# Towards Abstract Categorial Grammars

**Philippe de Groote**

LORIA UMR n$^o$ 7503 – INRIA
Campus Scientifique, B.P. 239
54506 Vandœuvre lès Nancy Cedex – France
degroote@loria.fr

## Abstract

We introduce a new categorial formalism based on intuitionistic linear logic. This formalism, which derives from current type-logical grammars, is abstract in the sense that both syntax and semantics are handled by the same set of primitives. As a consequence, the formalism is reversible and provides different computational paradigms that may be freely composed together.

## 1 Introduction

Type-logical grammars offer a clear cut between syntax and semantics. On the one hand, lexical items are assigned syntactic categories that combine via a categorial logic akin to the Lambek calculus (Lambek, 1958). On the other hand, we have so-called semantic recipes, which are expressed as typed $\lambda$-terms. The syntax-semantics interface takes advantage of the Curry-Howard correspondence, which allows semantic readings to be extracted from categorial deductions (van Benthem, 1986). These readings rely upon a homomorphism between the syntactic categories and the semantic types.

The distinction between syntax and semantics is of course relevant from a linguistic point of view. This does not mean, however, that it must be wired into the computational model. On the contrary, a computational model based on a small set of primitives that combine via simple composition rules will be more flexible in practice and easier to implement.

In the type-logical approach, the syntactic contents of a lexical entry is outlined by the following patern:

$$<\textit{atom}> : <\textit{syntactic category}>$$

On the other hand, the semantic contents obeys the following scheme:

$$<\lambda\textit{-term}> : <\textit{semantic type}>$$

This asymmetry may be broken by:

1. allowing $\lambda$-terms on the syntactic side (atomic expressions being, after all, particular cases of $\lambda$-terms),
2. using the same type theory for expressing both the syntactic categories and the semantic types.

The first point is a powerfull generalization of the usual scheme. It allows $\lambda$-terms to be used at a syntactic level, which is an approach that has been advocated by (Oehrle, 1994). The second point may be satisfied by dropping the non-commutative (and non-associative) aspects of categorial logics. This implies that, contrarily to the usual categorial approaches, word order constraints cannot be expressed at the logical level. As we will see this apparent loss in expressive power is compensated by the first point.

## 2 Definition of a multiplicative kernel

In this section, we define an elementary grammatical formalism based on the ideas presented in the introduction. This elementary formalism is founded on the multiplicative fragment of linear logic (Girard, 1987). For this reason, we call it a *multiplicative kernel*. Possible extensions based on other fragments of linear logic are discussed in Section 5.

### 2.1 Types, signature, and $\lambda$-terms

We first introduce the mathematical apparatus that is needed in order to define our notion of an abstract categorial grammar.

Let $A$ be a set of atomic types. The set $\mathscr{T}(A)$ of *linear implicative types* built upon $A$ is inductively defined as follows:

1. if $a \in A$, then $a \in \mathscr{T}(A)$;
2. if $\alpha, \beta \in \mathscr{T}(A)$, then $(\alpha \multimap \beta) \in \mathscr{T}(A)$.

We now introduce the notion of a *higher-order linear signature*. It consists of a triple $\Sigma = \langle A, C, \tau \rangle$, where:

1. $A$ is a finite set of atomic types;
2. $C$ is a finite set of constants;
3. $\tau : C \to \mathscr{T}(A)$ is a function that assigns to each constant in $C$ a linear implicative type in $\mathscr{T}(A)$.

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

1. if $c \in C$, then $c \in \Lambda(\Sigma)$;
2. if $x \in X$, then $x \in \Lambda(\Sigma)$;
3. if $x \in X$, $t \in \Lambda(\Sigma)$, and $x$ occurs free in $t$ exactly once, then $(\lambda x.\, t) \in \Lambda(\Sigma)$;
4. if $t, u \in \Lambda(\Sigma)$, and the sets of free variables of $t$ and $u$ are disjoint, then $(t\, u) \in \Lambda(\Sigma)$.

$\Lambda(\Sigma)$ is provided with the usual notion of capture avoiding substitution, $\alpha$-conversion, and $\beta$-reduction (Barendregt, 1984).

