Lambek Categorial Grammars as Abstract Categorial Grammars

Philippe de Groote

Inria Nancy - Grand Est
France

Abstract. This paper describes a construction that allows Lambek Categorial Grammars to be represented as Abstract Categorial Grammars. This construction is based upon a family of combinators that allows Lambek lexical entries to be interpreted as linear λ-terms.

1 Introduction

Abstract Categorial Grammars [2] (ACG, for short) differ from classical categorial grammars in an essential way: the ACG type system is based on a commutative logic (namely, the implicative fragment of multiplicative linear logic [4]). For this reason, it has been argued that the way of encoding wh-extraction in an ACG corresponds to an uncontroled form of extraction, which results in syntactic overgeneration. In particular, an ACG could not accomodate left and right peripheral extractions like a Lambek categorial grammar [9] (LG, for short) does.

This claim about ACG and LG is certainly not true at the level of the string languages. Indeed, Pentus' theorem [11, 12] states that every Lambek grammar can be transformed into a context-free grammar, and there is a canonical way of representing a context-free grammar as an ACG [3].

The claim is not quite true eiteher at the level of the derivations since Kanazawa and Salvati have shown that Pentus' construction preserve, in some sense, the derivations of the original Lambek grammar [6]. As a consequence, given a Lambek grammar G, it is possible to define an ACG that generates a set of λ-terms that correspond to the derivation of G. What is then needed in order to turn these derivations into strings is an appropriate way of interpreting the lexical entries of the original Lambek grammar as linear λ-terms. The main goal of this paper is to devise such an interpretation.

2 Mathematical preliminaries

Let A be a set of atomic types. The set $\mathscr{T}_A$ of the *simple types* (built upon A) is inductively defined according to the following rules:

$$\mathscr{T}_A \quad ::= \quad A \mid (\mathscr{T}_A \rightarrow \mathscr{T}_A)$$

The order of a simple type is inducively defined as follows:

1. $\mathrm{ord}(a) = 1$, for $a \in A$;
2. $\mathrm{ord}(\alpha \rightarrow \beta) = \max\{\mathrm{ord}(\alpha) + 1, \mathrm{ord}(\beta)\}$

A finite set of typed constants is called a *higher-order signature*. More formally, such a higher-order signature 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. $\tau : C \rightarrow \mathscr{T}_A$ is a function that assigns to each constant in C a simple type in $\mathscr{T}_A$.

Let $\Sigma = \langle A, C, \tau \rangle$ be a signature. The order of Σ is defined to be $\max\{\mathrm{ord}(\tau(c)) \mid c \in C\}$.

Let X be an infinite countable set of λ-variables. Given a higher-order signature $\Sigma = \langle A, C, \tau \rangle$, the set $\Lambda(\Sigma)$ of the *linear λ-terms* built upon Σ 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, α-conversion, β-reduction, and η-reduction [1]. we take the relation of $\beta\eta$-equivalence as the notion of equality between λ-terms.

Each λ-terms in $\Lambda(\Sigma)$ may be assigned a simple type according to the following type system:

$$\vdash_{\Sigma} c : \tau(c) \qquad\qquad x : \alpha \vdash_{\Sigma} x : \alpha$$

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

Given two higher-order signatures $\Sigma_1 = \langle A_1, C_1, \tau_1 \rangle$ and $\Sigma_2 = \langle A_2, C_2, \tau_2 \rangle$, a morphism $\Phi : \Sigma_1 \to \Sigma_2$ consists of an interpretation of the atomic types of Σ_1 as types built upon A_2 together with an interpretation of the constants of Σ_1 as linear λ-terms built upon Σ_2. These two interpretations must be such that their homomorphic extensions commute with the typing relations. More formally, a *morphism between higher-order signatures*, $\Phi : \Sigma_1 \to \Sigma_2$, is defined to be a pair $\Phi = \langle \eta, \theta \rangle$ such that:

1. $\eta : A_1 \to \mathscr{T}(A_2)$ is a function that interprets the atomic types of Σ_1 as simple types built upon A_2;
2. $\theta : C_1 \to \Lambda(\Sigma_2)$ is a function that interprets the constants of Σ_1 as linear λ-terms built upon Σ_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:

   $$\vdash_{\Sigma_2} \theta(c) : \hat{\eta}(\tau_1(c)),$$

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

Condition 3, in the above definition ensures that if $x_1 : \alpha_1, \ldots, x_n : \alpha_n \vdash_{\Sigma_1} t : \alpha$, then:

$$x_1 : \Phi(\alpha_1), \ldots, x_n : \Phi(\alpha_n) \vdash_{\Sigma_2} \Phi(t) : \Phi(\alpha)$$

where, according to the context, $\Phi(\cdot)$ denotes either the homomorphic extension of η or the homomorphic extension of η.

