On the Syntax-Semantics Interface: From Convergent Grammar to Abstract Categorial Grammar

Philippe de Groote, Sylvain Pogodalla, Carl Pollard

On the Syntax-Semantics Interface: From Convergent Grammar to Abstract Categorial Grammar *

Philippe de Groote[1], Sylvain Pogodalla[2], and Carl Pollard[3]

[1] philippe.degroote@loria.fr, LORIA/INRIA Nancy – Grand Est
[2] sylvain.pogodalla@loria.fr, LORIA/INRIA Nancy – Grand Est
[3] pollard@ling.ohio-state.edu, The Ohio State University

Abstract. Cooper's storage technique for scoping *in situ* operators has been employed in theoretical and computational grammars of natural language (NL) for over thirty years, but has been widely viewed as ad hoc and unprincipled. Recent work by Pollard within the framework of convergent grammar (CVG) took a step in the direction of clarifying the logical status of Cooper storage by encoding its rules within an explicit but nonstandard natural deduction (ND) format. Here we provide further clarification by showing how to encode a CVG with storage within a logical grammar framework—abstract categorial grammar (ACG)—that utilizes no logical resources beyond those of standard linear deduction.

Introduction

A long-standing challenge for designers of NL grammar frameworks is posed by **in situ operators**, expressions such as quantified noun phrases (QNPs, e.g. *every linguist*), wh-expressions (e.g. *which linguist*), and comparative phrases (e.g. *more than five dollars*), whose semantic scope is underdetermined by their syntactic position. One family of approaches, employed by computational semanticists [1] and some versions of categorial grammar [2] and phrase structure grammar [3, 4] employs the **storage** technique first proposed by Cooper [5]. In these approaches, syntactic and semantic derivations proceed in parallel, much as in classical Montague grammar (CMG [6]) except that sentences which differ only with respect to the scope of in-situ operators have identical syntactic derivations.[4] Where they differ is in the semantic derivations: the meaning of an in-situ operator is *stored* together with a copy of the variable that occupies the hole in a delimited semantic continuation over which the stored operator will scope when it is *retrieved*; ambiguity arises from nondeterminism with respect to the retrieval site.

* The authors wish to acknowledge support from the Conseil Régional de Lorraine.

[4] In CMG, syntactic derivations for different scopings of a sentence differ with respect to the point from which a QNP is 'lowered' into the position of a syntactic variable.

Although storage is easily grasped on an intuitive level, it has resisted a clear and convincing logical characterization, and is routinely scorned by theoreticians as 'ad hoc', 'baroque', or 'unprincipled'. Recent work [7, 8] within the CVG framework provided a partial clarification by encoding storage and retrieval rules within a somewhat nonstandard ND semantic calculus (Section 1). The aim of this paper is to provide a logical characterization of storage/retrieval free of nonstandard features. To that end, we provide an explicit transformation of CVG interface derivations (parallel syntax-semantic derivations) into a framework (ACG [9]) that employs no logical resources beyond those of standard (linear) natural deduction. Section 2 provides a preliminary conversion of CVG by showing how to re-express the storage and retrieval rules (respectively) by standard ND hypotheses and another rule already present in CVG (analogous to Gazdar's [10] rule for unbounded dependencies). Section 3 introduces the target framework ACG. And Sect. 4 describes the transformation of a (pre-converted) CVG into an ACG.

1 Convergent Grammar

A CVG for an NL consists of three term calculi for syntax, semantics, and the interface. The syntactic calculus is a kind of applicative multimodal categorial grammar, the semantic calculus is broadly similar to a standard typed lambda calculus, and the interface calculus recursively specifies which syntax-semantics term pairs belong to the NL.[5] Formal presentation of these calculi are given in Appendix A.

In the **syntactic calculus**, types are syntactic categories, constants (nonlogical axioms) are words (broadly construed to subsume phrasal affixes, including intonationally realized ones), and variables (assumptions) are traces (axiom schema T), corresponding to 'overt movement' in generative grammar. Terms are (candidate syntactic analyses of) words and phrases.

For simplicity, we take as our basic syntactic types *np* (noun phrase), *s* (nontopicalized sentence), and *t* (topicalized sentence). Flavors of implication correspond not to directionality (as in Lambek calculus) but to grammatical functions. Thus syntactic arguments are explicitly identitifed as subjects ($-\circ_{\mathrm{s}}$), complements ($-\circ_{\mathrm{c}}$), or hosts of phrasal affixes ($-\circ_{\mathrm{a}}$). Additionally, there is a ternary ('Gazdar') type constructor A^C_B for the category of 'overtly moved' phrases that bind an A-trace in a B, resulting in a C.

