Abstract Categorial Parsing as Linear Logic Programming

Philippe de Groote
Inria Nancy - Grand Est
France
Philippe.deGroote@inria.fr

Abstract

This paper shows how the parsing problem for general Abstract Categorial Grammars can be reduced to the provability problem for Multiplicative Exponential Linear Logic. It follows essentially a similar reduction by Kanazawa, who has shown how the parsing problem for second-order Abstract Categorial Grammars reduces to datalog queries.

1 Introduction

Kanazawa (2007; 2011) has shown how parsing and generation may be reduced to datalog queries for a class of grammars that encompasses mildly context-sensitive formalisms. These grammars, which he calls *context-free λ-term grammars*, correspond to second-order abstract categorial grammars (de Groote, 2001).

In this paper, we show how Kanazawa's reduction may be carried out in the case of abstract categorial grammars of a degree higher than two. The price to pay is that we do not end up with a datalog query, but with a provability problem in multiplicative exponential linear logic (Girard, 1987). This is of course a serious difference. In particular, it is not known whether the multiplicative exponential fragment of linear logic is decidable.

The paper is organized as follows. Section 2 presents some mathematical preliminaries concerning the linear λ-calculus. We then introduce, in Section 3, the notion of abstract categorial grammar. Section 4 is the core of the paper, where we explain Kanazawa's reduction. To this end, we proceed by stepwise refinement. We first introduce an obviously correct but inefficient parsing algorithm. We then improve it by successive correctness-preserving transformations. Finally, we conclude in Section 5.

2 Linear λ-calculus

We assume from the reader some acquaintance with the basic concepts of the (simply typed) λ-calculus. Nevertheless, in order to fix the terminology and the notations, we briefly reminds the main definitions and properties that will be needed in the sequel. In particular, we review the notions *linear implicative types*, *higher-order linear signature*, and *linear λ-terms* built upon a higher-order linear signature.

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 $(\alpha \multimap \beta) \in \mathscr{T}(A)$.

Given two sets of atomic types, A and B, a mapping $h : \mathscr{T}(A) \to \mathscr{T}(B)$ is called a *type homomorphism* (or a *type substitution*) if it satisfies the following condition:

$$h(\alpha \multimap \beta) = h(\alpha) \multimap h(\beta)$$

A type substitution that maps atomic types to atomic types is called a *relabeling*.

In order to save parentheses, we use the usual convention of right association, i.e., we write $\alpha_1 \multimap \alpha_2 \multimap \cdots \alpha_n \multimap \alpha$ for $(\alpha_1 \multimap (\alpha_2 \multimap \cdots (\alpha_n \multimap \alpha) \cdots))$.

A *higher-order linear 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. $\tau : C \to \mathscr{T}(A)$ is a function that assigns to each constant in C a linear implicative type in $\mathscr{T}(A)$.

Given, a higher-order linear signature Σ, we write A_Σ, C_Σ, and τ_Σ, for its respective components.

The above notion of linear implicative type is isomorphic to the usual notion of simple type. Consequently, there is no technical difference between a higher-order linear signature and a higher-order signature. The only reason for using the word *linear* is to emphasize that we will be concerned with the typing of the *linear* λ*-terms*, i.e., the λ-terms whose typing system corresponds to the implicative fragment of multiplicative linear logic (Girard, 1987).

Let X be a infinite countable set of λ-variables. The set $\Lambda(\Sigma)$ of *linear* λ*-terms* built upon a higher-order linear signature Σ is inductively defined as follows:

1. if $c \in C_\Sigma$, 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 notions of capture-avoiding substitution, α-conversion, β-reduction, and η-reduction (Barendregt, 1984). Let t and u be linear λ-terms. We write $t \twoheadrightarrow_\beta u$ and $t =_\beta u$ for the relations of β-reduction and β-conversion, respectively, We use similar notations for the relations of reduction and conversion induced by η and $\beta\eta$.

Let Σ_1 and Σ_2 be two signatures. We say that a mapping $h : \Lambda(\Sigma_1) \to \Lambda(\Sigma_2)$ is a λ-term homomorphism if it satisfies the following conditions:

$$h(x) = x$$
$$h(\lambda x.\, t) = \lambda x.\, h(t)$$
$$h(t\, u) = h(t)\, (h(u))$$

Given a higher-order linear signature Σ, each linear λ-term in $\Lambda(\Sigma)$ may possibly be assigned a linear implicative type in $\mathscr{T}(A_\Sigma)$. This type assignment obeys the following typing rules:

$$\vdash_\Sigma\ c : \tau_\Sigma(c) \quad \text{(CONS)}$$

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

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

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

where $\mathrm{dom}(\Gamma) \cap \mathrm{dom}(\Delta) = \varnothing$.

We end this section by reviewing some properties that will turn out to be useful in the sequel.

The set of linear λ-terms being a subset of the set of simply typed λ-terms, it inherits the universal properties of the latter (e.g., strong normalization, or existence of a principal type scheme). It also satisfies the usual subject-reduction property.

Proposition 1 *Let Σ, t, u, Γ, and α be such that $\Gamma \vdash_\Sigma t : \alpha$ and $t \twoheadrightarrow_\beta u$. Then $\Gamma \vdash_\Sigma u : \alpha$.* □

The set of simply typed λ-terms, which is not closed in general under β-expansion, is known to be closed under *linear* β-expansion. Consequently, the set of linear λ-terms satisfies the subject-expansion property.