Given a higher-order linear signature $\Sigma = \langle A, C, \tau \rangle$, each linear $\lambda$-term in $\Lambda(\Sigma)$ may be assigned a linear implicative type in $\mathscr{T}(A)$. This type assignment obeys an inference system whose judgements are sequents of the following form:

$$\Gamma \vdash_\Sigma t : \alpha$$

where:

1. $\Gamma$ is a finite set of $\lambda$-variable typing declarations of the form '$x : \beta$' (with $x \in X$ and $\beta \in \mathscr{T}(A)$), such that any $\lambda$-variable is declared at most once;
2. $t \in \Lambda(\Sigma)$;
3. $\alpha \in \mathscr{T}(A)$.

The axioms and inference rules are the following:

$$\vdash_\Sigma c : \tau(c) \quad \text{(cons)}$$

$$x : \alpha \vdash_\Sigma x : \alpha \quad \text{(var)}$$

$$\frac{\Gamma, x : \alpha \vdash_\Sigma t : \beta}{\Gamma \vdash_\Sigma (\lambda x.\, t) : (\alpha \multimap \beta)} \quad \text{(abs)}$$

$$\frac{\Gamma \vdash_\Sigma t : (\alpha \multimap \beta) \quad \Delta \vdash_\Sigma u : \alpha}{\Gamma, \Delta \vdash_\Sigma (t\, u) : \beta} \quad \text{(app)}$$

### 2.2 Vocabulary, lexicon, grammar, and language

We now introduce the abstract notions of a vocabulary and a lexicon, on which the central notion of an abstract categorial grammar is based.

A *vocabulary* is simply defined to be a higher-order linear signature.

Given two vocabularies $\Sigma_1 = \langle A_1, C_1, \tau_1 \rangle$ and $\Sigma_2 = \langle A_2, C_2, \tau_2 \rangle$, a *lexicon* $\mathscr{L}$ from $\Sigma_1$ to $\Sigma_2$ (in notation, $\mathscr{L} : \Sigma_1 \to \Sigma_2$) is defined to be a pair $\mathscr{L} = \langle F, G \rangle$ such that:

1. $F : A_1 \to \mathscr{T}(A_2)$ is a function that interprets the atomic types of $\Sigma_1$ as linear implicative types built upon $A_2$;
2. $G : C_1 \to \Lambda(\Sigma_2)$ is a function that interprets the constants of $\Sigma_1$ as linear $\lambda$-terms built upon $\Sigma_2$;
3. the interpretation functions are compatible with the typing relation, *i.e.*, for any $c \in C_1$, the following typing judgement is derivable:

   $$\vdash_{\Sigma_2} G(c) : \hat{F}(\tau_1(c)),$$

   where $\hat{F}$ is the unique homomorphic extension of $F$.

As stated in Clause 3 of the above definition, there exists a unique type homomorphism $\hat{F} : \mathscr{T}(A_1) \to \mathscr{T}(A_2)$ that extends $F$. Similarly, there exists a unique $\lambda$-term homomorphism $\hat{G} : \Lambda(\Sigma_1) \to \Lambda(\Sigma_2)$ that extends $G$. In the sequel, when '$\mathscr{L}$' will denote a lexicon, it will also denote the homorphisms $\hat{F}$ and $\hat{G}$ induced by this

lexicon. In any case, the intended meaning will be clear from the context.

Condition 3, in the above definition of a lexicon, is necessary and sufficient to ensure that the homomorphisms induced by a lexicon $\mathscr{L} : \Sigma_1 \to \Sigma_2$ and any derivable judgement

$$x_0 \colon \alpha_0, \ldots, x_n \colon \alpha_n \vdash_{\Sigma_1} t : \alpha$$

the following judgement

$$x_0 \colon \mathscr{L}(\alpha_0), \ldots, x_n \colon \mathscr{L}(\alpha_n) \vdash_{\Sigma_2} \mathscr{L}(t) \colon \mathscr{L}(\alpha)$$

is derivable. This property, which is reminiscent of Montague's homomorphism requirement (Montague, 1970b), may be seen as an abstract realization of the compositionality principle.

We are now in a position of giving the definition of an abstract categorial grammar.