Contexts (left of the $\vdash$) in syntactic rules represent unbound traces. The elimination rules (flavors of modus ponens) for the implications, also called merges (M), combine 'heads' with their syntactic arguments. The elimination rule G for the Gazdar constructor implements Gazdar's ([10]) rule for discharging traces; thus G compiles in the effect of a hypothetical proof step (trace binding) immediately and obligatorily followed by the consumption of the resulting abstract by the 'overtly moved' phrase. G requires no introduction rule because it is only

[5] To handle phonology, ignored here, a fourth calculus is needed; and then the interface specifies phonology/syntax/semantics triples.

introduced by lexical items ('overt movement triggers' such as wh-expressions, or the prosodically realized topicalizer).

In the CVG **semantic calculus**, as in familiar semantic λ-calculi, terms correspond to meanings, constants to word meanings, and implication elimination to function application. But there is no λ-abstraction! Instead, binding of semantic variables is effected by either (1) a semantic 'twin' of the Gazdar rule, which binds the semantic variable corresponding to a trace by (the meaning of) the 'overtly moved' phrase; or (2) by the Responsibility (retrieval) rule (R), which binds the semantic variable that marks the argument position of a stored ('covertly moved') *in situ* operator. Correspondingly, there are two mechanisms for introducing semantic variables into derivations: (1) ordinary hypotheses, which are the semantic counterparts of ('overt movement') traces; and the Commitment (Cooper storage) rule (C), which replaces a semantic operator a of type A^C_B with a variable $x : A$ while placing a (subscripted by x) in the store (also called the *co-context*), written to the left of the $\dashv$ (called co-turnstile).

The CVG **interface calculus** recursively defines a relation between syntactic and semantic terms. Lexical items pair syntactic words with their meanings. Hypotheses pair a trace with a semantic variable and enter the pair into the context. The C rule leaves the syntax of an *in situ* operator unchanged while storing its meaning in the co-context. The implication elimination rules pair each (subject-, complement-, or affix-)flavored syntactic implication elimination rule with ordinary semantic implication elimination. The G rule simultaneously binds a trace by an 'overtly moved' syntactic operator and a semantic variable by the corresponding semantic operator. And the R rule leaves the syntax of the retrieval site unchanged while binding a 'committed' semantic variable by the retrieved semantic operator.

2 About the Commitment and Retrieve Rules

In the CVG semantic calculus, C and R are the only rules that make use of the store (co-context), and their logical status is not obvious. This section shows that they can actually be derived from the other rules, in particular from the G rule. Indeed, the derivation on the left can be replaced by the one on the right[6]:

one:

$$\dfrac{\dfrac{\vdots\,\pi_1}{\Gamma \vdash a : A^C_B \dashv \Delta}}{\Gamma \vdash x : A \dashv a_x : A^C_B, \Delta}\,\mathrm{C}$$

$$\vdots\,\pi_2$$

$$\dfrac{\Gamma, \Gamma' \vdash b : B \dashv a_x : A^C_B, \Delta', \Delta}{\Gamma, \Gamma' \vdash a_x\, b : C \dashv \Delta', \Delta}\,\mathrm{R}$$

$$\rightsquigarrow$$

$$\dfrac{\dfrac{}{x : A \vdash x : A \dashv}}{\begin{array}{cc}\vdots\,\pi_1 & \vdots\,\pi_2\\ \Gamma \vdash a : A^C_B \dashv \Delta & x : A, \Gamma' \vdash b : B \dashv \Delta'\end{array}}\,\mathrm{G}$$

$$\Gamma, \Gamma' \vdash a_x\, b : C \dashv \Delta, \Delta'$$

This shows we can eliminate the store, resulting in a more traditional presentation of the underlying logical calculus. On the other hand, in the CVG interface

[6] The fact that we can divide the context into Γ and Γ' and the store into Δ and Δ', and that Γ and Δ are preserved, is shown in Proposition 1 of Appendix B.

calculus, this technique for elimiating C and R rules does not quite go through because the G rule requires both the syntactic type and the semantic type to be of the form α^{γ}_{β}. This difficulty is overcome by adding the following Shift rule to the interface calculus:

$$\frac{\Gamma \vdash a, b : A, B^D_C \dashv \Delta}{\Gamma \vdash \mathrm{S}_E\, a, b : A^E_E, B^D_C \dashv \Delta}\ \mathrm{Shift}_E$$

where S_E is a functional term whose application to an A produces a A^E_E. Then we can transform

$$\dfrac{\dfrac{\vdots \pi_1}{\Gamma \vdash a, b : A, B^D_C \dashv \Delta}\ \mathrm{C}}{\Gamma \vdash a, x : A, B \dashv b_x : B^D_C, \Delta}$$