Proposition 2 *Let Σ, t, u, Γ, and α be such that $\Gamma \vdash_\Sigma u : \alpha$ and $t \twoheadrightarrow_\beta u$. Then $\Gamma \vdash_\Sigma t : \alpha$.* □

The subject-reduction property also holds for the relation of $\beta\eta$-reduction. This is not the case, however, for the subject-expansion property. This possible difficulty may be circumvented by using the notion of η-long form.

A linear λ-term is said to be in η-long form when every of its sub-terms of functional type is either a λ-abstraction or the operator of an application. The set of linear λ-terms in η-long forms is closed under both β-reduction and β-expansion. Consequently, the following proposition holds.

Proposition 3 *Let t and u be λ-terms in η-long forms. Then, $t =_{\beta\eta} u$ if and only if $t =_\beta u$.* □

In the sequel, we will often assume that the linear λ-terms under consideration are in η-long forms. This

will allow us to only consider β-reduction and β-expansion, while using the relation of $\beta\eta$-conversion as the notion of equality between linear λ-terms.

Finally, it is known from a categorical coherence theorem that every *balanced* simple type is inhabited by at most one λ-term up to $\beta\eta$-conversion (see (Babaev and Solov'ev, 1982; Mints, 1981)). It is also known that the principal type of a pure linear λ-term is balanced (Hirokawa, 1991). Consequently, the following property holds.

Proposition 4 *Let t be a pure linear λ-term (i.e., a linear λ-term that does not contain any constant), and let $\Gamma \vdash t : \alpha$ be its principal typing. If u is a pure linear λ-term such that $\Gamma \vdash u : \alpha$, then $t =_{\beta\eta} u$.* □

3 Abstract Categorial Grammar

This section gives the definition of an abstract categorial grammar (ACG, for short) (de Groote, 2001).

We first define a *lexicon* to be a morphism between higher-order linear signatures. Let $\Sigma_1 = \langle A_1, C_1, \tau_1\rangle$ and $\Sigma_2 = \langle A_2, C_2, \tau_2\rangle$ be two higher-order signatures. A lexicon $\mathscr{L} : \Sigma_1 \to \Sigma_2$ is a realization of Σ_1 into Σ_2, i.e., 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 *lexicon* $\mathscr{L}$ from Σ_1 to Σ_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 Σ_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 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:

$$\vdash_{\Sigma_2} G(c) : \hat{F}(\tau_1(c)) \qquad (1)$$

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

Remark that Condition (1) compels $G(c)$ to be typable with respect to the empty typing environment. This means that G interprets each constant c as a closed linear λ-term. Now, defining $\hat{G}$ to be the unique homomorphic extension of G, Condition (1) ensures that the following commutation property holds for every $t \in \Lambda(\Sigma_1)$:

$$\text{if} \quad \vdash_{\Sigma_1} t : \alpha \quad \text{then} \quad \vdash_{\Sigma_2} \hat{G}(t) : \hat{F}(\alpha)$$

In the sequel, given such a lexicon $\mathscr{L} = \langle F, G\rangle$, $\mathscr{L}(a)$ will stand for either $\hat{F}(a)$ or $\hat{G}(a)$, according to the context.

We now define an *abstract categorial grammar* as quadruple, $\mathscr{G} = \langle \Sigma_1, \Sigma_2, \mathscr{L}, S\rangle$, where:

1. Σ_1 and Σ_2 are two higher-order linear signatures; they are called the *abstract vocabulary* and the *object vocabulary*, respectively;

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

3. S is an atomic type of the abstract vocabulary; it is called the *distinguished type* of the grammar.

Every ACG $\mathscr{G}$ generates two languages: an *abstract language*, $\mathcal{A}(\mathscr{G})$, and an object language $\mathcal{O}(\mathscr{G})$.

The abstract language, which may be seen as a set of abstract parse structures, is the set of closed linear λ-terms built upon the abstract vocabulary and whose type is the distinguished type of the grammar. It is formally defined as follows:

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

The *object language*, which may be seen as the set of surface forms generated by the grammar, is defined to be the image of the abstract language by the term homomorphism induced by the lexicon.

$$\mathcal{O}(\mathscr{G}) = \{t \in \Lambda(\Sigma_2) \,:\, \exists u \in \mathcal{A}(\mathscr{G}).\ t =_{\beta\eta} \mathscr{L}(u)\}$$

Both the abstract language and the object language generated by an ACG are sets of linear λ-terms. This allows more specific data structures such as strings, trees, or first-order terms to be represented. A string of symbols, for instance, can be encoded as a composition of functions. Consider an

$$
\begin{aligned}
\textsc{man} &: N \\
\textsc{woman} &: N \\
\textsc{wise} &: N \multimap N \\
\textsc{a}_s &: N \multimap (NP_s \multimap \overline{S}) \multimap S \\
\textsc{a}_o &: N \multimap (NP_o \multimap \overline{S}) \multimap \overline{S} \\
\textsc{seek} &: ((NP_o \multimap S) \multimap \overline{S}) \multimap NP_s \multimap S \\
\textsc{inj} &: S \multimap \overline{S}
\end{aligned}
$$

