in \Lambda_\Phi(\Sigma)$ and l_Φ is of concrete type τ_Φ(ty)
- $l_\lambda \in \Lambda_\lambda(C)$ and l_λ is of concrete type τ_λ(ty)

We distinguish between three classes of signs, namely *variable* signs (when $l_\Phi \in \mathcal{V}_\Phi$ and $l_\lambda \in \mathcal{V}_\lambda$), *constant* signs (when $l_\Phi \in \Sigma$ and $l_\lambda \in C$) and *compound* signs (when l_Φ or l_λ is a compound term).

These signs are used to define lexical entries. Lexical entries of $\mathcal{LGL}$ are proper axioms which can be coupled with prespecified sequences of controlled hypotheses. Such hypotheses will occupy intermediary sites, they should be introduced in the appropriate order and then discharged at the same time by their associated entry.
Lexical entries obey the syntax below:

$$\vdash (a_\phi\ ,\ a_\lambda) : ty \quad \dashv\!\exists \quad l_{hyps}$$

where:

- $(a_\phi\ ,\ a_\lambda) : ty$ is a 2-dimensional sign.
- l_{hyps}= ($[H_1 : t \vdash H'_1 : t], ..., [H_k : t \vdash H'_k : t]$) is a sequence of controlled axioms of length $|l_{hyps}|$=k, ($\forall$ i$\in${1..k}, H_i=($h_{\phi i}$, $h_{\lambda i}$) and H'_i=($h'_{\phi i}$, $h_{\lambda i}$) where $h_{\phi i} \in \mathcal{V}_\Phi$ (Φ-variable), $h_{\lambda i} \in \mathcal{V}_\lambda$ (λ-variable) and $h'_{\phi i} \in \Lambda_\Phi(\Sigma)$) .

Lexical entries are classified in two groups: *linked entries* (when k>0) and *free ones* (when k=0). Linked entries are coupled with non-empty sequences of controlled hypotheses. Each hypothesis is encapsulated inside an axiom

'$(h_{\phi i}\ ,\ h_{\lambda i})$:t $\vdash$ $(h'_{\phi i}\ ,\ h_{\lambda i})$:t ' which can be either logical (if $h_{\phi i}= h'_{\phi i}$) or extra-logical (if $h_{\phi i}\neq h'_{\phi i}$). Extra-logical axioms are extremely useful since they represent pronounced variables or phonetically non-empty traces stemming from displacement (e.g. pronouns: he, her ...).
The abstract type **ty** of the lexical entry should verify the following specification:

1. if k=0 then ty is an arbitrary abstract type
2. if k=1 then ty=$t_1 \multimap ... \multimap t_n \multimap$ (t$\multimap$ t')$\multimap$t"
3. otherwise ty=$t_1\multimap ... \multimap t_n \multimap$ (!$t \multimap t$')$\multimap$t"

Intuitively, the second (resp. third) point above means that our lexical entry represents a constituent that needs to merge with exactly n (n$\geq$0) expressions of types t_1 ... t_n respectively, and then move once (resp. an unspecified number of times, e.g. cyclic move) to reach its final position.
Finally, a lexicon is nothing else but a finite set of lexical entries $\{e_1, ..., e_n\}$.

Let us illustrate the previous definitions in a concrete example. If we assume that ($whom\in\Sigma$) and ($\wedge\in C$) then the phonetic behavior and the semantic representation of the relative pronoun '*whom*' can be modelled using the linked entry below:

$$\vdash \left(\begin{array}{l} \lambda\phi\ \lambda m.\ m \bullet (whom \bullet \phi(\epsilon)) \\ \lambda P\ \lambda Q\ \lambda x.\ P(x) \wedge Q(x) \end{array} \right) : (d_{acc} \multimap c) \multimap n \multimap n \ \ \multimap\!\!\!\!\exists \ \ [X : d_{acc} \vdash X : d_{acc}]$$

Our entry is linked to one hypothesis which will occupy the initial position of '*whom*', namely the object of its relative clause (e.g. *(book) whom Noam wrote* _). This assumption will be discharged afterwards by its related entry, thus guaranteeing the combination between the relative pronoun and its subordinate clause. Formal rules that manage this overt displacement will be set forth in the next section.