An abstract categorial grammar (ACG) is a quadruple $\mathscr{G} = \langle \Sigma_1, \Sigma_2, \mathscr{L}, s \rangle$ where:

1. $\Sigma_1 = \langle A_1, C_1, \tau_1 \rangle$ and $\Sigma_2 = \langle A_2, C_2, \tau_2 \rangle$ are two higher-order linear signatures; $\Sigma_1$ is called the *abstract vovabulary* and $\Sigma_2$ is called the *object vovabulary*;

2. $\mathscr{L} : \Sigma_1 \to \Sigma_2$ is a lexicon from the abstract vovabulary to the object vovabulary;

3. $s \in \mathscr{T}(A_1)$ is a type of the abstract vocabulary; it is called the *distinguished type* of the grammar.

Any ACG generates two languages, an abstract language and an object language. The *abstract language* generated by $\mathscr{G}$ ($\mathcal{A}(\mathscr{G})$) is defined as follows:

$$\mathcal{A}(\mathscr{G}) = \{t \in \Lambda(\Sigma_1) \mid \vdash_{\Sigma_1} t \colon s \text{ is derivable}\}$$

In words, the abstract language generated by $\mathscr{G}$ is the set of closed linear $\lambda$-terms, built upon the abstract vocabulary $\Sigma_1$, whose type is the distinguished type $s$. On the other hand, the *object language* generated by $\mathscr{G}$ ($\mathcal{O}(\mathscr{G})$) is defined to be the image of the abstract language by the term homomorphism induced by the lexicon $\mathscr{L}$:

$$\mathcal{O}(\mathscr{G}) = \{t \in \Lambda(\Sigma_2) \mid \exists u \in \mathcal{A}(\mathscr{G}).\ t = \mathscr{L}(u)\}$$

It may be useful of thinking of the abstract language as a set of abstract grammatical structures, and of the object language as the set of concrete forms generated from these abstract structures. Section 4 provides examples of ACGs that illustrate this interpretation.

### 2.3 Example

In order to exemplify the concepts introduced so far, we demonstrate how to accomodate the PTQ fragment of Montague (1973). We concentrate on Montague's famous sentence:

**John seeks a unicorn** (1)

For the purpose of the example, we make the two following assumptions:

1. the formalism provides an atomic type '*string*' together with a binary associative operator '$+$' (that we write as an infix operator for the sake of readability);

2. we have the usual logical connectives and quantifiers at our disposal.

We will see in Section 4 and 5 that these two assumptions, in fact, are not needed.

In order to handle the syntactic part of the example, we define an ACG ($\mathscr{G}_{12}$). The first step consists in defining the two following vocabularies:

$$\begin{aligned}\Sigma_1 = \langle\ & \{n, np, s\}, \{J, S_{re}, S_{dicto}, A, U\}, \\ & \{J \mapsto np,\ S_{re} \mapsto (np \multimap (np \multimap s)), \\ & S_{dicto} \mapsto (np \multimap (np \multimap s)), \\ & A \mapsto (n \multimap np),\ U \mapsto n\} \ \rangle\end{aligned}$$

$$\begin{aligned}\Sigma_2 = \langle\ & \{\mathit{string}\}, \{\mathbf{John}, \mathbf{seeks}, \mathbf{a}, \mathbf{unicorn}\}, \\ & \{\mathbf{John} \mapsto \mathit{string},\ \mathbf{seeks} \mapsto \mathit{string}, \\ & \mathbf{a} \mapsto \mathit{string}, \mathbf{unicorn} \mapsto \mathit{string}\} \ \rangle\end{aligned}$$

Then, we define a lexicon $\mathscr{L}_{12}$ from the abstract vocabulary $\Sigma_1$ to the object vocabulary $\Sigma_2$:

$$\begin{aligned}\mathscr{L}_{12} = \langle\ & \{n \mapsto \mathit{string}, np \mapsto \mathit{string}, \\ & s \mapsto \mathit{string}\}, \\ & \{J \mapsto \mathbf{John}, \\ & S_{re} \mapsto \lambda x.\, \lambda y.\, x + \mathbf{seeks} + y, \\ & S_{dicto} \mapsto \lambda x.\, \lambda y.\, x + \mathbf{seeks} + y, \\ & A \mapsto \lambda x.\, \mathbf{a} + x, \\ & U \mapsto \mathbf{unicorn}\} \ \rangle\end{aligned}$$

Finally we have $\mathscr{G}_{12} = \langle \Sigma_1, \Sigma_2, \mathscr{L}_{12}, s \rangle$.