$$\dfrac{\vdots \pi_2}{\dfrac{\Gamma, \Gamma' \vdash e, c : E, C \dashv b_x : B^D_C, \Delta, \Delta'}{\Gamma, \Gamma' \vdash e, b_x\, c : E, D \dashv \Delta', \Delta}\ \mathrm{R}}$$

to:

$$\dfrac{\dfrac{\dfrac{\vdots \pi_1}{\Gamma \vdash a, b : A, B^D_C \dashv \Delta}\ \mathrm{Shift}_E}{\Gamma \vdash \mathrm{S}_E\, a, b : A^E_E, B^D_C \dashv \Delta} \qquad \dfrac{\dfrac{}{t, x : A, B \vdash t, x : A, B \dashv}}{\vdots \pi_2}}{\Gamma, \Gamma' \vdash (\mathrm{S}_E a)_t\, e, b_x\, c : E, D \dashv \Delta, \Delta'}\ \mathrm{G}$$

provided $(\mathrm{S}_E a)_t\, e = (\mathrm{S}_E\, a)\, (\lambda t.e) = e[t := a]$. This follows from β-reduction as long as we take S_E to be $\lambda y\, P.P\, y$. Indeed:

$$(\mathrm{S}_E\, a)\, (\lambda t.e) = (\lambda y\, P.P\, y)\, a\, (\lambda t.e) =_\beta (\lambda P.P\, a)\, (\lambda t.e) =_\beta (\lambda t.e)\, a =_\beta e[t := a]$$

With this additional construct, we can get rid of the C and R rules in the CVG interface calculus. This construct is used in Section 4 to encode CVG into ACG. It can be seen as a rational reconstruction of Montague's quantifier lowering technique as nothing more than β-reduction in the syntax (unavailable to Montague since his syntactic calculus was purely applicative).

3 Abstract Categorial Grammar

Motivations. Abstract Categorial Grammars (ACGs) [9], which derive from type-theoretic grammars in the tradition of Lambek [11], Curry [12], and Montague [6], provide a framework in which several grammatical formalisms may be encoded [13]. The definition of an ACG is based on a small set of mathematical primitives from type-theory, λ-calculus, and linear logic. These primitives combine via simple composition rules, which offers ACGs a good flexibility. In particular, ACGs generate languages of linear λ-terms, which generalizes both string and tree languages. They also provide the user direct control over the parse structures of the grammar, which allows several grammatical architectures to be defined in terms of ACG.

Mathematical preliminaries. Let A be a finite set of atomic types, and let $\mathscr{T}_A$ be the set of linear functional types types (in notation, $\alpha \multimap \beta$) built upon A. A *higher-order linear signature* is then defined to be a triple $\Sigma = \langle A, C, \tau \rangle$, where: A is a finite set of atomic types; C is a finite set of constants; and τ is a mapping from C to $\mathscr{T}_A$. A higher-order linear signature will also be called a *vocabulary*. In the sequel, we will write A_Σ, C_Σ, and τ_Σ to designate the three components of a signature Σ, and we will write $\mathscr{T}_\Sigma$ for $\mathscr{T}_{A_\Sigma}$.

We take for granted the definition of a λ-term, and we let the relation of $\beta\eta$-conversion to be the notion of equality between λ-terms. Given a higher-order signature Σ, we write Λ_Σ for the set of *linear simply-typed λ-terms*.

Let Σ and Ξ be two higher-order linear signatures. A *lexicon* $\mathscr{L}$ from Σ to Ξ (in notation, $\mathscr{L} : \Sigma \longrightarrow \Xi$) is defined to be a pair $\mathscr{L} = \langle \eta, \theta \rangle$ such that: η is a mapping from A_Σ into $\mathscr{T}_\Xi$; θ is a mapping from C_Σ into Λ_Ξ; and for every $c \in C_\Sigma$, the following typing judgement is derivable: $\vdash_\Xi \theta(c) : \hat{\eta}(\tau_\Sigma(c))$, where $\hat{\eta} : \mathscr{T}_\Sigma \to \mathscr{T}_\Xi$ is the unique homomorphic extension of η.[7]

Let $\hat{\theta} : \Lambda_\Sigma \to \Lambda_\Xi$ be the unique λ-term homomorphism that extends θ.[8] We will use $\mathscr{L}$ to denote both $\hat{\eta}$ and $\hat{\theta}$, the intended meaning being clear from the context. When Γ denotes a typing environment '$x_1 : \alpha_1, \ldots, x_n : \alpha_n$', we will write $\mathscr{L}(\Gamma)$ for '$x_1 : \mathscr{L}(\alpha_1), \ldots, x_n : \mathscr{L}(\alpha_n)$'. Using these notations, we have that the last condition for $\mathscr{L}$ induces the following property: if $\Gamma \vdash_\Sigma t : \alpha$ then $\mathscr{L}(\Gamma) \vdash_\Xi \mathscr{L}(t) : \mathscr{L}(\alpha)$.