1.3 Logical simulation of Minimalism

1.3.1 Inference rules

Let lex=$\{e_1, e_2, ..., e_n\}$ be a lexicon. $\mathcal{LGL}$ grammar with lexicon lex is based upon a deductive logical system which deals simultaneously with two interfaces (syntactic-phonetic, syntactic-semantic).
Judgments of our calculus are sequents of the following form:

$$\Gamma \vdash (l_\Phi\ , l_\lambda) : ty\ ;\ \mathbf{E}$$

where:

- Γ the context is a finite multiset of 2-dimensional variable signs
- $(l_\Phi\ , l_\lambda) : ty$ is a 2-dimensional sign

- **E** is a finite multiset containing identifiers of all linked lexical entries that were used in the course of the derivation and whose associated assumptions are not yet discharged

Variable signs included in the context Γ correspond to controlled hypotheses that were introduced in the course of the derivation. Each hypothesis will be marked using a superscript '$\uparrow^i$' which points at the lexical entry to which the assumption is attached (e.g. $x_{\Phi}^{\uparrow^i}$: hypothesis linked to e_i entry).

The first group of $\mathcal{LGL}$ inference rules are axioms which coincide with derivations' leaves. Figure 1 shows axioms that our system supports[2].

$$\frac{e_i = (\vdash a_\phi : ty \multimap l)}{\vdash a_\phi : ty;\ \ if\ l = ()\ then\ \emptyset\ else\ \{e_i\}}\ Lex$$

$$\frac{e_i = (\vdash \_ \multimap l_{hyp}) \quad l_{hyp}[j] = (x_\phi : A \vdash y_\phi : A)}{x_\phi^{\uparrow^i} : A\ \vdash\ y_\phi : A;\ \emptyset}\ Ctrl$$

FIGURE 1 Axioms of $\mathcal{LGL}$(lex)

Our core logic includes extra-logical axioms which are extracted from lexical entries owing to rule *Lex*. If the involved entry is linked, then its identifier is added to the multiset **E**. On the other hand, our system excludes the freely accessible identity axiom. Available axioms stem from controlled hypotheses which are coupled with linked lexical entries. These axioms can be introduced in the derivations by means of *Ctrl* rule.

Linked entries in $\mathcal{LGL}$ can be attached to more than one controlled hypothesis. This specification has a very strong linguistic motivation. In fact, it can happen that a constituent occupies more than one intermediary site before reaching its target. Such phenomenon is illustrated for instance in the interrogative sentence '*Which book did John file _ without reading it*?'. In that case, the wh-element '*which book*' occupied two positions before displacement (in the D-structure), namely the complement of the verb *file* and that of the infinitive *without reading*. After movement, the first position becomes empty while the second is occupied by a pronounced variable '*it*'. At the semantic level, both these sites of origin represent the same object.
To account for such non-linear phenomena within $\mathcal{LGL}$, we use the exponential ! whose behavior is described by the usual rules of linear logic (Girard (1987)). Figure 2 presents the derived rules which are relevant to our study.

The generic process that handles the management of controlled hypotheses can be summarized as follows. On the first hand, each assumption of type

[2] For the sake of readability, we focus on the syntactic-phonetic interface

$$\frac{\Delta, x_\phi^{\uparrow^i} : B \vdash y_\phi : A;\ \mathbf{E_1}}{\Delta, x_\phi^{\uparrow^i} :!B \vdash y_\phi : A;\ \mathbf{E_1}}\ !L \qquad \frac{\Delta, x_\phi^{\uparrow^i} :!B, y_\phi^{\uparrow^i} :!B \vdash u_\phi : A;\ \mathbf{E_1}}{\Delta, b_\phi^{\uparrow^i} :!B \vdash u_\phi[x_\phi := b_\phi, y_\phi := b_\phi] : A;\ \mathbf{E_1}}\ !L^c$$

FIGURE 2 Relevant derived rules for !

ty will get the decorated type *!ty* if it is related to a linked entry e_i which is attached to more than one controlled hypothesis. This transformation is carried out by means of *!L* rule. Intuitively, this means that a hypothesis which represents only one controlled assumption (i.e. of type *ty*) is a particular case of hypotheses that encapsulate *at least* one controlled assumption (i.e. of type *!ty*). On the second hand, contraction rule $!L^c$ is applied to gather all the hypotheses linked to a specific entry e_i in one assumption. This will make it possible to abstract these hypotheses in tandem.