Figure 1: The abstract vocabulary Σ_1

$$
\begin{aligned}
\textsc{man} &:= \textbf{\textit{man}} \ : \ \sigma \\
\textsc{woman} &:= \textbf{\textit{woman}} \ : \ \sigma \\
\textsc{wise} &:= \lambda x.\, \textbf{\textit{wise}} + x \ : \ \sigma \multimap \sigma \\
\textsc{a}_s &:= \lambda x p.\, p\,(\textbf{\textit{a}} + x) \ : \ \sigma \multimap (\sigma \multimap \sigma) \multimap \sigma \\
\textsc{a}_o &:= \lambda x p.\, p\,(\textbf{\textit{a}} + x) \ : \ \sigma \multimap (\sigma \multimap \sigma) \multimap \sigma \\
\textsc{seek} &:= \lambda p x.\, p\,(\lambda y.\, x + \textbf{\textit{seeks}} + y) \ : \ ((\sigma \multimap \sigma) \multimap \sigma) \multimap \sigma \multimap \sigma \\
\textsc{inj} &:= \lambda x.\, x \ : \ \sigma \multimap \sigma
\end{aligned}
$$

Figure 2: The lexicon $\mathscr{L} : \Sigma_1 \to \Sigma_2$

arbitrary atomic type s, and define $\sigma \triangleq s \multimap s$ to be the type of strings. Then, a string such as '$abbac$' may be represented by the linear λ-term:

$$\lambda x.\, a\,(b\,(b\,(a\,(c\,x)))),$$

where the atomic strings 'a', 'b', and 'c' are declared to be constants of type σ. In this setting, the empty word is represented by the identity function:

$$\epsilon \triangleq \lambda x.\, x$$

and concatenation is defined to be functional composition:

$$\_ + \_ \triangleq \lambda\alpha.\, \lambda\beta.\, \lambda x.\, \alpha\,(\beta\, x),$$

which is indeed an associative operator that admits the identity function as a unit.

We end this section by giving a fragment of a categorial grammar that will serve as a running example throughout the rest of this paper.[1]

The abstract vocabulary, which specifies the abstract parse structures, is given in Fig. 1. In this signature, the atomic types (N, NP_s, NP_o, $\overline{S}$, S) must be thought of as atomic syntactic categories. The lexicon, which is given in Fig. 2, allows the abstract structures to be transformed in surface forms. These surface forms are strings that are built upon an object vocabulary, Σ_2, which includes the following atomic strings as constants of type σ: ***man***, ***woman***, ***wise***, ***a***, ***seeks***.

For such a grammar, the parsing problem consists in deciding whether a possible surface form (i.e.,

[1] This grammar, which follows the categorial type-logical tradition (Moortgat, 1997), has been devised in order to present the main difficulties encountered in ACG parsing: it is higher order (it assigns third-order types to the quantified noun phrases, and a fourth-order type to an intensional transitive verb such as *seek*); it is lexically ambiguous (it assigns two different lexical entries to the indefinite determiner); and it includes a non-lexicalized entry (the coercion operator INJ).

term $t \in \Lambda(\Sigma_2)$) belongs to the object vocabulary of the grammar. Spelling it out, is there an abstract parse structure (i.e., a term $u \in \Lambda(\Sigma_1)$ of type S) whose image through the lexicon is the given surface form (i.e., $\mathscr{L}(u) = t$).

Consider, for instance, the following string:

$$\textbf{\textit{a}} + \textbf{\textit{wise}} + \textbf{\textit{woman}} + \textbf{\textit{seeks}} + \textbf{\textit{a}} + \textbf{\textit{wise}} + \textbf{\textit{man}} \quad (2)$$

One can show that it belongs to the object language of the grammar. Indeed, when applying the lexicon to the following abstract term:

$$\begin{array}{l} \mathrm{A}_s\,(\textsc{wise woman}) \\ \quad (\lambda x.\ \textsc{inj}\,(\textsc{seek}\,(\lambda p.\ \mathrm{A}_o\,(\textsc{wise man}) \\ \qquad\qquad\qquad\qquad (\lambda y.\ \textsc{inj}\,(p\,y))) \\ \qquad\qquad x)) \end{array} \quad (3)$$

one obtains a λ-term that is $\beta\eta$-convertible to (2). In fact, it is even the case that (2) is ambiguous in the sense that there is another abstract term, essentially different from (3), whose image through the lexicon yields (2).[2] This abstract term is the following:

$$\begin{array}{l} \mathrm{A}_s\,(\textsc{wise woman}) \\ \quad (\lambda x.\ \mathrm{A}_o\,(\textsc{wise man}) \\ \qquad (\lambda y.\ \textsc{inj}\,(\textsc{seek}\,(\lambda p.\ \textsc{inj}\,(p\,y)) \\ \qquad\qquad\qquad x))) \end{array} \quad (4)$$

4 Development of the parsing algorithm

In this section, we develop a parsing algorithm based on proof-search in the implicative fragment of linear logic. We start with a simple non-deterministic algorithm, which is rather inefficient but whose correctness and semi-completeness are obvious. Then, we proceed by stepwise refinement, preserving the correctness and semi-completeness of the algorithm.

By correctness, we mean that if the parsing algorithm answers positively then it is indeed the case that the input term belongs to the object language of the grammar. By semi-completeness, we mean that if the input term belongs to the object language of the grammar, then the parsing algorithm will eventually give a positive answer.

In the present state of knowledge, semi-completeness is the best we may expect. Indeed, the ACG membership problem is known to be equivalent to provability in multiplicative exponential logic (de Groote et al., 2004; Yoshinaka and Kanazawa, 2005), the decidability of which is still open.

[2] If the grammar was provided with a Montague semantics, the abstract parse structures (3) and (4) would correspond to the *de dicto* and *de re* readings, respectively.