The semantic part of the example is handled by another ACG ($\mathscr{G}_{13}$), which shares with $\mathscr{G}_{12}$ the same abstract language. The object language of this second ACG is defined as follows:

$$\begin{aligned}\Sigma_3 = \langle\ &\{e, t\},\\ &\{\text{JOHN}, \text{TRY-TO}, \text{FIND}, \text{UNICORN}\},\\ &\{\text{JOHN} \mapsto e,\\ &\ \text{TRY-TO} \mapsto (e \multimap ((e \multimap t) \multimap t)),\\ &\ \text{FIND} \mapsto (e \multimap (e \multimap t)),\\ &\ \text{UNICORN} \mapsto (e \multimap t)\}\ \rangle\end{aligned}$$

Then, a lexicon from $\Sigma_1$ to $\Sigma_3$ is defined:

$$\begin{aligned}\mathscr{L}_{13} = \langle\ &\{n \mapsto (e \multimap t), np \mapsto ((e \multimap t) \multimap t),\\ &\ s \mapsto t\},\\ &\{J \mapsto \lambda P.\, P\ \text{JOHN},\\ &\ S_{re} \mapsto\\ &\quad \lambda P.\, \lambda Q.\, Q\ (\lambda x.\, P\\ &\qquad (\lambda y.\ \text{TRY-TO}\ y\ (\lambda z.\ \text{FIND}\ z\ x))),\\ &\ S_{dicto} \mapsto\\ &\quad \lambda P.\, \lambda Q.\, P\\ &\qquad (\lambda x.\ \text{TRY-TO}\ x\\ &\qquad\quad (\lambda y.\, Q\ (\lambda z.\ \text{FIND}\ y\ z))),\\ &\ A \mapsto \lambda P.\, \lambda Q.\, \exists x.\, P\ x \wedge Q\ x,\\ &\ U \mapsto \lambda x.\ \text{UNICORN}\ x\}\ \rangle\end{aligned}$$

This allows the ACG $\mathscr{G}_{13}$ to be defined as $\langle \Sigma_1, \Sigma_3, \mathscr{L}_{13}, s \rangle$.

The abstract language shared by $\mathscr{G}_{12}$ and $\mathscr{G}_{13}$ contains the two following terms:

$$S_{re}\ J\ (A\ U) \quad (2) \qquad S_{dicto}\ J\ (A\ U) \quad (3)$$

The syntactic lexicon $\mathscr{L}_{12}$ applied to each of these terms yields the same image. It $\beta$-reduces to the following object term:

$$\mathbf{John} + \mathbf{seeks} + \mathbf{a} + \mathbf{unicorn}$$

On the other hand, the semantic lexicon $\mathscr{L}_{13}$ yields the *de re* reading when applied to (2):

$$\exists x.\ \text{UNICORN}\ x \wedge \text{TRY-TO JOHN}\ (\lambda z.\ \text{FIND}\ z\ x)$$

and it yields the *de dicto* reading when applied to (3):

$$\text{TRY-TO JOHN}\ (\lambda y.\ \exists x.\ \text{UNICORN}\ x \wedge \text{FIND}\ y\ x)$$

Our handling of the two possible readings of (1) differs from the type-logical account of Morrill (1994) and Carpenter (1996). The main difference is that our abstract vocabulary contains two constants corresponding to *seek*. Consequently, we have two distinct entries in the semantic lexicon, one for each possible reading. This is only a matter of choice. We could have adopt Morrill's solution (which is closer to Montague original analysis) by having only one abstract constant $S$ together with the following type assignment:

$$S \mapsto (np \multimap (((np \multimap s) \multimap s) \multimap s))$$

Then the types of $J$ and $A$, and the two lexicons should be changed accordingly. The semantic lexicon of this alternative solution would be simpler. The syntactic lexicon, however, would be more involved, with entries such as:

$$\begin{aligned}S &\mapsto \lambda x.\, \lambda y.\, x + \mathbf{seeks} + y\ (\lambda z.\, z)\\ A &\mapsto \lambda x.\, \lambda y.\, y\ (\mathbf{a} + x)\end{aligned}$$

## 3 Three computational paradigms

Compositional semantics associates meanings to utterances by assigning meanings to atomic items, and by giving rules that allows to compute the meaning of a compound unit from the meanings of its parts. In the type logical approach, following the Montagovian tradition, meanings are expressed as typed $\lambda$-terms and combine via functional application.