Now, the ground is well prepared to present our logical simulation of Minimalism. It is not difficult to simulate *merge* operation of Minimalist Grammars in a logical setting. In our case, it is nothing else but the direct $\multimap$ elimination ($\multimap$E, cf. Fig.3) which merges two Φ-terms (resp. λ-terms) by means of application operation.

$$\frac{\Gamma \vdash f_\phi : A \multimap B;\ \mathbf{E}_1 \quad \Delta \vdash a_\phi : A;\ \mathbf{E}'_1}{\Gamma, \Delta \vdash (f_\phi\ a_\phi) : B;\ \mathbf{E}_1 \cup \mathbf{E}'_1}\ \multimap E$$

$$\frac{\Gamma \vdash f_\phi : (C \multimap D) \multimap B; \{e_i\} \cup \mathbf{E}_1 \quad \Delta, c_\phi^{\uparrow^i} : C \vdash d_\phi : D; \mathbf{E}'_1}{\Gamma, \Delta \vdash (f_\phi\ (\lambda c_\phi.\ d_\phi)) : B;\ \mathbf{E}_1 \cup \mathbf{E}'_1}\ \multimap IE\ \ddagger$$

FIGURE 3 Behavior of $\multimap$ connective

Move operation is logically captured thanks to the refined elimination rule $\multimap$IE. This rule allows a constituent to reach its final position by simultaneously discharging its controlled hypotheses which occupied intermediary positions. Our logical formalization of *move* operation shares some ideas with Vermaat's one in Vermaat (1999). In fact, we both consider this operation as the combination of two phases, namely a *merge* step and a *hypothetical reasoning*[3] step (abstraction over sites of origin). Thus, the elements which are expected to move are assigned a higher order type (C$\multimap$ D)$\multimap$ B[4]. Such elements wait to merge with a constituent of type C$\multimap$ D, which results from the abstraction of the intermediary positions in the initial structure (of type D).

However, Vermaat proposal is encoded in a directional calculus: *move* operation is then captured using additional postulates which reintroduce structural

[3]The introduction rule of $\multimap$ is not freely available, it is rather encapsulated inside $\multimap$IE rule

[4]Vermaat considers only the case where D=B

flexibility in a controlled fashion. Our proposal is simpler as it is based upon a flexible undirected calculus. Moreover, it makes it possible to limit the operation of hypothetical reasoning used in displacement which is constrained to a specific amount of hypotheses explicitly given by the lexicon.

Rule $-\circ$IE cannot be applied unless the pre-condition $\ddagger$ is verified: all linked axioms coupled with the lexical entry e_i must be introduced in an appropriate order (from the right to the left of l_{hyps} sequence) during the derivation of $(\Delta, c_\phi^{\uparrow^i} : C \vdash d_\phi : D;\ \mathbf{E}'_1)$. Once these assumptions are abstracted, entry e_i regains its final position and is automatically withdrawn from the multiset of unstable lexical entries involved in the derivation.

To formalize the pre-condition $\ddagger$, we assume that each assumption $x^{\uparrow^i}$ of the context encapsulates a kind of *history* used to record some relevant data. This additional parameter does not have any impact on our logical system. It only ensures the efficiency of parsing by making the constraint $\ddagger$ easier to check. The notation $x^{\uparrow^i}\lfloor\sigma\rfloor$ is used when the history σ of the assumption $x^{\uparrow^i}$ is explicitly given. Otherwise, a function *hist()* can be applied to a given hypothesis $x^{\uparrow^i}$ to get its masked history.

Owing to the contraction rule $!L^c$, each hypothesis $x^{\uparrow^i}$ gathers a sub-set of controlled hypotheses related to entry e_i. The history of an assumption $x^{\uparrow^i}$ can then be encoded as a set of pairs of natural numbers. The first number of each pair represents the index of an involved controlled hypothesis taken from l_{hyps} sequence, while the second one is nothing else but the depth[5] of this hypothesis in the current bottom-up derivation.

Each deduction step updates the history of all assumptions included in the context. For instance, *Ctrl* rule enables the introduction of a specific controlled hypothesis of index j and initiates its history with the single pair (j, 0). On the other hand, rules of Fig.2 and Fig.3 increment[6] the depth of the previously introduced controlled hypotheses. We show below two logical rules enhanced with their explicit management of histories:

$$\frac{e_i = (\vdash \_ \dashv l_{hyp}) \quad l_{hyp}[j] = (x_\phi : A \vdash y_\phi : A)}{x_\phi^{\uparrow^i}\lfloor\{(j,0)\}\rfloor : A \ \vdash\ y_\phi : A;\ \emptyset}\ Ctrl$$

$$\frac{\Delta, x_\phi^{\uparrow^i}\lfloor\sigma_1\rfloor :!B, y_\phi^{\uparrow^i}\lfloor\sigma_2\rfloor :!B \vdash u_\phi : A;\ \mathbf{E_1}}{\Delta, b_\phi^{\uparrow^i}\lfloor\sigma_1^{++} \cup \sigma_2^{++}\rfloor :!B \vdash u_\phi[x_\phi := b_\phi, y_\phi := b_\phi] : A;\ \mathbf{E_1}}\ !L^c$$