Let $\Phi : \Sigma_1 \to \Sigma_2$ be a morphism between signatures. The order of Φ is defined to be $\max\{\mathrm{ord}(\Phi(a)) \mid a \in A_1\}$.

We end this section by introducing a few additional notations that will be useful in the sequel of the paper. Let $I = \{i_1, \ldots, i_n\}$ be a totally ordered finite set of indices, and let $(x_i)_{i \in I}$ (respectively, $(\alpha_i)_{i \in I}$) be a sequence of λ-variables (respectively, simple types) indexed by I. We write $(x_i\!: \alpha_i)_{i \in I}$ for the typing environment $x_{i_1}\!: \alpha_{i_1}, \ldots, x_{i_n}\!: \alpha_{i_n}$. Similarly, we write $t[x_i\!:=\!u_i]_{i \in I}$ for the simultaneous substitution in the λ-term t of the terms $u_{i_1}, \ldots, u_{i_n}$ for the free variables $x_{i_1}, \ldots, x_{i_n}$. In this latter case, we assume that the set of variables $\{x_i \mid i \in I\}$ corresponds exactly to the set of free variables of t.

Let Σ be an alphabet. As usual, we write Σ^* for the set of strings generated by Σ. We use ϵ to denote the empty string, and the infix operator '+' to denote string concatenation. Accordingly, given a sequence of strings $(w_i)_{i \in I}$, we write $\sum_{i \in I} w_i$ for the string $w_{i_1} \ldots w_{i_n}$.

Lambek Grammar	
man	: n
woman	: n
some	: $np \, / \, n$
every	: $np \, / \, n$
loves	: $(np \setminus s) \, / \, np$
who	: $(n \setminus n) \, / (np \setminus s)$
whom	: $(n \setminus n) \, / (s \, / \, np)$

Fig. 1.

3 Lambek categorial grammars

The classical notion of a Lambek categorial grammar is based on a deductive system known as the associative Lambek calculus [9]. This calculus may be seen as a non-commutative fragment of implicative linear logic [4].

Let A be a set of atomic formulas. The syntax of the Lambek formulas (built upon A) obeys the following formation rules:

$$\mathscr{F}_A \ ::= \ A \ | \ (\mathscr{F}_A \setminus \mathscr{F}_A) \ | \ (\mathscr{F}_A \, / \, \mathscr{F}_A)$$

where formulas of the form $\alpha \setminus \beta$ correspond to left-to-right implications (i.e., α *implies* β), and formulas of the form $\alpha \, / \, \beta$ to right-to-left implications (i.e., α *is implied by* β). Lambek formulas are also called *syntactic types*.

The deduction relation is then specified by means of the following sequent calculus.

$$\alpha \vdash_L \alpha$$

$$\frac{\alpha, \Gamma \vdash_L \beta}{\Gamma \vdash_L \alpha \setminus \beta} \qquad \frac{\Gamma, \alpha \vdash_L \beta}{\Gamma \vdash_L \beta \, / \, \alpha}$$

$$\frac{\Gamma \vdash_L \alpha \quad \Delta, \beta, \Theta \vdash_L \gamma}{\Delta, \Gamma, \alpha \setminus \beta, \Theta \vdash_L \gamma} \qquad \frac{\Gamma \vdash_L \alpha \quad \Delta, \beta, \Theta \vdash_L \gamma}{\Delta, \beta \, / \, \alpha, \Gamma, \Theta \vdash_L \gamma}$$

It should be stressed that the above system does not include any structural rule. In particular, the non-commutativity of the systems is reflected by the absence of an exchange rule. This, in turn, explains the presence of two different implications.

A Lambek categorial grammar (L-grammar, for short) is defined to be a quadruple $G = \langle \Sigma, A, \mathcal{L}, s \rangle$ such that:

1. Σ is a finite set of terminal symbols;
2. A is a finite set of atomic types;
3. $\mathcal{L} : \Sigma \longrightarrow 2^{\mathscr{F}_A}$ is a *lexicon* that assigns to each terminal symbol a finite set of types built upon A;
4. $s \in A$ is a distinguished type, called the *initial type* of the grammar.

A word $a_0 a_1 \ldots a_n \in \Sigma^*$ belongs to the language generated by G if and only if there exist $\alpha_0 \in \mathcal{L}(a_0), \alpha_1 \in \mathcal{L}(a_1), \ldots \alpha_n \in \mathcal{L}(a_n)$ such that $\alpha_0, \alpha_1, \ldots \alpha_n \vdash_L s$ is derivable.