4.1 Generate and test

Our starting point is a simple *generate and test* algorithm:

1. *derive S using the rules of implicative linear logic with the types of the abstract constants* (Fig. 3) *as proper axioms;*
2. *interpret the obtained derivation as a linear λ-term (through the Curry-Howard isomorphism);*
3. *apply the lexicon to the resulting λ-term, and check whether it yields a term $\beta\eta$-convertible to the input term.*

$$\begin{array}{ll} N & (\textsc{man}) \\ N & (\textsc{woman}) \\ N \multimap N & \\ N \multimap (NP_s \multimap \overline{S}) \multimap S & \\ N \multimap (NP_o \multimap \overline{S}) \multimap \overline{S} & \\ ((NP_o \multimap S) \multimap \overline{S}) \multimap NP_s \multimap S & \\ S \multimap \overline{S} & \end{array}$$

Figure 3: The type of the abstract constants as proper axioms

The above algorithm is obviously correct. It is also semi-complete because it enumerates all the terms of the abstract language. Now, if the input term belongs to the object language of the grammar then its abstract parse structure(s) will eventually appear in the enumeration.

4.2 Type-driven search

The generate and test algorithm proceeds by trial and error without taking into account the form of the input term. In order to improve our algorithm, we must focus on the construction of an abstract term

whose image by the lexicon would be the input term. To this end, we take advantage of Proposition 4.

In general, the input term is not a pure λ-term. Consequently, in order to apply Proposition 4, we must consider each occurrence of a constant in the input term as a fresh free variable. Applying this idea to our example, we obtain the following principal typing that characterizes uniquely the input string (in η-long β-normal form):

$$\begin{array}{l}\boldsymbol{a}_1 : s_1 \multimap s_0,\ \boldsymbol{wise}_1 : s_2 \multimap s_1,\\ \boldsymbol{woman} : s_3 \multimap s_2, \boldsymbol{seeks} : s_4 \multimap s_3,\\ \boldsymbol{a}_2 : s_5 \multimap s_4,\ \boldsymbol{wise}_2 : s_6 \multimap s_5,\ \boldsymbol{man} : s_7 \multimap s_6\\ \vdash\ \lambda z.\, \boldsymbol{a}_1\,(\boldsymbol{wise}_1\,(\boldsymbol{woman}\\ \qquad (\boldsymbol{seeks}\,(\boldsymbol{a}_2\,(\boldsymbol{wise}_2\,(\boldsymbol{man}\,z)))))))) : s_7 \multimap s_0\end{array}$$

The types assigned to the constant occurrences of the input term induce a new specialized object vocabulary, which we will call Σ_2^S. We take for granted the definition of the forgetful homomorphism

$$|\cdot| : \Sigma_2^S \to \Sigma_2$$

that allows to project Σ_2^S on Σ_2. Roughly speaking, this forgetful homomorphism consists simply in identifying the several occurences of a same object constant. Remark that at the level of the types, this forgetful homomorphism is a relabeling because the input string has been given in η-long form. In our case, this relabeling is the following one:

$$|s_i| = s \quad (0 \leq i \leq 7)$$

The next step is to adapt the abstract vocabulary and the lexicon to this specialized object vocabulary. We start with the abstract atomic types. Let $a \in A_{\Sigma_1}$, and define the set $\xi(a)$ as follows:

$$\xi(a) = \{\alpha \in \mathscr{T}(A_{\Sigma_2^S}) \,:\, |\alpha| = \mathcal{L}(\tau_{\Sigma_1}(a))\}$$

For instance, we have:

$$\xi(N) = \{s_i \multimap s_j \,:\, 0 \leq i \leq 7 \,\&\, 0 \leq j \leq 7\}$$

Then we define the set of atomic types of the specialized abstract signature as follows:

$$A_{\Sigma_1^S} = \{a_\alpha \,:\, a \in A_{\Sigma_1} \,\&\, \alpha \in \xi(a)\}$$

and we let

$$\mathcal{L}^S(a_\alpha) = \alpha$$

Back to our example, it means that the specialised abstract signature contains 64 copies of N:

$$\begin{array}{cccc} N_{s_0 \multimap s_0}, & N_{s_0 \multimap s_1}, & \ldots & N_{s_0 \multimap s_7},\\ \vdots & \vdots & \ddots & \vdots\\ N_{s_7 \multimap s_0}, & N_{s_7 \multimap s_1}, & \ldots & N_{s_7 \multimap s_7}.\end{array}$$

In order to accommodate the abstract constants, we look at the lexicon. Consider the first two lexical entries. Their typing, according to the specialized object vocabulary, is as follows:

$$\begin{array}{rcl}\textsc{man} & := & \lambda z.\, \boldsymbol{man}\, z \ :\ s_7 \multimap s_6\\ \textsc{woman} & := & \lambda z.\, \boldsymbol{woman}\, z \ :\ s_3 \multimap s_2\end{array}$$

Accordingly, we let the specialized abstract vocabulary contain the following two constants:

$$\begin{array}{rcl}\textsc{man} & : & N_{s_7 \multimap s_6}\\ \textsc{woman} & : & N_{s_3 \multimap s_2}\end{array}$$

Consider now the third entry:

$$\textsc{wise} := \lambda xz.\, \boldsymbol{wise}\,(x\,z) \ :\ (s \multimap s) \multimap s \multimap s$$