[5] The number of deduction steps between the introduction of the hypothesis and the current state of the derivation

[6] Incrementing operation is denoted by $()^{++}$: $\{...;(k_i,d_i);...\}^{++}=\{...;(k_i,d_i+1);...\}$

Therefore, the side condition ‡ can be stated formally as follows:

$$\ddagger \quad iff \quad \begin{cases} \forall k,\ 1 \leq k \leq |\ l_{hyps}\ | \Rightarrow\ \exists!\ d \mid (k,\ d) \in hist(c_{\phi}^{\uparrow^i}) \\ \forall (k,\ d) \in hist(c_{\phi}^{\uparrow^i})\ \forall (k',\ d') \in hist(c_{\phi}^{\uparrow^i}),\ \ k < k'\ \Rightarrow\ d < d' \end{cases}$$

Finally, it is worth noticing that the constraint ‡ is signifi cant only if the considered derivations are in normal form. Therefore, the absence of both the freely accessible identity axiom and the $-\circ$I rule is necessary to the success of our approach.

1.3.2 $\mathcal{LGL}$ grammars & generated language

$\mathcal{LGL}$ grammars have two parameters, namely a lexicon and an atomic distinguished type **c**. Let $\mathcal{G}$(lex, c) be a $\mathcal{LGL}$ grammar and '**at**' an atomic syntactic type. We say that a sequence of phonetic constants l=$m_1 m_2 ... m_n$ has abstract type '**at**' within $\mathcal{G}$ (i.e. l$\in \mathcal{L}_{at}(\mathcal{G})$) iff:

$$\exists\ x_{\phi}, x_{\lambda}\ | x_{\phi} \in struct(m_1, ..., m_n)\ \wedge\ (\vdash (x_{\phi},\ x_{\lambda}) : at; \emptyset)$$

where $struct(m_1, ..., m_n)$ is the range of phonetic structures built using $\bullet$ operator and whose leaves are m_1, m_2, ..., m_n in that order.
Notice that the convergence of derivations requires the introduction and the simultaneous abstraction of all controlled assumptions related to involved lexical entries.
Finally, checking whether a sequence of phonetic constants *l* is recognized by the grammar $\mathcal{G}$ (i.e. l$\in \mathcal{L}(\mathcal{G})$) amounts to verifying that *l* has abstract type **c**.

1.3.3 Example of $\mathcal{LGL}$ derivations

This section is devoted to the study of a hybrid example '*More logicians met Godel than physicists knew him*' which involves two complex linguistic phenomena: binding and discontinuity. The analysis of these phenomena within the directional approach constitutes a real challenge for researchers. All proposed solutions are complex insofar as they led to the extension of the core logic either by defi ning new syntactic connectives (discontinuity connectives: Morrill (2000)) or by introducing additional packages of postulates as in Hendriks (1995). However, our proposal is able to capture such phenomena in an elegant fashion without using any additional material.
Our treatment of binding follows the same ideas of Kayne. R in Kayne (2002) where he argues that the antecedent-pronoun relation (e.g. between *Godel* and *him*) stems from the fact that both enter the derivation together as a doubling constituent ([Godel, him]) and are subsequently separated after movement. In our system, we account for this idea by defi ning a linked entry e_1 (cf. Fig. 4) associated with the proper noun *Godel*. This entry requires the introduction of two hypotheses (where the fi rst '*him*' is a pronounced one) which must be discharged at the same time. Therefore, e_1 entry will reach its fi nal position

thus making it possible to semantically link the pronoun with its antecedent.

Id	Φ-terms	λ-terms	Abstract types	Hyps
e_1	$\lambda\, P_\phi.\ P_\phi$(Godel)	$\lambda\, P_\lambda.\ P_\lambda$(**Godel**)	$(!d_{acc} \multimap c) \multimap c$	[X:d_{acc}⊢X:d_{acc}], [X':d_{acc}⊢him:d_{acc}]
e_2	logicians	**Logician**	n	()
e_3	physicists	**Physicist**	n	()
e_4	λx. λy. (y•(met•x))	λx. λy. **Meet**$_{Past}$(y,x)	$d_{acc} \multimap d_{nom} \multimap$ c	()
e_5	λx. λy. (y•(knew•x))	λx. λy. **Know**$_{Past}$(y,x)	$d_{acc} \multimap d_{nom} \multimap c$	()
e_6	λx. λy. λ P. λ Q. ((more•y)•Q(ϵ))• (than•(x•P(ϵ)))	$\lambda P_1.\ \lambda Q_1.\ \lambda P_2.\ \lambda Q_2.$ **More**(λx. Q_1(x)$\wedge Q_2$(x), λx. P_1(x)$\wedge P_2$(x))	n$\multimap$n$\multimap$ $(d_{nom} \multimap c) \multimap$ $(d_{nom} \multimap c) \multimap c$	()