Definition 1. *An* abstract categorial grammar *is a quadruple $\mathscr{G} = \langle \Sigma, \Xi, \mathscr{L}, s \rangle$ where:*

1. *Σ and Ξ are two higher-order linear signatures, which are called the* abstract vocabulary *and the* object vocabulary*, respectively;*
2. *$\mathscr{L} : \Sigma \longrightarrow \Xi$ is a lexicon from the abstract vocabulary to the object vocabulary;*
3. *$s \in \mathscr{T}_\Sigma$ is a type of the abstract vocabulary, which is called the* distinguished type *of the grammar.*

A possible intuition behind this definition is that the object vocabulary specifies the surface structures of the grammars, the abstract vocabulary specifies its abstract parse structures, and the lexicon specifies how to map abstract parse structures to surface structures. As for the distinguished type, it plays the same part as the start symbol of the phrase structures grammars. This motivates the following definitions.

The *abstract language* of an ACG is the set of closed linear λ-terms that are built on the abstract vocabulary, and whose type is the distinguished type:

$$\mathcal{A}(\mathscr{G}) = \{t \in \Lambda_\Sigma \mid \vdash_\Sigma t : s \text{ is derivable}\}$$

On the other hand, the object language of the grammar is defined to be the image of its abstract language by the lexicon:

[7] That is $\hat{\eta}(a) = \eta(a)$ and $\hat{\eta}(\alpha \multimap \beta) = \hat{\eta}(\alpha) \multimap \hat{\eta}(\beta)$.

[8] That is $\hat{\theta}(c) = \theta(c)$, $\hat{\theta}(x) = x$, $\hat{\theta}(\lambda x.\, t) = \lambda x.\, \hat{\theta}(t)$, and $\hat{\theta}(t\, u) = \hat{\theta}(t)\, \hat{\theta}(u)$.

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

It is important to note that, from a purely mathematical point of view, there is no structural difference between the abstract and the object vocabulary: both are higher-order signatures. Consequently, the intuition we have given above is only a possible interpretation of the definition, and one may conceive other possible grammatical architectures. Such an architecture consists of two ACGs sharing the same abstract vocabulary, the object vocabulary of the first ACG corresponding to the syntactic structures of the grammar, and the one of the second ACG corresponding to the semantic structures of the grammar. Then, the common abstract vocabulary corresponds to the transfer structures of the syntax/semantics interface. This is precisely the architecture that the next section will exemplify.

4 ACG encoding of CVG

The Overall Architecture. As Section 1 shows, whether a pair of a syntactic term and a semantic term belongs to the language depends on whether it is derivable from the lexicon in the CVG interface calculus. Such a pair is indeed an *(interface) proof term* corresponding to the derivation. So the first step towards the encoding of CVG into ACG is to provide an abstract language that generates the same proof terms as those of the CVG interface. For a given CVG G, we shall call $\Sigma_{I(G)}$ the higher-order signature that will generate the same proof terms as G. Then, any ACG whose abstract vocabulary is $\Sigma_{I(G)}$ will generate these proof terms. And indeed we will use two ACG sharing this abstract vocabulary to map the (interface) proof terms into syntactic terms and into semantic terms respectively. So we need two other signatures: one allowing us to express the syntactic terms, which we call $\Sigma_{\mathrm{SimpleSyn}(G)}$, and another allowing us to express the semantic terms, which we call $\Sigma_{\mathrm{Log}(G)}$.

Finally, we need to be able to recover the two components of the pair out of the proof term of the interface calculus. This means having two ACG sharing the same abstract language (the closed terms of $\Lambda(\Sigma_{I(G)})$ of some distinguished type) and whose object vocabularies are respectively $\Sigma_{\mathrm{SimpleSyn}(G)}$ and $\Sigma_{\mathrm{Log}(G)}$. Fig. 1 illustrates the architecture with $\mathscr{G}_{\mathrm{Syn}} = \langle \Sigma_{I(G)}, \Sigma_{\mathrm{SimpleSyn}(G)}, \mathscr{L}_{\mathrm{Syn}}, s\rangle$ the first ACG that encodes the mapping from interface proof terms to syntactic terms, and $\mathscr{G}_{\mathrm{Sem}} = \langle \Sigma_{I(G)}, \Sigma_{\mathrm{Log}(G)}, \mathscr{L}_{\mathrm{Log}}, s\rangle$ the second ACG that encodes the mapping from interface proof terms to semantic formulas. It should be clear that this architecture can be extended so as to get phonological forms and conventional logical forms (say, in TY_2) using similar techniques. The latter requires non-linear λ-terms, an extension already available to ACG [14] . So we focus here on the (simple) syntax-semantics interface only, which requires only linear terms.