Dalrymple et al. (1995) offer an alternative to this applicative paradigm. They present a deductive approach in which linear logic is used as a glue language for assembling meanings. Their approach is more in the tradition of logic programming.

The grammatical framework introduced in the previous section realizes the compositionality principle in a abstract way. Indeed, it provides compositional means to associate the terms of a given language to the terms of some other language. Both the applicative and deductive paradigms are available.

### 3.1 Applicative paradigm

In our framework, the applicative paradigm consists simply in computing, according to the lexicon of a given grammar, the object image of an abstract term. From a computational point of view it amounts to performing substitution and $\beta$-reduction.

### 3.2 Deductive paradigm

The deductive paradigm, in our setting, answers the following problem: does a given term, built upon the object vocabulary of an ACG, belong to the object language of this ACG. It amounts to a kind of proof-search that has been described by Merenciano and Morrill (1997) and by Pogodalla (2000). This proof-search relies on linear higher-order matching, which is a decidable problem (de Groote, 2000).

### 3.3 Transductive paradigm

The example developped in Section 2.3 suggests a third paradigm, which is obtained as the composition of the applicative paradigm with the deductive paradigm. We call it the transductive paradigm because it is reminiscent of the mathematical notion of transduction (see Section 4.2). This paradigm amounts to the transfer from one object language to another object language, using a common abstract language as a pivot.

## 4 Relating ACGs to other grammatical formalisms

In this section, we illustrate the expressive power of ACGs by showing how some other families of formal grammars may be subsumed. It must be stressed that we are not only interested in a weak form of correspondence, where only the generated languages are equivalent, but in a strong form of correspondence, where the grammatical structures are preserved.

First of all, we must explain how ACGs may manipulate strings of symbols. In other words, we must show how to encode strings as linear $\lambda$-terms. The solution is well known: it suffices to represent strings of symbols as compositions of functions. Consider an arbitrary atomic type $*$, and define the type '*string*' to be $(* \multimap *)$. Then, a string such as 'abbac' may be represented by the linear $\lambda$-term $\lambda x.\, a\,(b\,(b\,(a\,(c\,x))))$, where the atomic strings '$a$', '$b$', and '$c$' are declared to be constants of type $(* \multimap *)$. In this setting, the empty word ($\epsilon$) is represented by the identity function ($\lambda x.\, x$) and concatenation ($+$) is defined to be functional composition ($\lambda f.\, \lambda g.\, \lambda x.\, f\,(g\,x)$), which is indeed an associative operator that admits the identity function as a unit.

### 4.1 Context-free grammars

Let $G = \langle T, N, P, S\rangle$ be a context-free grammar, where $T$ is the set of terminal symbols, $N$ is the set of non-terminal symbol, $P$ is the set of rules, and $S$ is the start symbol. We write $\mathcal{L}(G)$ for the language generated by $G$. We show how to construct an ACG $\mathscr{G}_G = \langle \Sigma_1, \Sigma_2, \mathscr{L}, S\rangle$ corresponding to $G$.

The abstract vocabulary $\Sigma_1 = \langle A_1, C_1, \tau_1\rangle$ is defined as follows:

1. The set of atomic types $A_1$ is defined to be the set of non-terminal symbols $N$.
2. The set of constants $C_1$ is a set of symbols in 1-1-correspondence with the set of rules $P$.
3. Let $c \in C_1$ and let '$X \to \omega$' be the rule corresponding to $c$. $\tau_1$ is defined to be the function that assigns the type $[\![\omega]\!]_X$ to $c$, where $[\![\cdot]\!]_X$ obeys the following inductive definition:
   (a) $[\![\epsilon]\!]_X = X$;
   (b) $[\![Y\omega]\!]_X = (Y \multimap [\![\omega]\!]_X)$, for $Y \in N$;
   (c) $[\![a\omega]\!]_X = [\![\omega]\!]_X$, for $a \in T$.

The definition of the object vocabulary $\Sigma_2 = \langle A_2, C_2, \tau_2\rangle$ is as follows:

1. $A_2$ is defined to be $\{*\}$.
2. The set of constants $C_2$ is defined to be the set of terminal symbols $T$.
3. $\tau_2$ is defined to be the function that assigns the type '*string*' to each $c \in C_2$.