FIGURE 4 Example of $\mathcal{LGL}$ lexicon

On the other hand, we capture discontinuity by gathering the different components of a discontinuous expression in the same lexical entry. For instance, entry e_6 defi nes the phonetic and semantic behavior of the discontinuous constituent (*more ... than*).

We present, in the following, the main steps of our example's analysis. For the sake of legibility, the bottom-up derivation tree is split into different key parts which will be commented on progressively.

$$\dfrac{\dfrac{\dfrac{}{\vdash \begin{pmatrix} \lambda x.\ \lambda y.\ y \bullet (knew \bullet x) \\ \lambda x.\ \lambda y.\ \mathbf{Know}_{Past}(y,x) \end{pmatrix} : d_{acc} \multimap d_{nom} \multimap c; \emptyset}\ Lex \qquad \dfrac{}{\begin{pmatrix} x_\Phi^{\uparrow^1} \\ x_\lambda^{\uparrow^1} \end{pmatrix} : d_{acc} \vdash \begin{pmatrix} him \\ x_\lambda \end{pmatrix} : d_{acc}; \emptyset}\ Ctrl}{\begin{pmatrix} x_\Phi^{\uparrow^1} \\ x_\lambda^{\uparrow^1} \end{pmatrix} : d_{acc} \vdash \begin{pmatrix} \lambda y.\ y \bullet (knew \bullet him) \\ \lambda y.\ \mathbf{Know}_{Past}(y, x_\lambda) \end{pmatrix} : d_{nom} \multimap c; \emptyset}\ \multimap E}{\begin{pmatrix} x_\Phi^{\uparrow^1} \\ x_\lambda^{\uparrow^1} \end{pmatrix} : d_{acc} \vdash \begin{pmatrix} \lambda Q.\ ((more \bullet logicians) \bullet Q(\epsilon)) \bullet (than \bullet (physicists \bullet (\epsilon \bullet (knew \bullet him)))) \\ \lambda Q_2.\ \mathbf{More}(\lambda x.\ \mathbf{Logician}(x) \wedge Q_2(x)\ ,\ \lambda x.\ \mathbf{Physicist}(x) \wedge \mathbf{Know}_{Past}(x, x_\lambda)) \end{pmatrix} : \begin{matrix} (d_{nom} \multimap c) \\ \multimap c; \emptyset \end{matrix}}$$

$$\begin{pmatrix} x_\Phi^{\uparrow^1} \\ x_\lambda^{\uparrow^1} \end{pmatrix} :!d_{acc} \vdash \begin{pmatrix} \lambda Q.\ ((more \bullet logicians) \bullet Q(\epsilon)) \bullet (than \bullet (physicists \bullet (\epsilon \bullet (knew \bullet him)))) \\ \lambda Q_2.\ \mathbf{More}(\lambda x.\ \mathbf{Logician}(x) \wedge Q_2(x)\ ,\ \lambda x.\ \mathbf{Physicist}(x) \wedge \mathbf{Know}_{Past}(x, x_\lambda)) \end{pmatrix} : \begin{matrix} (d_{nom} \multimap c) \\ \multimap c; \emptyset \end{matrix}$$

The derivation above starts by introducing the last controlled hypothesis (i.e. the assumption representing the accusative pronoun *him*) of the sequence attached to e_1 entry. This hypothesis, then, merges with lexical entry e_5 by means of $\multimap$E rule. On the other hand, a partial derivation is built by consecutively combining entry e_6 with entries e_3 and e_2. The resulting sequent then merges with the previous one. The last deduction step does nothing but decorating the type of the introduced hypothesis by a ! marker in order to express its ability to gather with the other controlled hypothesis linked to its proper entry. At this stage of analysis, only the second controlled hypothesis of e_1 has been used. Moreover, it was involved in exactly three deduction steps after its

introduction, so we can deduce that its current history is: hist($x^{\uparrow^1}$)={(2,3)}.