Figure 1 gives the lexicon of a Lambek categorial grammar that will serve as a running example throughout this paper. According to this grammar, the sentence "***every man who***

Context-free Grammar

$$\begin{aligned}
<np> &\rightarrow <np/n>\ <n> \\
<s> &\rightarrow <np>\ <(np\backslash s)\,/\,np>\ <np> \\
<n> &\rightarrow <n>\ <(n\backslash n)/(np\backslash s)>\ <np\backslash s> \\
<n> &\rightarrow <n>\ <(n\backslash n)/(s/np)>\ <s/np> \\
<np\backslash s> &\rightarrow <(np\backslash s)/np>\ <np> \\
<s/np> &\rightarrow <np>\ <(np\backslash s)/np> \\
<s/np> &\rightarrow <np>\ <(np\backslash s)\,/\,np>\ <np/np> \\
<np/np> &\rightarrow <np/n>\ <n/np> \\
<n/np> &\rightarrow <n>\ <(n\backslash n)/(np\backslash s)>\ <(np\backslash s)/np> \\
<(np\backslash s)/np> &\rightarrow <(np\backslash s)/np>\ <np/np> \\
<n> &\rightarrow \textbf{\textit{man}} \\
<n> &\rightarrow \textbf{\textit{woman}} \\
<np/n> &\rightarrow \textbf{\textit{some}} \\
<np/n> &\rightarrow \textbf{\textit{every}} \\
<(np\backslash s)/np> &\rightarrow \textbf{\textit{loves}} \\
<(n\backslash n)/(np\backslash s)> &\rightarrow \textbf{\textit{who}} \\
<(n\backslash n)/(s/np)> &\rightarrow \textbf{\textit{whom}}
\end{aligned}$$

Fig. 2.

loves some woman loves every woman" is grammatical because the following sequent is derivale:

$$np\,/\,n, n, (n\,\backslash\,n)\,/(np\,\backslash\,s), (np\,\backslash\,s)\,/\,np, np\,/\,n, n, (np\,\backslash\,s)\,/\,np, np\,/\,n, n \vdash_L s$$

The above example illustrates that when dealing with a categorial grammar, parsing corresponds to proof-search. Consequently, a categorial parse structure amounts to a formal derivation. Then, using the Curry-Howard correspondance, it is possible to associate a simply typed (actually, linear) λ-term to any derivation of the Lambek calculus. This is realized by the following system:

$$x : \alpha \vdash_{\lambda L} x : \alpha$$

$$\frac{x : \alpha, \Gamma \vdash_{\lambda L} t : \beta}{\Gamma \vdash_{\lambda L} \lambda x.t : \alpha\,\backslash\,\beta} \qquad \frac{\Gamma, x : \alpha \vdash_{\lambda L} t : \beta}{\Gamma \vdash_{\lambda L} \lambda x.t : \beta\,/\,\alpha}$$

$$\frac{\Gamma \vdash_{\lambda L} u : \alpha \quad \Delta \vdash_{\lambda L} t : \alpha\,\backslash\,\beta}{\Gamma, \Delta \vdash_{\lambda L} t\,u : \beta} \qquad \frac{\Gamma \vdash_{\lambda L} t : \beta\,/\,\alpha \quad \Delta \vdash_{\lambda L} u : \alpha}{\Gamma, \Delta \vdash_{\lambda L} t\,u : \beta}$$

According to Pentus' theorem [11, 12], every Lambek grammar may be turned into an equivalent context-free grammar. The context-free grammar of Figure 2, for instance, is a grammar that generates the same language as the Lambek grammar of Figure 1.

In the grammar of Figure 2, in agreement with Pentus' construction, types of the Lambek calculus are used as non-terminal symbols. In addition, every production rule $<\alpha> \rightarrow <\alpha_1> \ldots <\alpha_n>$ corresponds to a derivable sequent $\alpha_1, \ldots, \alpha_n \vdash_L \alpha$ whose derivation, in turn, corresponds to a λ-term. These sequents, together with the λ-terms encoding their derivations, are given in Figure 3.

Derivable sequents
$x : np/n,\ y : n \vdash_{\lambda L} x\, y : np$
$x : np,\ y : (np\backslash s)\, /\, np,\ z : np \vdash_{\lambda L} y\, z\, x : s$
$w : n,\ x : (n\backslash n)/(np\backslash s),\ y : np\backslash s \vdash_{\lambda L} x\, (\lambda z.\, y\, z)\, w : n$
$w : n,\ x : (n\backslash n)/(s/np),\ y : s/np \vdash_{\lambda L} x\, (\lambda z.\, y\, z)\, w : n$
$x : (np\backslash s)/np,\ y : np \vdash_{\lambda L} \lambda z.\, x\, y\, z : np\backslash s$
$x : np,\ y : (np\backslash s)/np \vdash_{\lambda L} \lambda z.\, y\, z\, x : s/np$
$w : np,\ x : (np\backslash s)\, /\, np,\ y : np/np \vdash_{\lambda L} \lambda z.\, x\, (y\, z)\, w : s/np$
$x : np/n,\ y : n/np \vdash_{\lambda L} \lambda z.\, x\, (y\, z) : np/np$
$v : n,\ w : (n\backslash n)/(np\backslash s),\ x : (np\backslash s)/np \vdash_{\lambda L} \lambda y.\, w\, (\lambda z.\, x\, y\, z)\, v : n/np$
$w : (np\backslash s)/np,\ x : np/np \vdash_{\lambda L} \lambda yz.\, w\, (x\, y)\, z : (np\backslash s)/np$

Fig. 3.