There are two ways of specializing it. On the one hand, the object constant ***wise*** may be replaced by its first occurrence (***wise***$_1$) or by its second one (***wise***$_2$). On the other hand, each occurrence of the atomic type s may be instantiated by one of s_0, s_1, ..., s_7. This give rise to 8,192 a priori possibilities. These possibilities, however, do not all correspond to actual typing judgements. Filtering out the ill-typed ones (which is effective since typing is decidable), we are left with 16 new lexical entries which obey the following schemes:

$$\begin{array}{rcl}\textsc{wise}_{1i} & := & \lambda xz.\, \boldsymbol{wise}_1\,(x\,z) \ :\\ & & \quad (s_i \multimap s_2) \multimap s_i \multimap s_1 \quad (0 \leq i \leq 7)\\ \textsc{wise}_{2i} & := & \lambda xz.\, \boldsymbol{wise}_2\,(x\,z) \ :\\ & & \quad (s_i \multimap s_6) \multimap s_i \multimap s_5 \quad (0 \leq i \leq 7)\end{array}$$

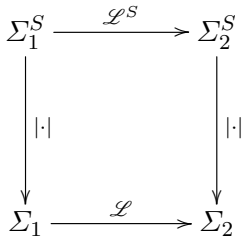
and we add the following 16 constants to the specialized abstract vocabulary:

$$\begin{array}{rcl}\textsc{wise}_{1i} & : & N_{s_i \multimap s_2} \multimap N_{s_i \multimap s_1} \quad (0 \leq i \leq 7)\\ \textsc{wise}_{2i} & : & N_{s_i \multimap s_6} \multimap N_{s_i \multimap s_5} \quad (0 \leq i \leq 7)\end{array}$$

By proceeding in the same way with the other lexical entries, we obtain a new specialized abstract signature Σ_1^S together with a new specialized lexicon:

$$\mathscr{L}^S : \Sigma_1^S \to \Sigma_2^S$$

Clearly, there exists a forgetful homomorphism between Σ_1^S and Σ_1, and the specialized abstract signature and specialized lexicon are such that the following diagram commutes:



We may now use the specialized grammar to drive the proof-search on which the generate and test algorithm is based. Remember that the specialized object type assigned to the input string is $s_7 \multimap s_0$. Our parsing problem is then reduced to the following proof-search problem:

> *derive $S_{s_7 \multimap s_0}$ using the rules of implicative linear logic with the types of the specialized abstract constants as proper axioms.*

Now, suppose that we derive $S_{s_7 \multimap s_0}$, and that $t \in \Lambda(\Sigma_1^S)$ is the specialized abstract linear λ-term corresponding to this derivation. By construction of the specialized grammar, we have that:

$$\vdash_{\Sigma_2^S} \mathscr{L}^s(t) : s_7 \multimap s_0 \tag{5}$$

Then, by Proposition 4, we have that

$$\mathscr{L}^s(t) =_{\beta\eta} \lambda z.\, \boldsymbol{a}_1\,(\boldsymbol{wise}_1\,(\boldsymbol{woman}\,(\boldsymbol{seeks}\,(\boldsymbol{a}_2\,(\boldsymbol{wise}_2\,(\boldsymbol{man}\,z))))))) \tag{6}$$

because

$$\vdash_{\Sigma_2^S} \lambda z.\, \boldsymbol{a}_1\,(\boldsymbol{wise}_1\,(\boldsymbol{woman}\,(\boldsymbol{seeks}\,(\boldsymbol{a}_2\,(\boldsymbol{wise}_2\,(\boldsymbol{man}\,z)))))) : s_7 \multimap s_0 \tag{7}$$

amounts to a principal typing. Finally, by taking $t' = |t|$, we obtain a term $t' \in \Lambda(\Sigma_1)$ such that:

$$\mathscr{L}(t') =_{\beta\eta} \lambda z.\, \boldsymbol{a}\,(\boldsymbol{wise}\,(\boldsymbol{woman}\,(\boldsymbol{seeks}\,(\boldsymbol{a}\,(\boldsymbol{wise}\,(\boldsymbol{man}\,z)))))) \tag{8}$$

This shows the correctness of the algorithm.

To establish its semi-completeness, suppose that there exists an abstract linear λ-term $t' \in \Lambda(\Sigma_1)$ such that (8). From this, one can easily construct a term $t \in \Lambda(\Sigma_1^S)$ of type S such that $|t| = t'$ and Equation (6) holds. Since the lexical entries are given in η-long forms, so is $\mathscr{L}^s(t)$. Then, because the specialized input term is in η-long β-normal form, by Proposition 3, we have that:

$$\mathscr{L}^s(t) \twoheadrightarrow_\beta \lambda z.\, \boldsymbol{a}_1\,(\boldsymbol{wise}_1\,(\boldsymbol{woman}\,(\boldsymbol{seeks}\,(\boldsymbol{a}_2\,(\boldsymbol{wise}_2\,(\boldsymbol{man}\,z)))))) \tag{9}$$

Then, (5) follows from (7) and (9) by Proposition 2. From this, it is not too difficult to establish that t is of type $S_{s_7 \multimap s_0}$.

4.3 Proof-search in the implicative fragment of linear logic

The type-driven algorithm that we have sketched presents two serious defects. On the one hand, the construction of the specialized grammar is both time and space consuming. For our simple running example, for instance, we would obtain 6,226 specialized lexical entries. On the other hand, the reduction depends upon the input string.