$$\dfrac{\dfrac{\dfrac{}{\vdash \begin{pmatrix} \lambda x.\ \lambda y.\ y \bullet (met \bullet x) \\ \lambda x.\ \lambda y.\ \mathbf{Meet}_{Past}(y,x) \end{pmatrix} : d_{acc} \multimap d_{nom} \multimap c; \emptyset}\ Lex \quad \dfrac{}{\begin{pmatrix} z_\Phi^{\uparrow^1} \\ z_\lambda^{\uparrow^1} \end{pmatrix} : d_{acc} \vdash \begin{pmatrix} z_\Phi \\ z_\lambda \end{pmatrix} : d_{acc}; \emptyset}\ Ctrl}{\begin{pmatrix} z_\Phi^{\uparrow^1} \\ z_\lambda^{\uparrow^1} \end{pmatrix} : d_{acc} \vdash \begin{pmatrix} \lambda y.\ y \bullet (met \bullet z_\Phi) \\ \lambda y.\ \mathbf{Meet}_{Past}(y,z_\lambda) \end{pmatrix} : d_{nom} \multimap c; \emptyset}\ \multimap E}{\begin{pmatrix} z_\Phi^{\uparrow^1} \\ z_\lambda^{\uparrow^1} \end{pmatrix} :!d_{acc} \vdash \begin{pmatrix} \lambda y.\ y \bullet (met \bullet z_\Phi) \\ \lambda y.\ \mathbf{Meet}_{Past}(y,z_\lambda) \end{pmatrix} : d_{nom} \multimap c; \emptyset}\ !L$$

In this second part of analysis, the first controlled assumption linked to e_1 entry is introduced. Then, it merges with e_4 entry which represents the past form of the transitive verb *meet*. This branch of the derivation ends by a !L step like the previous one. We can easily check that, at this point of the derivation, the history of $z^{\uparrow^1}$ assumption is nothing else but hist($z^{\uparrow^1}$)={(1,2)}.

$$\dfrac{\begin{pmatrix} x_\Phi^{\uparrow^1} \\ x_\lambda^{\uparrow^1} \end{pmatrix} :!d_{acc}, \begin{pmatrix} z_\Phi^{\uparrow^1} \\ z_\lambda^{\uparrow^1} \end{pmatrix} :!d_{acc} \vdash \begin{pmatrix} ((more \bullet logicians) \bullet (\epsilon \bullet (met \bullet z_\Phi))) \bullet (than \bullet (physicists \bullet (\epsilon \bullet (knew \bullet him)))) \\ \mathbf{More}(\lambda x.\ \mathbf{Logician}(x) \wedge \mathbf{Meet}_{Past}(x,z_\lambda)\ ,\ \lambda x.\ \mathbf{Physicist}(x) \wedge \mathbf{Know}_{Past}(x,x_\lambda)) \end{pmatrix} : c; \emptyset}{\begin{pmatrix} y_\Phi^{\uparrow^1} \\ y_\lambda^{\uparrow^1} \end{pmatrix} :!d_{acc} \vdash \begin{pmatrix} ((more \bullet logicians) \bullet (\epsilon \bullet (met \bullet y_\Phi))) \bullet (than \bullet (physicists \bullet (\epsilon \bullet (knew \bullet him)))) \\ \mathbf{More}(\lambda x.\ \mathbf{Logician}(x) \wedge \mathbf{Meet}_{Past}(x,y_\lambda)\ ,\ \lambda x.\ \mathbf{Physicist}(x) \wedge \mathbf{Know}_{Past}(x,y_\lambda)) \end{pmatrix} : c; \emptyset}\ \ldots$$

The partial derivation above stems from merging the two previously presented branches into one tree. Contraction rule is then applied to encapsulate both controlled hypotheses linked to e_1 in one assumption $y^{\uparrow^1}$. The current history of this latter compound assumption is: hist($y^{\uparrow^1}$)={(1,4) ; (2,5)}.

$$\dfrac{\dfrac{}{\vdash \begin{pmatrix} \lambda P_\Phi.P_\Phi(Godel) \\ \lambda P_\lambda.P_\lambda(\mathbf{Godel}) \end{pmatrix} : (!d_{acc} \multimap c) \multimap c; \{e_1\}}\ Lex \quad \dfrac{\begin{pmatrix} y_\Phi^{\uparrow^1} \\ y_\lambda^{\uparrow^1} \end{pmatrix} :!d_{acc} \vdash \begin{pmatrix} (more \bullet logicians) \bullet \ldots \\ \mathbf{More}(\ldots,\ldots) \end{pmatrix} : c; \emptyset}{\ldots}}{\vdash \begin{pmatrix} ((more \bullet logicians) \bullet (met \bullet Godel)) \bullet (than \bullet (physicists \bullet (knew \bullet him))) \\ \mathbf{More}(\lambda x.\ \mathbf{Logician}(x) \wedge \mathbf{Meet}_{Past}(x,\mathbf{Godel})\ ,\ \lambda x.\ \mathbf{Physicist}(x) \wedge \mathbf{Know}_{Past}(x,\mathbf{Godel})) \end{pmatrix} : c; \emptyset}\ \multimap IE$$