We begin by providing an example of a CVG lexicon (Table 1). Recall that the syntactic type t is for overtly topicalized sentences, and $\multimap a$ is the flavor of implication for affixation. We recursively define the translation $\overline{\cdot}^{\tau}$ of CVG pairs of syntactic and semantics types to $\Sigma_{I(G)}$ as:

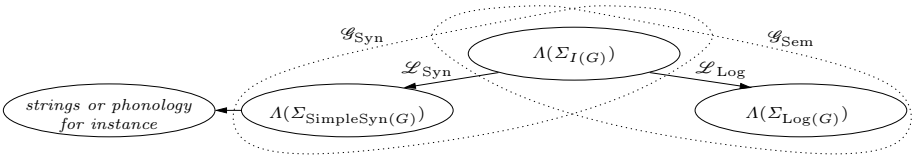

Fig. 1. Overall architecture of the ACG encoding of a CVG

- $\overline{\alpha, \beta}^{\tau} = \langle \alpha, \beta \rangle$ if either α or β is atomic or of the form $\gamma^{\epsilon}_{\delta}$. Note that this new type $\langle a, \beta \rangle$ is an *atomic* type of $\Sigma_{I(G)}$;
- $\overline{\alpha \multimap \beta, \alpha' \multimap \beta'}^{\tau} = \overline{\alpha, \alpha'}^{\tau} \multimap \overline{\beta, \beta'}^{\tau}$[9].

When ranging over the set of types provided by the CVG lexicon[10], we get all the atomic types of $\Sigma_{I(G)}$. Then, for any $w, f : \alpha, \beta$ of the CVG lexicon of G, we add the constant $\overline{w, f}^{\text{c}} = \text{w}$ of type $\overline{\alpha, \beta}^{\tau}$ to the signature $\Sigma_{I(G)}$.

The application of $\overline{\cdot}^{\text{c}}$ and $\overline{\cdot}^{\tau}$ to the lexicon of Table 1 yields the signature $\Sigma_{I(G)}$ of Table 2. Being able to use the constants associated to the topicalization operators in building new terms requires additional constants having e.g. $\langle np, \iota^{\pi}_{\pi} \rangle$ as parameters. We delay this construct to Sect. 4.

Table 1. CVG lexicon for topicalization

$\mathsf{Chris}, \mathsf{Chris'}$:	np, ι		$\mathsf{top}, \mathsf{top'}$	:	$np \multimap_{\text{a}} np^{t}_{s}, \iota \multimap \iota^{\pi}_{\pi}$
$\mathsf{liked}, \mathsf{like'}$	: $np \multimap_{\text{c}} np \multimap_{\text{s}} s, \iota \multimap \iota \multimap \pi$		$\mathsf{top}_{\text{in-situ}}, \mathsf{top'}$:		$np \multimap_{\text{a}} np, \iota \multimap \iota^{\pi}_{\pi}$

Table 2. ACG translation of the CVG lexicon for topicalization

CHRIS :	$\langle np, \iota \rangle$		TOP	:	$\langle np, \iota \rangle \multimap \langle np^{t}_{s}, \iota^{\pi}_{\pi} \rangle$
LIKED :	$\langle np, \iota \rangle \multimap \langle np, \iota \rangle \multimap \langle s, \pi \rangle$		$\text{TOP}_{\text{IN-SITU}}$:		$\langle np, \iota \rangle \multimap \langle np, \iota^{\pi}_{\pi} \rangle$

Constants and types in $\Sigma_{\text{SimpleSyn}(G)}$ and $\Sigma_{\text{Log}(G)}$ simply reflect that we want them to build terms in the syntax and in the semantics respectively. First, note that a term of type α^{γ}_{β}, according to the CVG rules, can be applied to a term of type $\alpha \multimap \beta$ to return a term of type γ. Moreover, the type α^{γ}_{β} does not exist in any of the ACG object vocabularies. Hence we recursively define the $[\![\cdot]\!]$ function that turns CVG syntactic and semantic types into linear types (as used in higher-order signatures) as:

- $[\![a]\!] = a$ if a is atomic
- $[\![\alpha^{\gamma}_{\beta}]\!] = ([\![\alpha]\!] \multimap [\![\beta]\!]) \multimap [\![\gamma]\!]$
- $[\![\alpha \multimap_{\text{x}} \beta]\!] = [\![\alpha]\!] \multimap [\![\beta]\!]$