It remains to define the lexicon $\mathscr{L} = \langle F, G\rangle$:

1. $F$ is defined to be the function that interprets each atomic type $a \in A_1$ as the type '*string*'.
2. Let $c \in C_1$ and let '$X \to \omega$' be the rule corresponding to $c$. $G$ is defined to be the function that interprets $c$ as $\lambda x_1 \ldots . \lambda x_n .\, |\omega|$, where $x_1 \ldots x_n$ is the sequence of $\lambda$-variables occurring in $|\omega|$, and $|\cdot|$ is inductively defined as follows:
   (a) $|\epsilon| = \lambda x.\, x$;
   (b) $|Y\omega| = y + |\omega|$, for $Y \in N$, and where $y$ is a fresh $\lambda$-variable;

(c) $|a\omega| = a + |\omega|$, for $a \in T$.

It is then easy to prove that $\mathscr{G}_G$ is such that:

1. the abstract language $\mathcal{A}(\mathscr{G}_G)$ is isomorphic to the set of parse-trees of $G$.

2. the language generated by $G$ coincides with the object language of $\mathscr{G}_G$, *i.e.*, $\mathcal{O}(\mathscr{G}_G) = \mathcal{L}(G)$.

For instance consider the CFG whose production rules are the following:

$$S \to \epsilon,$$
$$S \to aSb,$$

which generates the language $a^n b^n$. The corresponding ACG has the following abstract language, object language, and lexicon:

$$\Sigma_1 = \langle\ \{S\}, \{A, B\},\ \{A \mapsto S, B \mapsto ((S \multimap S)\}\ \rangle$$

$$\Sigma_2 = \langle\ \{*\}, \{a, b\},\ \{a \mapsto \mathit{string}, b \mapsto \mathit{string}\}\ \rangle$$

$$\mathscr{L} = \langle\ \{S \mapsto \mathit{string}\},\ \{A \mapsto \lambda x.\, x, B \mapsto \lambda x.\, a + x + b\}\ \rangle$$

### 4.2 Regular grammars and rational transducers

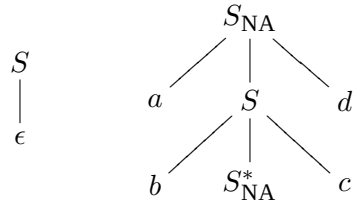
Regular grammars being particular cases of context-free grammars, they may be handled by the same construction. The resulting ACGs (which we will call "regular ACGs" for the purpose of the discussion) may be seen as finite state automata. The abstract language of a regular ACG correspond then to the set of accepting sequences of transitions of the corresponding automaton, and its object language to the accepted language.

More interestingly, rational transducers may also be accomodated. Indeed, two regular ACGs that shares the same abstract language correspond to a regular language homomorphism composed with a regular language inverse homomorphism. Now, after Nivat's theorem (Nivat, 1968), any rational transducer may be represented as such a bimorphism.

### 4.3 Tree adjoining grammars

The construction that allows to handle the tree adjoining grammars of Joshi (Joshi and Schabes, 1997) may be seen as a generalization of the construction that we have described for the context-free grammars. Nevertheless, it is a little bit more involved. For instance, it is necessary to triplicate the non-terminal symbols in order to distinguish the initial trees from the auxiliary trees.

We do not have enough room in this paper for giving the details of the construction. We will rather give an example. Consider the TAG with the following initial tree and auxiliary tree:




It generates the non context-free language $a^n b^n c^n d^n$. This TAG may be represented by the ACG, $\mathscr{G} = \langle \Sigma_1, \Sigma_2, \mathscr{L}, S \rangle$, where:

$$\Sigma_1 = \langle\ \{S, S', S''\}, \{A, B, C\},\ \{A \mapsto ((S'' \multimap S') \multimap S),\ B \mapsto (S'' \multimap ((S'' \multimap S') \multimap S')),\ C \mapsto (S'' \multimap S')\}\ \rangle$$

$$\Sigma_2 = \langle\ \{*\}, \{a, b, c, d\},\ \{a \mapsto \mathit{string}, b \mapsto \mathit{string}, c \mapsto \mathit{string}, d \mapsto \mathit{string}\}\ \rangle$$

$$\mathscr{L} = \langle\ \{S \mapsto \mathit{string}, S' \mapsto \mathit{string}, S'' \mapsto \mathit{string}\},\ \{A \mapsto \lambda f.\, f\,(\lambda x.\, x),\ B \mapsto \lambda x.\, \lambda g.\, a + g\,(b + x + c) + d,\ C \mapsto \lambda x.\, x\}\ \rangle$$