In some sense, a context-free grammar resulting from Pentus' construction preserves the parse structures of the original Lambek grammar. Consider, for instance, a sentence S belonging to the language generated by the Lambek grammar of Figure 1. Accordingly, there exists a derivation of S using the rules of the context-free grammar of Figure 2. Now, using this context-free derivation together with the λ-terms given in Figure 3 it is possible to compute the λ-term corresponding to the original derivation of S in the Lambek grammar of Figure 1. This has been shown by Kanazawa and Salvati [6].

4 Abstract categorial grammars

Abstract categorial grammars have been introduced in [2]. Contrarily to the case of most other notions of categorial grammar, they are based on a fully commutative logic.

Formally, an *abstract categorial grammar* is a quadruple $G = \langle \Sigma_1, \Sigma_2, \mathcal{L}, s\rangle$ where:

1. Σ_1 and Σ_2 are two higher-order signatures; they are called the *abstract vovabulary* and the *object vovabulary*, respectively;
2. $\mathcal{L} : \Sigma_1 \to \Sigma_2$ is a morphism between the abstract vovabulary and the object vovabulary; it is called the *lexicon*;
3. s is an atomic type of the abstract vocabulary; it is called the *distinguished type* of the grammar.

The *abstract language* generated by G, $\mathcal{A}(\mathcal{G})$, is defined as follows:

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

In words, the abstract language generated by G is the set of closed linear λ-terms, built upon the abstract vocabulary Σ_1, whose type is the distinguished type s.

The *object language* generated by G, $\mathcal{O}(\mathcal{G})$, is then defined to be the image of the abstract language by the lexicon $\mathcal{L}$:

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

The abstract language of an ACG may be thought of as its parse structures, and its object language as the language it generates. Using this intuition, it is not difficult to show that

Abstract Vocabulary

PROD_0 :	$<np/n> \to <n> \to <np>$
PROD_1 :	$<(np\backslash s)\,/\,np> \to <np> \to <np> \to <s>$
PROD_2 :	$<(n\backslash n)/(np\backslash s)> \to <np\backslash s> \to <n> \to <n>$
PROD_3 :	$<(n\backslash n)/(s/np)> \to <s/np> \to <n> \to <n>$
PROD_4 :	$<(np\backslash s)/np> \to <np> \to <np\backslash s>$
PROD_5 :	$<(np\backslash s)/np> \to <np> \to <s/np>$
PROD_6 :	$<(np\backslash s)\,/\,np> \to <np/np> \to <np> \to <s/np>$
PROD_7 :	$<np/n> \to <n/np> \to <np/np>$
PROD_8 :	$<(n\backslash n)/(np\backslash s)> \to <(np\backslash s)/np> \to <n> \to <n/np>$
PROD_9 :	$<(np\backslash s)/np> \to <np/np> \to <(np\backslash s)/np>$
MAN :	$<n>$
WOMAN :	$<n>$
SOME :	$<np/n>$
EVERY :	$<np/n>$
LOVES :	$<(np\backslash s)/np>$
WHO :	$<(n\backslash n)/(np\backslash s)>$
WHOM :	$<(n\backslash n)/(s/np)>$

Fig. 4.

Object Vocabulary

man :	n
woman :	n
some :	$n \to np$
every :	$n \to np$
loves :	$np \to np \to s$
who :	$(np \to s) \to n \to n$
whom :	$(np \to s) \to n \to n$

Fig. 5.

every context-free grammar may be represented as an ACG [3]. Accordingly, the context-free grammar of Figure 2 could be turned into an ACG. Now, using a similar construction together with the result of Kanazawa and Salvati [6], one may take advantage of the λ-terms given in Figure 3 in order to devise an ACG that generates the λ-terms that encodes the derivations of the original Lambek grammar of Figure 1. Lets call this ACG $LDER$. Its abstract vocabulary and its object vocabulary are respectively given in Figure 4 and Figure 5. Its lexicon is defined by Figure 6 and Figure 7.