In order to circumvent these difficulties, consider again the specialized lexical entries corresponding to the third lexical entry of the original grammar:

$$\begin{aligned}
\textsc{wise}_{10} &:= \lambda xz.\, \boldsymbol{wise}_1\,(x\,z) : (s_0 \multimap s_2) \multimap s_0 \multimap s_1\\
\textsc{wise}_{11} &:= \lambda xz.\, \boldsymbol{wise}_1\,(x\,z) : (s_1 \multimap s_2) \multimap s_1 \multimap s_1\\
&\vdots\\
\textsc{wise}_{17} &:= \lambda xz.\, \boldsymbol{wise}_1\,(x\,z) : (s_7 \multimap s_2) \multimap s_7 \multimap s_1\\
\textsc{wise}_{20} &:= \lambda xz.\, \boldsymbol{wise}_2\,(x\,z) : (s_0 \multimap s_6) \multimap s_0 \multimap s_5\\
\textsc{wise}_{21} &:= \lambda xz.\, \boldsymbol{wise}_2\,(x\,z) : (s_1 \multimap s_6) \multimap s_1 \multimap s_5\\
&\vdots\\
\textsc{wise}_{27} &:= \lambda xz.\, \boldsymbol{wise}_2\,(x\,z) : (s_7 \multimap s_6) \multimap s_7 \multimap s_5
\end{aligned}$$

In fact, all the specialized object types assigned to these lexical entries are instances of the principal typing of the corresponding lexical entry of the original lexicon:

$$\boldsymbol{wise} : j \multimap i \vdash \lambda xz.\, \boldsymbol{wise}\,(x\,z) : (k \multimap j) \multimap k \multimap i$$

This means that if the specialized object vocabulary assigns the constant ***wise*** with the following type:

$$\boldsymbol{wise} : j \multimap i \tag{10}$$

then the specialized abstract vocabulary should contain abstract constants obeying the following type scheme:

$$N_{k \multimap j} \multimap N_{k \multimap i} \tag{11}$$

$$\textbf{\textit{man}}[i,j] \vdash N[i,j]$$
$$\textbf{\textit{woman}}[i,j] \vdash N[i,j]$$
$$\textbf{\textit{wise}}[i,j] \vdash N[j,k] \multimap N[i,k]$$
$$\textbf{\textit{a}}[i,j] \vdash N[j,k] \multimap (NP_s[i,k] \multimap \overline{S}[l,m]) \multimap S[l,m]$$
$$\textbf{\textit{a}}[i,j] \vdash N[j,k] \multimap (NP_o[i,k] \multimap \overline{S}[l,m]) \multimap \overline{S}[l,m]$$
$$\textbf{\textit{seeks}}[i,j] \vdash ((NP_o[j,k] \multimap S[l,k]) \multimap \overline{S}[m,n]) \multimap NP_s[l,i] \multimap S[m,n]$$
$$\vdash S[i,j] \multimap \overline{S}[i,j]$$

Figure 4: The lexicon as a linear logic program

$$\frac{\Gamma \vdash \textbf{\textit{man}}[i,j]}{\Gamma \vdash N[i,j]}\ (\mathrm{M}) \qquad \frac{\Gamma \vdash \textbf{\textit{woman}}[i,j]}{\Gamma \vdash N[i,j]}\ (\mathrm{W}) \qquad \frac{\Gamma \vdash \textbf{\textit{wise}}[i,j] \quad \Delta \vdash N[j,k]}{\Gamma, \Delta \vdash N[i,k]}\ (\mathrm{WI})$$

$$\frac{\Gamma \vdash \textbf{\textit{a}}[i,j] \qquad \Delta \vdash N[j,k] \qquad \Theta, NP_s[i,k] \vdash \overline{S}[l,m]}{\Gamma, \Delta, \Theta \vdash S[l,m]}\ (\mathrm{A}_s)$$

$$\frac{\Gamma \vdash \textbf{\textit{a}}[i,j] \qquad \Delta \vdash N[j,k] \qquad \Theta, NP_o[i,k] \vdash \overline{S}[l,m]}{\Gamma, \Delta, \Theta \vdash \overline{S}[l,m]}\ (\mathrm{A}_o)$$

$$\frac{\Gamma \vdash \textbf{\textit{seeks}}[i,j] \qquad \Delta, NP_o[j,k] \multimap S[l,k] \vdash \overline{S}[m,n] \qquad \Theta \vdash NP_s[l,i]}{\Gamma, \Delta, \Theta \vdash S[m,n]}\ (\mathrm{S})$$

$$\frac{\Gamma \vdash S[i,j]}{\Gamma \vdash \overline{S}[i,j]}\ (\mathrm{I})$$

Figure 5: The lexicon as a set of inference rules

Writing $N[j,k]$ for $N_{k-\circ j}$ and representing (10) by the predicate $\textbf{\textit{wise}}[i,j]$, we may represent the dependence between 10 and 11 by the following linear logic sequent:[3]

$$\textbf{\textit{wise}}[i,j] \vdash N[j,k] \multimap N[i,k]$$

Applying the same process to the other lexical entries, we end up with the set of sequents given in Fig. 4. Our parsing problem amounts then to a proof-search problem in linear logic:

> *derive*
>
> $\textbf{\textit{a}}[0,1], \textbf{\textit{wise}}[1,2], \textbf{\textit{woman}}[2,3], \textbf{\textit{seeks}}[3,4],$
> $\textbf{\textit{a}}[4,5], \textbf{\textit{wise}}[5,6], \textbf{\textit{man}}[6,7] \vdash S[0,7]$
>
> *using the rules of implicative linear logic with the set of sequents of Fig. 4 as proper axioms.*

We give in Fig. 6 and Fig. 7 (in the annex) the derivations corresponding to the *de dicto* parsing (3) and to the *de re* parsing (4). These two derivations use the inference rules given in Fig. 5, which are equivalent to the sequents of Fig. 4.