The whole derivation ends by simultaneously discharging controlled hypotheses linked to entry e_1 by means of $\multimap$IE rule. In fact, the application of this rule is allowed since the side-condition ‡ is entirely verified: as $y^{\uparrow^1}$'s history shows, the leftmost hypothesis linked to e_1 was introduced in the derivation after the rightmost one. The semantic representation of our sentence is computed in tandem. Indeed, the final semantics coincides with the intuitive meaning of the sentence, namely that the set of logicians who met Godel is larger than the range of physicists that knew him.

1.4 Enhancing $\mathcal{LGL}$

It is not difficult to notice that our logic is too flexible as the application of movement is not constrained. For instance, if we assign the entry below[7] to the wh-element '*which*', we can analyze both sentences **which man do you think the child of _ speaks?* and '*which man do you think John loves the child of _*?', where the first is ungrammatical.

$$\vdash \left(\begin{array}{c} \lambda m\ \lambda \phi\ (which \bullet_{<} m) \bullet_{>} \phi(\epsilon) \\ \lambda P\ \lambda Q\ \lambda x.P(x) \wedge Q(x) \end{array} \right) : n \multimap (d_{dat} \multimap c) \multimap c \ \multimap \exists\ [X : d_{dat} \vdash X : d_{dat}]$$

In fact, we need to control displacement operation to rule out extraction from islands. For that purpose, we propose to enhance $\mathcal{LGL}$ with some meta-rules encoding locality constraints (e.g. SPIC: Specifier Island Condition, SMC: Shortest Move Condition). We focus in the following on the *SPIC* defined in Koopman and Szabolcsi (2000) which stipulates that the moved element should be a member of the extraction domain (i.e. $comp^{+}$: transitive closure of the complement relation, or a specifier of a $comp^{+}$).

In order to locate the position of the head, the complement and the specifier inside a phonetic expression, we decorate the building structure connective $\bullet$ with a mode of composition taken from the set $\{<, >\}$. This mode points towards the sub-tree where the head is located: $\bullet_{<}$ (resp. $\bullet_{>}$) if the head is located on the left (resp. right) sub-tree.

A linked lexical entry which is expected to undergo an overt constituent movement has a phonetic-term that obeys the following syntax:

$$\lambda\, x_1\, ...\, \lambda\, x_n\, \lambda\, P_\Phi\, \lambda\, y_1\, ...\, \lambda\, y_k.\ \mathrm{g}(y_1, ...\, , y_k, \mathrm{f}(x_1, ...\, , x_n) \bullet_{>} P_\Phi(\epsilon))$$

In the expression above, $x_1, ...\, , x_n, y_1, ...\, , y_k$ (n ≥ 0, k ≥ 0) are Φ-variables of arbitrary types, whereas P_Φ is a Φ-variable of type s⊸s. Moreover, f (resp. g) is a function that takes n (resp. k+1) Φ-terms and builds a phonetic structure using these parameters together with constants of Σ.

Intuitively, this syntax means that our entry will firstly combine with n structures $x_1, ...\, , x_n$ by means of merge operation, thus leading to a maximal projection $f(x_1, ...\, , x_n)$. Then, the intermediary sites will be replaced by traces in the initial structure P_Φ and our maximal projection will be placed in specifier position, hence making it possible to carry out the expected movement. Finally, our resulting constituent can merge with other structures, thus yielding a complete expression (e.g. *whom* entry in section 2.2).

Notice that this syntax suits the type specification defined in section 2.2 (points 2 & 3) if we add additional conditions, namely that both types t (type of intermediary sites) and t' (type of the D-structure before movement) are atomic. The first condition (i.e. t∈$\mathcal{A}$) follows from constraints proposed by Koopman and Szabolcsi (Koopman and Szabolcsi (2000)) which forces

[7] d_{dat} represent noun phrases with dative case

moved elements to be maximal projections (i.e. complete expressions). However, the latter condition (t'$\in \mathcal{A}$) is a logical formalization of the *merge over move* principle Chomsky (1995) which stipulates that merge operation has priority over movement because of its simplicity. Therefore, a structure that will undergo move operation should be complete.
According to the syntax of phonetic terms associated with moved elements, SPIC condition can be encoded in $\mathcal{LGL}$ as a pre-condition of $\multimap$IE rule (cf. Fig 3) stipulating the inclusion of all occurrences of Φ-variable c_Φ within the extraction domain of the Φ-term d_Φ. Therefore, adding this meta-rule to $\mathcal{LGL}$ prevents us from analyzing the previous ungrammatical sentence.