While every Lambek grammar is strongly lexicalized, $LDER$ is not (in the sense that some of its abstract constants are interpreted by pure combinators). In [7, 8], Kanazawa and Yoshinaka show that every second-order ACG (i.e., an ACG whose abstract vocabulary is

Lexicon: type interpretation

$$\begin{aligned}
<n> &:= n \\
<np> &:= np \\
<s> &:= s \\
<n/np> &:= np \rightarrow n \\
<np/n> &:= n \rightarrow np \\
<np/np> &:= np \rightarrow np \\
<np\backslash s> &:= np \rightarrow s \\
<s/np> &:= np \rightarrow s \\
<(np\backslash s)/np> &:= np \rightarrow np \rightarrow s \\
<(n\backslash n)/(np\backslash s)> &:= (np \rightarrow s) \rightarrow n \rightarrow n \\
<(n\backslash n)/(s/np)> &:= (np \rightarrow s) \rightarrow n \rightarrow n
\end{aligned}$$

Fig. 6.

Lexicon: term interpretation

$$\begin{aligned}
\textsc{prod}_0 &:= \lambda xy.\, x\, y \\
\textsc{prod}_1 &:= \lambda xyz.\, x\, y\, z \\
\textsc{prod}_2 &:= \lambda wxy.\, w\, (\lambda z.\, x\, z)\, y \\
\textsc{prod}_3 &:= \lambda wxy.\, w\, (\lambda z.\, x\, z)\, y \\
\textsc{prod}_4 &:= \lambda xyz.\, x\, y\, z \\
\textsc{prod}_5 &:= \lambda xyz.\, x\, z\, y \\
\textsc{prod}_6 &:= \lambda wxyz.\, w\, (x\, z)\, y \\
\textsc{prod}_7 &:= \lambda xyz.\, x\, (y\, z) \\
\textsc{prod}_8 &:= \lambda vwxy.\, v\, (\lambda z.\, w\, y\, z)\, x \\
\textsc{prod}_9 &:= \lambda wxyz.\, w\, (x\, y)\, z \\
\textsc{man} &:= \mathsf{man} \\
\textsc{woman} &:= \mathsf{woman} \\
\textsc{some} &:= \mathsf{some} \\
\textsc{every} &:= \mathsf{every} \\
\textsc{loves} &:= \mathsf{loves} \\
\textsc{who} &:= \mathsf{who} \\
\textsc{whom} &:= \mathsf{whom}
\end{aligned}$$

Fig. 7.

second-order) can be lexicalized. As a matter of fact, $LDER$ is a second-order ACG because it derives from a context-free grammar. We may therefore lexicalize it. The resulting ACG, which we call $LDER_{lex}$ shares with $LDER$ the same object vocabulary (Fig. 5), the same set of abstract atomic types, and the same type interpretation (Fig. 6). As for the new

Abstract Vocabulary (lexicalized grammar)	
MAN	$: <n>$
WOMAN	$: <n>$
SOME	$: <n> \to <np>$
SOME$_0$	$: <n/np> \to <np/np>$
EVERY	$: <n> \to <np>$
EVERY$_0$	$: <n/np> \to <np/np>$
LOVES	$: <np> \to <np> \to <s>$
LOVES$_0$	$: <np> \to <np\backslash s>$
LOVES$_1$	$: <np> \to <s/np>$
LOVES$_2$	$: <np/np> \to <np> \to <s/np>$
LOVES$_3$	$: <np/np> \to <(np\backslash s)/np>$
LOVES$_4$	$: <(np\backslash s)/np>$
WHO	$: <np\backslash s> \to <n> \to <n>$
WHO$_0$	$: <(np\backslash s)/np> \to <n> \to <n/np>$
WHOM	$: <s/np> \to <n> \to <n>$

Fig. 8.

Lexicon (lexicalized grammar)	
SOME	$:= \lambda x.\, \mathsf{some}\, x$
EVERY	$:= \lambda x.\, \mathsf{every}\, x$
LOVES	$:= \lambda xy.\, \mathsf{loves}\, x\, y$
WHO	$:= \lambda xy.\, \mathsf{who}\, (\lambda z.\, x\, z)\, y$
WHOM	$:= \lambda xy.\, \mathsf{whom}\, (\lambda z.\, x\, z)\, y$
LOVES$_0$	$:= \lambda xy.\, \mathsf{loves}\, x\, y$
LOVES$_1$	$:= \lambda xy.\, \mathsf{loves}\, y\, x$
LOVES$_2$	$:= \lambda xyz.\, \mathsf{loves}\, (x\, z)\, y$
SOME$_0$	$:= \lambda xy.\, \mathsf{some}\, (x\, y)$
EVERY$_0$	$:= \lambda xy.\, \mathsf{every}\, (x\, y)$
WHO$_0$	$:= \lambda wxy.\, \mathsf{who}\, (\lambda z.\, w\, y\, z)\, x$
LOVE$_3$	$:= \lambda xyz.\, \mathsf{loves}\, (x\, y)\, z$
MAN	$:= \mathsf{man}$
WOMAN	$:= \mathsf{woman}$
LOVES$_4$	$:= \lambda xy.\, \mathsf{loves}\, x\, y$

Fig. 9.

abstract constants togheter with their lexicalized interpretations, they are given in Figure 8 and Figure 9.