Acknowledgments

I am grateful to Makoto Kanazawa for fruitful discussions about his work on parsing as datalog

[3] Following (Kanazawa, 2011) and (Kanazawa, 2007), when writing $N[j,k]$ for $N_{k-\circ j}$, we write the variables in the reverse order.

queries. A preliminary version of the results reported in this paper has been presented in a talk given in June 2007 at the Colloquium in Honor of Gérard Huet on the occasion of his 60th birthday. This work has been supported by the French agency *Agence Nationale de la Recherche* (ANR-12-CORD-0004).

References

A.A. Babaev and S.V. Solov'ev. 1982. A coherence theorem for canonical morphisms in cartesian closed categories. *Journal of Soviet Mathematics*, 20(4):2263–2279. *Original in Russian: Zapiski Nauchnykh Seminarov Leningradskogo Otdeleniya Matematicheskogo Instituta imeni V. A. Steklova Akademii Nauk SSSR (LOMI), 88:3–29, 1979.*

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

Ph. de Groote, B. Guillaume, and S. Salvati. 2004. Vector addition tree automata. In *Proceedings of the 19th annual IEEE symposium on logic in computer science*, pages 64–73.

Ph. de Groote. 2001. 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.

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

J.R. Hindley. 1969. The principal type-scheme of an object in combinatory logic. *Transaction of the American Mathematical Society*, 146:29–60.

S. Hirokawa. 1991. Principal type-schemes of bci-lambda-terms. In T. Ito and A.R. Meyer, editors, *Theoretical Aspects of Computer Software, TACS'91*, volume 526 of *Lecture Notes in Computer Science*, pages 633–650. Springer-Verlag.

M. Kanazawa. 2007. Parsing and generation as datalog queries. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics*, pages 176–183. Association for Computational Linguistics.

M. Kanazawa. 2011. Parsing and generation as datalog query evaluation. Last revised August 26, 2011. 74 pages. (Under review).

G.E. Mints. 1981. Closed categories and the theory of proofs. *Journal of Soviet Mathematics*, 15(1):45–62. *Original in Russian: Zapiski Nauchnykh Seminarov Leningradskogo Otdeleniya Matematicheskogo Instituta imeni V. A. Steklova Akademii Nauk SSSR (LOMI), 68:83–114, 1977.*

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

R. Yoshinaka and M. Kanazawa. 2005. The complexity and generative capacity of lexicalized abstract categorial grammars. In Philippe Blache, E. Stabler, J. Busquets, and R. Moot, editors, *Logical Aspects of Computational Linguistics, LACL 2005*, volume 3492 of *Lecture Notes in Computer Science*, pages 330–346. Springer-Verlag.