1.5 Conclusion & Future Work

$\mathcal{LGL}$ is a new logical formalism which proposes a deductive simulation of Minimalist Program. Our proposal is powerful enough to describe several linguistic phenomena such as medial extraction, binding, ellipsis and discontinuity thanks to using linked lexical entries (related to controlled hypotheses). Moreover, one can solve over-generation problems caused by the freedom of displacement by adding some meta-rules encoding locality constraints.
In addition, it is not difficult to show that these grammars are richer than context free grammars as they are able to generate crossed-dependencies languages (e.g. $\{a^n b^m c^n d^m \mid n, m \geq 0\}$). In fact, this latter language is recognized by $\mathcal{LGL}$ grammar containing the lexicon below [8]:

$\vdash \epsilon$: p_i ($\forall$ i$\in${1..4})
$\vdash \lambda$x. λy. λz. λu. x$\bullet$(y$\bullet$(z$\bullet$u)): ty
$\vdash \lambda$ P. λx. λy. λz. λu. P(a$\bullet$x, y, c$\bullet$z, u): ty $\multimap$ ty
$\vdash \lambda$ P. λx. λy. λz. λu. P(x, b$\bullet$y, z, d$\bullet$u): ty $\multimap$ ty

The next direction to explore concerns the study of $\mathcal{LGL}$ formal properties: expressive power, decidability, and complexity. We also intend to build bridges between $\mathcal{LGL}$ and other well-known grammatical frameworks (e.g. Minimalist Grammar, TAGs).
Finally, we are developing a meta-linguistic toolkit using **Coq** proof assistant (Coq Team (2004)), in order to study logical properties of $\mathcal{LGL}$ grammars being enhanced with packages of meta-constraints. This toolkit can help users manage complex derivations by automatically handling some technical proofs thanks to powerful computation tools (strategies).

References

Chomsky, N. 1995. *The minimalist program*. MIT Press.

[8] In that case, $\mathcal{A}$={p_1,p_2,p_3,p_4,c} and *ty* denotes the composite type $p_1 \multimap p_2 \multimap p_3 \multimap p_4 \multimap$c

Coq Team. 2004. The coq proof assistant, reference manual, version 8.0. Tech. rep., INRIA.

Curry, HB. 1961. Some logical aspects of grammatical structures. In R. Jackobson, ed., *Symposium in Applied Mathematics*, pages 56–68.

de Groote, P. 2001. Towards abstract categorial grammars. In *39th Annual Meeting of the Association for Computational Linguistics*. Toulouse.

Girard, Jean-Yves. 1987. Linear logic. *Theoretical Computer Science* 50:1–102.

Harkema, H. 2000. A recognizer for minimalist grammars. In *IWPT*.

Hendriks, P. 1995. *Comparatives and Categorial Grammar*. Ph.D. thesis, University of Groningen, The Netherlands.

Kayne, R. 2002. Pronouns and their antecedents. In S. E. . T. Seely, ed., *Derivation and Explanation in the Minimalist Program*. Blackwell.

Koopman, H. and A. Szabolcsi. 2000. Verbal complexes. In *Current series in Linguistic Theory*. MIT Press.

Lambek, J. 1958. The mathematics of sentence structure. *American Mathematical Monthly* .

Lecomte, A and C Retore. 2001. Extending lambek grammars: a logical account of minimalist grammars. In *39th Annual Meeting of the Association for Computational Linguistics*, pages 354–362. Toulouse.

Moortgat, M. 1997. Categorial type logic. In V. B. . ter Meulen, ed., *Handbook of Logic and Language*, chap. 2. Elsevier.

Morrill, G. 2000. Type logical anaphora. Tech. rep., Universitat Politecnica, Catalunya.

Muskens, R. 2003. Language, lambdas, and logic. In *Resource Sensitivity in Binding and Anaphora*, Studies in Linguistics and Philosophy. Kluwer.

Stabler, E. 1997. Derivational minimalism. In C. Retore, ed., *Logical Aspects of Computational Linguistics*. Springer.

Vermaat, W. 1999. *Controlling Movement: Minimalism in a deductive perspective*. Master's thesis, master's thesis, Utrecht University.