5 From string to terms and vice versa

In the previous section, we have defined a lexicalized ACG that generates the λ-terms corresponding to the derivations of a given Lambek grammar. It now remains to interpret these λ-terms as strings. To this end, we first associate to each lambek type, α, a simple type $\overline{\alpha}$, built over a type σ corresponding to a set of strings Σ^*:

$$\begin{aligned} \overline{a} &= \sigma, \text{ for } a \text{ atomic} \\ \overline{\alpha \setminus \beta} &= \overline{\alpha} \rightarrow \overline{\beta} \\ \overline{\beta \,/\, \alpha} &= \overline{\alpha} \rightarrow \overline{\beta} \end{aligned}$$

We then define two families of combinators, $\mathbb{E}_\alpha : \sigma \rightarrow \overline{\alpha}$ and $\mathbb{P}_\alpha : \overline{\alpha} \rightarrow \sigma$, which allow to transform a strings into a λ-terms and vice versa:

$$\begin{aligned} \mathbb{E}_a \, w &= w \\ \mathbb{E}_{\alpha \setminus \beta} \, w &= \lambda x.\, \mathbb{E}_\beta \, ((\mathbb{P}_\alpha \, x) + w) \\ \mathbb{E}_{\beta \,/\, \alpha} \, w &= \lambda x.\, \mathbb{E}_\beta \, (w + (\mathbb{P}_\alpha \, x)) \end{aligned}$$

$$\begin{aligned} \mathbb{P}_a \, t &= t \\ \mathbb{P}_{\alpha \setminus \beta} \, t &= \mathbb{P}_\beta \, (t \, (\mathbb{E}_\alpha \, \epsilon)) \\ \mathbb{P}_{\beta \,/\, \alpha} \, t &= \mathbb{P}_\beta \, (t \, (\mathbb{E}_\alpha \, \epsilon)) \end{aligned}$$

We first state and prove two technical lemmas.

Lemma 1. *For every string w and every type α, $\mathbb{P}_\alpha \, (\mathbb{E}_\alpha \, t) = t$.*

Proof. By induction on α. □

Lemma 2. *Let t be a λ-term in normal form such that $(x_i : \alpha_i)_{i \in I} \vdash_{\lambda L} t : \alpha$. Then, for every set of strings $\{w_i \mid i \in I\}$, the following properties hold:*

1. *if t is a variable or an application, then $t[x_i := (\mathbb{E}_{\alpha_i} \, w_i)]_{i \in I} = \mathbb{E}_\alpha \, (\sum_{i \in I} w_i)$;*
2. *if t is an abstraction, then $\mathbb{P}_\alpha \, t[x_i := (\mathbb{E}_{\alpha_i} \, w_i)]_{i \in I} = \sum_{i \in I} w_i$.*

Proof. By induction on t. □

The following lemma is the main property of the operators $\mathbb{E}_\alpha$ and $\mathbb{P}_\alpha$.

Lemma 3. *Let t be a λ-term such that $(x_i : \alpha_i)_{i \in I} \vdash_{\lambda L} t : \alpha$. Then, for every set of strings $\{w_i \mid i \in I\}$, $\mathbb{P}_\alpha \, t[x_i := (\mathbb{E}_{\alpha_i} \, w_i)]_{i \in I} = \sum_{i \in I} w_i$.*

Proof. Let t' be the normal form of t. The results follows from Lemma 2, Lemma 1, and the fact that $\mathbb{P}_\alpha \, t[x_i := (\mathbb{E}_{\alpha_i} \, w_i)]_{i \in I} = \mathbb{P}_\alpha \, t'[x_i := (\mathbb{E}_{\alpha_i} \, w_i)]_{i \in I}$. □

We then immediatelly obtain the following corollary.

Corollary 1. *Let t be a λ-term such that $(x_i : \alpha_i)_{i \in I} \vdash_{\lambda L} t : a$, where a is an atomic type. Then, for every set of strings $\{w_i \mid i \in I\}$, $t[x_i := (\mathbb{E}_{\alpha_i} \, w_i)]_{i \in I} = \sum_{i \in I} w_i$.* □

Interpretation of the object constants

$$
\begin{aligned}
\mathsf{man} &:= \textbf{\textit{man}}\\
\mathsf{woman} &:= \textbf{\textit{woman}}\\
\mathsf{some} &:= \lambda x.\, \textbf{\textit{some}} + x\\
\mathsf{every} &:= \lambda x.\, \textbf{\textit{every}} + x\\
\mathsf{loves} &:= \lambda xy.\, y + \textbf{\textit{loves}} + x\\
\mathsf{who} &:= \lambda xy.\, y + \textbf{\textit{who}} + (x\,\epsilon)\\
\mathsf{whom} &:= \lambda xy.\, y + \textbf{\textit{whom}} + (x\,\epsilon)
\end{aligned}
$$

Fig. 10.