Then, for any CVG constant $w, f : \alpha, \beta$ we have $\overline{w, f}^{\text{c}} = \text{w} : \overline{\alpha, \beta}^{\tau}$ in $\Sigma_{I(G)}$:

$$\begin{array}{llll} \mathscr{L}_{\text{Syn}}(\text{w}) = w & & \mathscr{L}_{\text{Log}}(\text{w}) = f & \\ \mathscr{L}_{\text{Syn}}(\overline{\alpha, \beta}^{\tau}) = & [\![\alpha]\!] & \mathscr{L}_{\text{Log}}(\overline{\alpha, \beta}^{\tau}) = & [\![\beta]\!] \end{array}$$

[9] This translation preserves the order of the types. Hence, in the ACG settings, it allows abstraction everywhere. This does not fulfill one of the CVG requirements. However, since it is always possible from an ACG $\mathscr{G}$ to build a new ACG $\mathscr{G}'$ such that $\mathcal{O}(\mathscr{G}') = \{t \in \mathcal{A}(\mathscr{G}) | t \text{ consists only in applications}\}$ (see the construct in Appendix C), we can assume without loss of generality that we here deal only with second order terms.

[10] Actually, we should also consider additional types issuing from types of the form α^{γ}_{β} when one of the α, β or γ is itself a type of this form.

So the lexicon of Table 1 gives[11]:

$$\begin{array}{ll} \mathscr{L}_{\mathrm{Syn}}(\textsc{Chris}) = \mathsf{Chris} & \mathscr{L}_{\mathrm{Syn}}(\textsc{liked}) = \lambda xy.\left[^{\mathrm{s}}\, y\ \left[\mathsf{liked}\ x\,^{\mathrm{c}}\right]\right] \\ \mathscr{L}_{\mathrm{Log}}(\textsc{Chris}) = \mathsf{Chris'} & \mathscr{L}_{\mathrm{Log}}(\textsc{liked}) = \lambda xy.\mathsf{like'}\, y\, x \end{array}$$

And we get the trivial translations:

$$\begin{array}{l} \mathscr{L}_{\mathrm{Syn}}(\textsc{liked Sandy Chris}) = \left[^{\mathrm{s}}\mathsf{Chris}\ \left[\mathsf{liked\ Sandy}\,^{\mathrm{c}}\right]\right] : s \\ \mathscr{L}_{\mathrm{Log}}(\textsc{liked Sandy Chris}) = \mathsf{like'\ Chris'\ Sandy'} : \pi \end{array}$$

On the Encoding of CVG Rules. There is a trivial one-to-one mapping between the CVG rules Lexicon, Trace, and Subject and Complement Modus Ponens, and the standard typing rules of linear λ-calculus of ACG: constant typing rule (non logical axiom), identity rule and application. So the ACG derivation that proves $\vdash_{\Sigma_{I(G)}} \textsc{liked Sandy Chris} : \langle s, \pi\rangle$ in $\Lambda(\Sigma_{I(G)})$ is isomorphic to $\vdash \left[^{\mathrm{s}}\mathsf{Chris}\ \left[\mathsf{liked\ Sandy}\,^{\mathrm{c}}\right]\right], \mathsf{like'\ Sandy'\ Chris'} : s, \pi\ \dashv$ as a CVG interface derivation. But the CVG G rule has no counterpart in the ACG type system. So it needs to be introduced using constants in $\Sigma_{I(G)}$.

Let's assume a CVG derivation using the following rule:

$$\frac{\begin{array}{cc} \vdots\,\pi_1 & \vdots\,\pi_2 \\ \Gamma \vdash a, d : A^C_B, D^F_E \dashv \Delta & t, x : A, D; \Gamma' \vdash b, e : B, E \dashv \Delta' \end{array}}{\Gamma; \Gamma' \vdash a_t\, b, d_x\, e : C, F \dashv \Delta; \Delta'}\ \mathrm{G}$$