One of the keystones in the above translation is to represent an adjunction node $A$ as a functional parameter of type $A'' \multimap A'$. Abrusci et al. (1999) use a similar idea in their translation of the TAGs into non-commutative linear logic.

## 5 Beyond the multiplicative fragment

The linear $\lambda$-calculus on which we have based our definition of an ACG may be seen as a rudimentary functional programming language. The results in Section 4 indicate that, in theory, this

rudimentary language is powerful enough. Nevertheless, in practice, it would be useful to increase the expressive power of the multiplicative kernel defined in Section 2 by providing features such as records, enumerated types, conditional expressions, etc.

From a methodological point of view, there is a systematic way of considering such extensions. It consists of enriching the type system of the formalism with new logical connectives. Indeed, each new logical connective may be interpreted, through the Curry-Howard isomorphism, as a new type constructor. Nonetheless, the possible additional connectives must satisfy the following requirements:

1. they must be provided with introduction and elimination rules that satisfy Prawitz's inversion principle (Prawitz, 1965) and the resulting system must be strongly normalizable;

2. the resulting term language (or at least an interesting fragment of it) must have a decidable matching problem.

The first requirement ensures that the new types come with appropriate data constructors and discriminators, and that the associated evaluation rule terminates. This is mandatory for the applicative paradigm of Section 3. The second requirement ensures that the deductive paradigm (and consequently the transductive paradigm) may be fully automated.

The other connectives of linear logic are natural candidates for extending the formalism. In particular, they all satisfy the first requirement. On the other hand, the satisfaction of the second requirement is, in most of the cases, an open problem.

### 5.1 Additives

The additive connectives of linear logic '$\&$' and '$\oplus$' corresponds respectively to the cartesian product and the disjoint union. The cartesian product allows records to be defined. The disjoint union, together with the unit type '1', allows enumerated types and case analysis to be defined. Consequently, the additive connectives offer a good theoretical ground to provide ACG with feature structures.

### 5.2 Exponentials

The exponentials of linear logic are modal operators that may be used to go beyond linearity. In particular, the exponential '!' allows the intuitionistic implication '$\rightarrow$' to be defined, which corresponds to the possibility of dealing with non-linear $\lambda$-terms. A need for such non-linear $\lambda$-terms is already present in the example of Section 2.3. Indeed, the way of getting rid of the second assumption we made at the beginning of section 2.3 is to declare the logical symbols (*i.e.*, the existential quantifier and the conjunction that occurs in the interpretation of $A$ in Lexicon $\mathscr{L}_{13}$) as constants of the object vocabulary $\Sigma_3$. Then, the interpretation of $A$ would be something like:

$$\lambda P.\, \lambda Q.\, \textsc{exists}\, (\lambda x.\, \textsc{and}\, (P\, x)\, (Q\, x)).$$

Now, this expression must be typable, which is not possible in a purely linear framework. Indeed, the $\lambda$-term to which EXISTS is applied is not linear (there are two occurrences of the bound variable $x$). Consequently, EXISTS must be given $((e \rightarrow t) \multimap t)$ as a type.

### 5.3 Quantifiers

Quantifiers may also play a part. Uses of first-order quantification, in a type logical setting, are exemplified by Morrill (1994), Moortgat (1997), and Ranta (1994). As for second-order quantification, it allows for polymorphism.