6 From LG to ACG

Consider the signature given in Figure 5. It has been used as the object vocabulary of both the ACGs $LDER$ and $LDER_{lex}$. We now use it as the abstract vocabulary of another ACG whose lexicon, which is given in Figure 10, is obtained by applying the operator $\mathbb{E}_{\alpha}$. Let us call this new ACG GEN_{string}.

The fact that the object vocabulary of $LDER_{lex}$ is, at the same time, the abstract vocabulary of GEN_{string} allows the two ACGs to be composed. Let us call the ACG resulting from this composition CFG_{λ}. The picture is then the following:

$$\begin{tikzcd}
\bullet \text{ Abstract Vocabulary (Fig. 4)} \arrow[d, "LDER_{lex}"] \arrow[dd, bend right, "CFG_{\lambda}"'] \\
\bullet \text{ Object vocabulary (Fig. 5)} \arrow[d, "GEN_{string}"] \\
\bullet \text{ Strings}
\end{tikzcd}$$

Now, in order to define an ACG that is strongly equivalent (up to a relabelling) to the original Lambek grammar, we need a last transformation. Consider the ACG CFG_{λ}, which has been obtained by omposition. It is a second-order ACG whose lexicon is third-order.[1] In fact, it is always possible to transform such an ACG in an object-language-equivalent third-order ACG whose lexicon is second order. Roughly speaking, the construction consists in replacing each atomic abstract type α, whose interpretation is $\alpha := a \rightarrow b$, by a type $\alpha_1 \rightarrow \alpha_2$ (where α_1 and α_2 are fresh symbols) together with the interpretations $\alpha_1 := a$ and $\alpha_2 := b$. Applying this transformation to CFG_{λ}, we end up with a grammar whose abstract vocabulary and lexicon are respectively given in Figure 11 and Figure 12. The object language of this ACG corresponds to the language generated by the original Lambek grammar. Its abstract language corresponds to the derivations of the original Lambek grammar (up to a relabelling, i.e., a homomorphism that is sending constants to constants).

[1] It is third-order because the type of the strings is defined to be the second-order type $o \rightarrow o$. Consequently, an apparently second-order interpretation such as $<n/np> := \sigma \rightarrow \sigma$ corresponds, in fact, to a third-order interpretation (namely, $<n/np> := (o \rightarrow o) \rightarrow o \rightarrow o$).

Abstract Vocabulary (final grammar)

MAN	:	n
WOMAN	:	n
SOME	:	$n \rightarrow np$
SOME$_0$	:	$(np_0 \rightarrow n_0) \rightarrow np_1 \rightarrow np_2$
EVERY	:	$n \rightarrow np$
EVERY$_0$	:	$(np_0 \rightarrow n_0) \rightarrow np_1 \rightarrow np_2$
LOVES	:	$np \rightarrow np \rightarrow s$
LOVES$_0$	:	$np \rightarrow np_3 \rightarrow s_0$
LOVES$_1$	:	$np \rightarrow np_4 \rightarrow s_1$
LOVES$_2$	:	$(np_1 \rightarrow np_2) \rightarrow np \rightarrow np_4 \rightarrow s_1$
LOVES$_3$	:	$(np_1 \rightarrow np_2) \rightarrow np_5 \rightarrow np_6 \rightarrow s_2$
LOVES$_4$	:	$np_5 \rightarrow np_6 \rightarrow s_2$
WHO	:	$(np_3 \rightarrow s_0) \rightarrow n \rightarrow n$
WHO$_0$	:	$(np_5 \rightarrow np_6 \rightarrow s_2) \rightarrow n \rightarrow np_0 \rightarrow n_0$
WHOM	:	$(np_4 \rightarrow s_1) \rightarrow n \rightarrow n$

Fig. 11.

Lexicon (final Grammar)

MAN	:=	***man***
WOMAN	:=	***woman***
SOME	:=	$\lambda x.$ ***some*** $+ x$
SOME$_0$	:=	$\lambda xy.$ ***some*** $+ (x\,y)$
EVERY	:=	$\lambda x.$ ***every*** $+ x$
EVERY$_0$	:=	$\lambda xy.$ ***every*** $+ (x\,y)$
LOVES	:=	$\lambda xy.\, y +$ ***loves*** $+ x$
LOVES$_0$	:=	$\lambda xy.\, y +$ ***loves*** $+ x$
LOVES$_1$	:=	$\lambda xy.\, x +$ ***loves*** $+ y$
LOVES$_2$	:=	$\lambda xyz.\, y +$ ***loves*** $+ (x\,z)$
LOVES$_3$	:=	$\lambda xyz.\, z +$ ***loves*** $+ (x\,y)$
LOVES$_4$	:=	$\lambda xy.\, y +$ ***loves*** $+ x$
WHO	:=	$\lambda xy.\, y +$ ***who*** $+ (x\,\epsilon)$
WHO$_0$	:=	$\lambda wxy.\, x +$ ***who*** $+ (w\,y\,\epsilon)$
WHOM	:=	$\lambda xy.\, y +$ ***whom*** $+ (x\,\epsilon)$

Fig. 12.