and that we are able to build two terms (or two ACG derivations) $\textsc{t}_1 : \langle A^C_B, D^F_E\rangle$ and $\textsc{t}_2 : \overline{B, E}^{\,\tau}$ of $\Lambda(\Sigma_{I(G)})$ corresponding to the two CVG derivations π_1 and π_2. Then, adding a constant $G_{\langle A^C_B, D^F_E\rangle}$ of type $\langle A^C_B, D^F_E\rangle \multimap (\overline{A, D}^{\,\tau} \multimap \overline{B, E}^{\,\tau}) \multimap \overline{C, F}^{\,\tau}$ in $\Sigma_{I(G)}$, we can build a new term $G_{\langle A^C_B, D^F_E\rangle}\ \textsc{t}_1\,(\lambda y.\textsc{t}_2) : \overline{C, F}^{\,\tau} \in \Lambda(\Sigma_{I(G)})$. It is then up to the lexicons to provide the good realizations of $G_{\langle A^C_B, D^F_E\rangle}$ so that if $\mathscr{L}_{\mathrm{Syn}}(\textsc{t}_1) = a$, $\mathscr{L}_{\mathrm{Log}}(\textsc{t}_1) = d$, $\mathscr{L}_{\mathrm{Syn}}(\textsc{t}_2) = b$ and $\mathscr{L}_{\mathrm{Log}}(\textsc{t}_2) = e$ then $\mathscr{L}_{\mathrm{Syn}}(G_{\langle A^C_B, D^F_E\rangle}\ \textsc{t}_1\,(\lambda y.\textsc{t}_2)) = a\,(\lambda y.b)$ and $\mathscr{L}_{\mathrm{Log}}(G_{\langle A^C_B, D^F_E\rangle}\ \textsc{t}_1\,(\lambda y.\textsc{t}_2)) = d\,(\lambda y.e)$. This is realized when $\mathscr{L}_{\mathrm{Syn}}(G_{\langle A^C_B, D^F_E\rangle}) = \mathscr{L}_{\mathrm{Log}}(G_{\langle A^C_B, D^F_E\rangle}) = \lambda Q\, R.Q\, R$. A CVG derivation using the (not in-situ) topicalization lexical item and the G rule from $\vdash \left[\mathsf{Sandy\ top}\,^{\mathrm{a}}\right], \mathsf{top'\ Sandy'} : np^t_s, \iota^\pi_\pi\ \dashv$ and from $t, x : np, \iota \vdash \left[^{\mathrm{s}}\mathsf{Chris}\ \left[\mathsf{liked}\ t\,^{\mathrm{c}}\right]\right], \mathsf{like'}\, x\, \mathsf{Chris'} : s, \pi\ \dashv$ would result (conclusion of a G rule) in a proof of $\vdash \left[\mathsf{Sandy\ top}\,^{\mathrm{a}}\right]_t \left[^{\mathrm{s}}\mathsf{Chris}\ \left[\mathsf{liked}\ t\,^{\mathrm{c}}\right]\right], (\mathsf{top'\ Sandy'})_x\,(\mathsf{like'}\, x\, \mathsf{Chris'}) : t, \pi\ \dashv$, the latter being isomorphic to the derivation in $\Lambda(\Sigma_{I(G)})$ proving:
$\vdash_{\Sigma_{I(G)}} G_{\langle np^t_s, \iota^\pi_\pi\rangle}(\textsc{top Sandy})(\lambda x.\textsc{liked}\, x\, \textsc{Chris}) : \langle t, \pi\rangle$. Let's call this term $\textsc{t}$. Then with $\mathscr{L}_{\mathrm{Syn}}(\textsc{top}) = \lambda x.\left[\mathsf{top}\ x\,^{\mathrm{a}}\right] : [\![np \multimap_{\mathrm{a}} np^t_s]\!] = np \multimap (np \multimap s) \multimap t$,

[11] In order to help recognizing the CVG syntactic forms, we use additional operators of arity 2 in $\Sigma_{\mathrm{SimpleSyn}(G)}$: $\left[^{\mathrm{s}}\, s\ p\right]$ instead of writing $(p\, s)$ when p is of type $\alpha \multimap_{\mathrm{s}} \beta$ and $\left[p\ c\,^{\mathrm{x}}\right]$ instead of just $(p\, c)$ when p is of type $\alpha \multimap_{\mathrm{x}} \beta$ with $x \neq s$. This syntactic sugar is not sufficient to model the different flavors of the implication in CVG, the latter topic being beyond the scope of this paper.

$\mathscr{L}_{\mathrm{Log}}(\textsc{top}) = \mathsf{top'} : [\![\iota \multimap \iota^{\pi}_{\pi}]\!] = \iota \multimap (\iota \multimap \pi) \multimap \pi$, and $\mathscr{L}_{\mathrm{Syn}}(G_{\langle np^{t}_{S}, \iota^{\pi}_{\pi}\rangle}) = \mathscr{L}_{\mathrm{Log}}(G_{\langle np^{t}_{S}, \iota^{\pi}_{\pi}\rangle}) = \lambda P\, Q.P\, Q$, we have the expected result:

$$\mathscr{L}_{\mathrm{Syn}}(t) = \left[\mathsf{Sandy\ top}^{\,\mathrm{a}}\right](\lambda x.\left[^{\mathrm{s}}\mathsf{Chris}\ \left[\mathsf{liked}\ x^{\,\mathrm{c}}\right]\right])$$
$$\mathscr{L}_{\mathrm{Log}}(t) = (\mathsf{top'\ Sandy'})(\lambda x.\mathsf{like'}\, x\, \mathsf{Chris'})$$

The C and R Rules. Section 2 shows how we can get rid of the C and R rules in CVG derivations. It brings into play an additional Shift rule and an additional operator S. It should be clear from the previous section that we could add an abstract constant corresponding to this Shift rule. The main point is that its realization in the syntactic calculus by $\mathscr{L}_{\mathrm{Syn}}$ should be $\mathrm{S} = \lambda e\, P.P\, e$ and its realization in the semantics by $\mathscr{L}_{\mathrm{Log}}$ should be the identity.