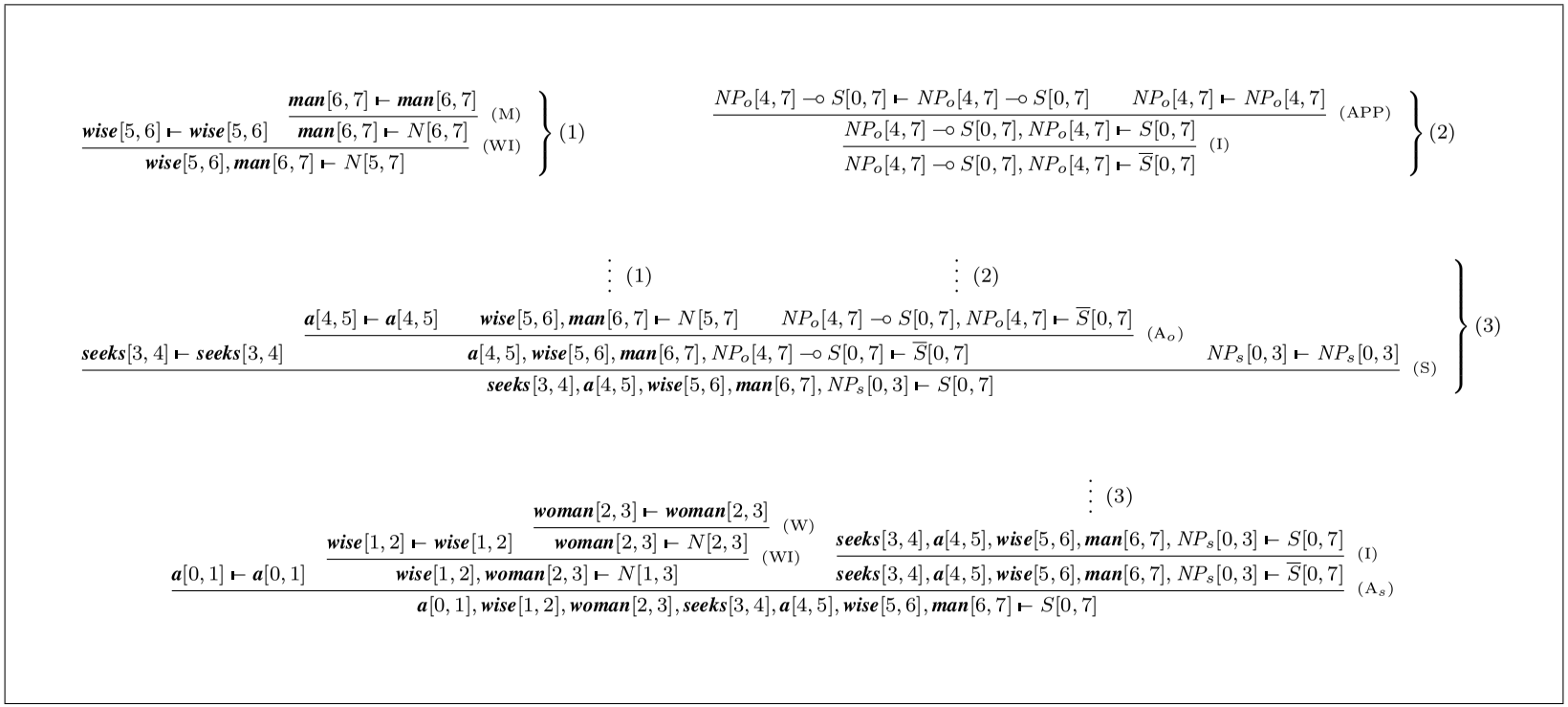


Figure 6: The derivation corresponding to $\textsc{a}_s\ (\textsc{wise woman})\ (\lambda x.\ \textsc{inj}\ (\textsc{seek}\ (\lambda p.\ \textsc{a}_o\ (\textsc{wise man})\ (\lambda y.\ \textsc{inj}\ (p\ y)))\ x))$

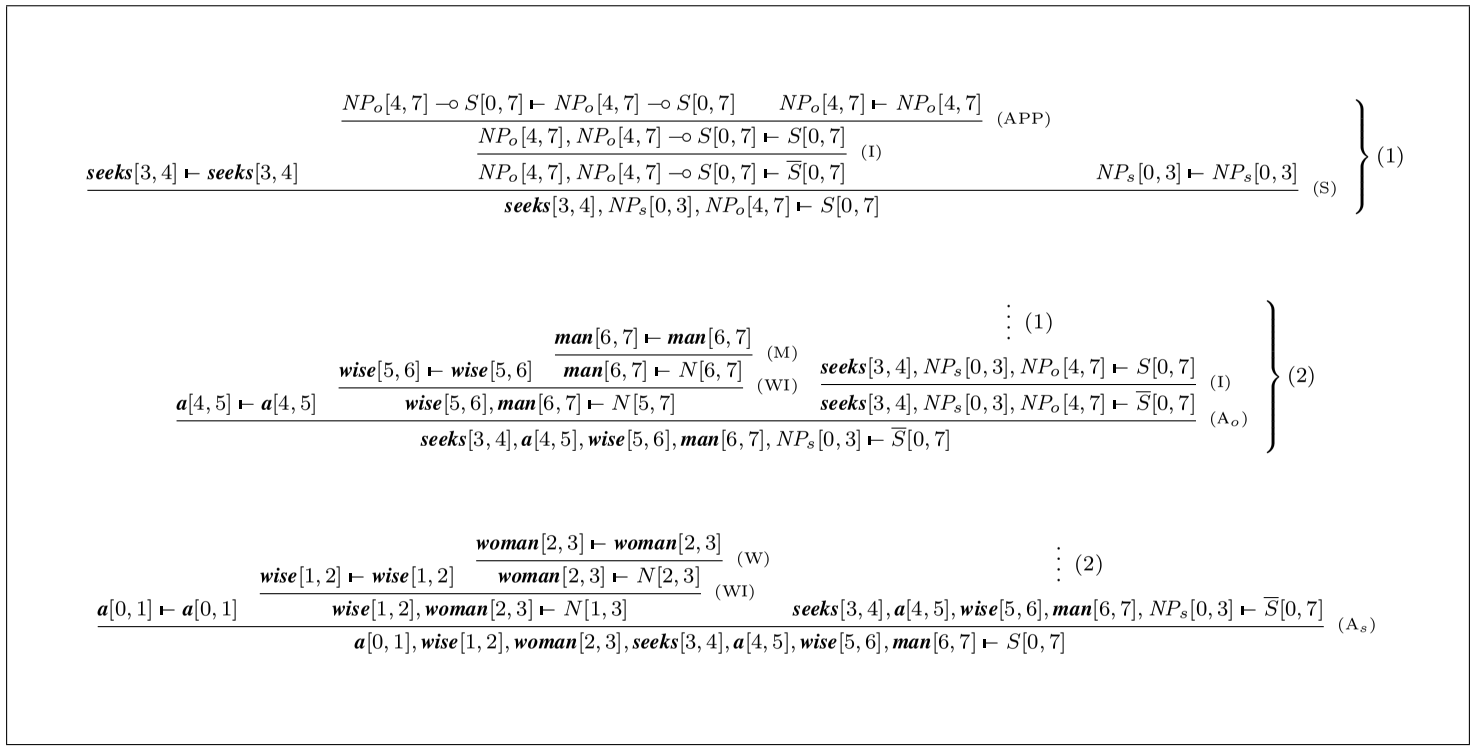


Figure 7: The derivation corresponding to $\text{A}_s\ (\text{WISE WOMAN})\ (\lambda x.\ \text{A}_o\ (\text{WISE MAN})\ (\lambda y.\ \text{INJ}\ (\text{SEEK}\ (\lambda p.\ \text{INJ}\ (p\ y))\ x)))$