7 Conclusions and future work

We have demonstrated, on an example, how to represent a given Lambek grammar as an ACG. The generality of our method is somehow ensured by the several mathematical results on which it is based: weak equivalence between Lambek grammars and context-free grammars (Pentus [11, 12]), representing context-free grammars as ACGs (Pogodalla and de Groote [3]), lexicalization of second-order ACGs (Kanazawa and Yoshinaka [7, 8]), preservation of the derivations (Kanazawa and Salvati [6]). In practice, however, it will not always be the case that the resulting grammar will be strongly equivalent to the original Lambek grammar. This potential problem is due to Pentus' construction which produces highly redundant grammars exhibiting all the spurious ambiguities related to the associativity of the Lambek calculus.

In this paper, we circumvented the spurious ambiguity problem by implicitely using a particular instantiation of Pentus' construction. This more specific construction seems to be less general than Pentus'. Consequently, we do not know whether it can be put at work in every case. Investigating this question will be the subject of future work.

References

1. H.P. Barendregt. *The lambda calculus, its syntax and semantics*. North-Holland, revised edition, 1984.
2. Ph. de Groote. Towards abstract categorial grammars. In *Association for Computational Linguistics, 39th Annual Meeting and 10th Conference of the European Chapter, Proceedings of the Conference*, pages 148–155, 2001.
3. Ph. de Groote and S. Pogodalla. On the expressive power of abstract categorial grammars: Representing context-free formalisms. *Journal of Logic, Language and Information*, 13(4):421–438, 2004.
4. J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.
5. M. Kanazawa. Syntactic features for regular constraints and an approximation of directional slashes in abstract categorial grammars. In Y. Kubota and R. Levine, editors, *Proceedings for ESSLLI 2015 Workshop 'Empirical Advances in Categorial Grammars' (CG 2015)*, pages 34–70, 2015.
6. M. Kanazawa and S. Salvati. The string-meaning relations definable by lambek grammars and context-free grammars. In G. Morrill and M.-J. Nederhof, editors, *Formal Grammar - 17th and 18th International Conferences, FG 2012, Opole, Poland, August 2012, Revised Selected Papers, FG 2013, Düsseldorf, Germany, August 2013, Proceedings*, volume 8036 of *Lecture Notes in Computer Science*, pages 191–208. Springer, 2013.
7. M. Kanazawa and R. Yoshinaka. The complexity and generative capacity of lexicalized abstract categorial grammars. In Ph. Blache, E. Stabler, J. Busquets, and R. Moot, editors, *Logical Aspects of Computational Linguistics, LACL 2005*, volume 3492 of *Lecture Notes in Artificial Intelligence*, pages 330–346. Springer Verlag, 2005.
8. M. Kanazawa and R. Yoshinaka. Lexicalization of second-order acgs. Technical Report NII-2005-012E, National Institute of Informatics, Tokyo, 2005.
9. J. Lambek. The mathematics of sentence structure. *Amer. Math. Monthly*, 65:154–170, 1958.
10. J. Lambek. On the calculus of syntactic types. In *Studies of Language and its Mathematical Aspects*, pages 166–178, Providence, 1961. Proc. of the 12th Symp. Appl. Math..
11. M. Pentus. Lambek grammars are context free. In *Proceedings of the eigth annual IEEE symposium on logic in computer science*, pages 429–433, 1993.
12. M. Pentus. Product-free Lambek calculus and context-free grammars. *Journal of Symbolic Logic*, 62(2):648–660, 1997.
13. S. Pogodalla and F. Pompigne. Controlling extraction in abstract categorial grammars. In Ph. de Groote and M.-J. Nederhof, editors, *Formal Grammar - 15th and 16th International Conferences, FG 2010, Copenhagen, Denmark, August 2010, FG 2011, Ljubljana, Slovenia, August 2011, Revised Selected Papers*, volume 7395 of *Lecture Notes in Computer Science*, pages 162–177. Springer, 2012.
14. C. Retoré and S. Salvati. A faithful representation of non-associative lambek grammars in abstract categorial grammars. *Journal of Logic, Language and Information*, 19(2):185–200, 2010.