## 6 Grammars as first-class citizen

The difference we make between an *abstract vocabulary* and an *object vocabulary* is purely conceptual. In fact, it only makes sense relatively to a given lexicon. Indeed, from a technical point of view, any vocabulary is simply a higher-order linear signature. Consequently, one may think of a lexicon $\mathscr{L}_{12} : \Sigma_1 \rightarrow \Sigma_2$ whose object language serves as abstract language of another lexicon $\mathscr{L}_{23} : \Sigma_2 \rightarrow \Sigma_3$. This allows lexicons to be sequentially composed. Moreover, one may easily construct a third lexicon $\mathscr{L}_{13} : \Sigma_1 \rightarrow \Sigma_3$ that corresponds to the sequential composition of $\mathscr{L}_{23}$ with $\mathscr{L}_{12}$. From a practical point of view, this means that the sequential composition of two lexicons may be compiled. From a theoretical point of view, it means that the ACGs form a category

whose objects are vocabularies and whose arrows are lexicons. This opens the door to a theory where operations for constructing new grammars from other grammars could be defined.

## 7 Conclusion

This paper presents the first steps towards the design of a powerful grammatical framework based on a small set of computational primitives. The fact that these primitives are well known from programming theory renders the framework suitable for an implementation. A first prototype is currently under development.

## References


M. Abrusci, C. Fouqueré, and J. Vauzeilles. 1999. Tree-adjoining grammars in a fragment of the Lambek calculus. *Computational Linguistics*, 25(2):209–236.

H.P. Barendregt. 1984. *The lambda calculus, its syntax and semantics*. North-Holland, revised edition.

J. van Benthem. 1986. *Essays in Logical Semantics*. Reidel, Dordrecht.

B. Carpenter. 1996. *Type-Logical Semantics*. MIT Press, Cambridge, Massachussetts and London England.

M. Dalrymple, M. Lamping, F. Pereira, and V. Saraswat. 1995. Linear logic for meaning assembly. In G. Morrill and D. Oehrle, editors, *Formal Grammar*, pages 75–93. FoLLI.

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

Ph. de Groote. 2000. Linear higher-order matching is NP-complete. In L. Bachmair, editor, *Rewriting Techniques and Applications, RTA'00*, volume 1833 of *Lecture Notes in Computer Science*, pages 127–140. Springer.

A. K. Joshi and Y. Schabes. 1997. Tree-adjoining grammars. In G. Rozenberg an A. Salomaa, editor, *Handbook of formal languages*, volume 3, chapter 2. Springer.

J. Lambek. 1958. The mathematics of sentence structure. *Amer. Math. Monthly*, 65:154–170.

J. M. Merenciano and G. Morrill. 1997. Generation as deduction on labelled proof nets. In C. Retoré, editor, *Logical Aspects of Computational Linguistics, LACL'96*, volume 1328 of *Lecture Notes in Artificial Intelligence*, pages 310–328. Springer Verlag.

R. Montague. 1970a. English as a formal language. In B. Visentini et al., editor, *Linguaggi nella Società e nella Tecnica*, Milan. Edizioni di Communità. Reprinted: (Montague, 1974, pages 188–221).

R. Montague. 1970b. Universal grammar. *Theoria*, 36:373–398. Reprinted: (Montague, 1974, pages 222–246).

R. Montague. 1973. The proper treatment of quantification in ordinary english. In J. Hintikka, J. Moravcsik, and P. Suppes, editors, *Approaches to natural language: proceedings of the 1970 Stanford workshop on Grammar and Semantics*, Dordrecht. Reidel. Reprinted: (Montague, 1974, pages 247–270).

R. Montague. 1974. *Formal Philosophy: selected papers of Richard Montague, edited and with an introduction by Richmond Thomason*. Yale University Press.

M. Moortgat. 1997. Categorial type logic. In J. van Benthem and A. ter Meulen, editors, *Handbook of Logic and Language*, chapter 2. Elsevier.

G. Morrill. 1994. *Type Logical Grammar: Categorial Logic of Signs*. Kluwer Academic Publishers, Dordrecht.

M. Nivat. 1968. Transduction des langages de Chomsky. *Annales de l'Institut Fourier*, 18:339–455.

R. T. Oehrle. 1994. Term-labeled categorial type systems. *Linguistic & Philosophy*, 17:633–678.

S. Pogodalla. 2000. Generation, Lambek Calculus, Montague's Semantics and Semantic Proof Nets. In *Proceedings of the $18^{th}$ International Conference on Computational Linguistics*, volume 2, pages 628–634.

D. Prawitz. 1965. *Natural Deduction, A Proof-Theoretical Study*. Almqvist & Wiksell, Stockholm.

A. Ranta. 1994. *Type theoretical grammar*. Oxford University Press.