Technically, it would amount to have a new constant $\mathrm{S}_{\langle A, B^D_C\rangle} : \langle a, B^D_C\rangle \multimap \langle A^E_E, B^D_C\rangle$ such that $\mathscr{L}_{\mathrm{Log}}(\mathrm{S}_{\langle A, B^D_C\rangle}) = \lambda x.x : [\![B^D_C]\!] \multimap [\![B^D_C]\!]$ (this rule does not change the semantics) and $\mathscr{L}_{\mathrm{Syn}}(\mathrm{S}_{\langle A, B^D_C\rangle}) = \lambda x\, P.P\, x : [\![A]\!] \multimap ([\![A]\!] \multimap [\![E]\!]) \multimap [\![E]\!]$ (this rule shift the syntactic type). But since this Shift rule is meant to occur together with a G rule to model C and R, the kind of term we will actually consider is: $t = G_{\langle A^E_E, B^D_C\rangle}(\mathrm{S}_{\langle A, B^D_C\rangle}\, x)\, Q$ for some $x : \langle A, B^D_C\rangle$ and $Q : \langle A^E_E E, B^D_C\rangle$. And the interpretations of t in the syntactic and in the semantic calculus are:

$$\begin{array}{ll}
\mathscr{L}_{\mathrm{Log}}(t) = (\lambda P\, Q.P\, Q) & \mathscr{L}_{\mathrm{Syn}}(t) = (\lambda P\, Q.P\, Q) \\
\qquad ((\lambda y.y)\mathscr{L}_{\mathrm{Log}}(x))\mathscr{L}_{\mathrm{Log}}(Q) & \qquad ((\lambda e P.P\, e)\mathscr{L}_{\mathrm{Syn}}(x))\mathscr{L}_{\mathrm{Syn}}(Q) \\
\qquad = \mathscr{L}_{\mathrm{Log}}(x)\, \mathscr{L}_{\mathrm{Log}}(Q) & \qquad = \mathscr{L}_{\mathrm{Syn}}(Q)\, \mathscr{L}_{\mathrm{Syn}}(x)
\end{array}$$

So basically, $\mathscr{L}_{\mathrm{Log}}(\lambda x\, Q.t) = \mathscr{L}_{\mathrm{Log}}(G_{\langle A^E_E E, B^D_C\rangle})$, and this expresses that nothing new happens on the semantic side, while $\mathscr{L}_{\mathrm{Syn}}(\lambda x\, Q.t) = \lambda x\, Q.Q\, x$ expresses that, somehow, the application is reversed on the syntactic side.

Rather than adding these new constants S (for each type), we integrate their interpretation into the associated G constant[12]. This amounts to compiling the composition of the two terms. So if we have a pair of type A, B^D_C occurring in a CVG G, we add to $\Sigma_{I(G)}$ a new constant $G^{\mathrm{S}}_{\langle A, B^D_C\rangle} : \langle A, B^D_C\rangle \multimap (\overline{\langle A, B\rangle}^{\,\tau} \multimap \overline{\langle E, C\rangle}^{\,\tau}) \multimap \overline{\langle E, D\rangle}^{\,\tau}$ (basically the above term t) whose interpretations are: $\mathscr{L}_{\mathrm{Syn}}(G^{\mathrm{S}}_{\langle A, B^D_C\rangle}) = \lambda P\, Q.Q\, P$ and $\mathscr{L}_{\mathrm{Syn}}(G^{\mathrm{S}}_{\langle A, B^D_C\rangle}) = \lambda P\, Q.P\, Q$.

For instance, if we now use the in-situ topicalizer of Table 1 (triggered by stress for instance), from $\vdash \mathrm{S}_S\left[\mathsf{Sandy\ top}_{\mathsf{in\text{-}situ}}{}^{\mathrm{a}}\right], \mathsf{top'\ Sandy'} : np^S_S, \iota^{\pi}_{\pi} \dashv$ and $t, x : np, \iota \vdash \left[^{\mathrm{s}}\mathsf{Chris}\ \left[\mathsf{liked}\ t^{\,\mathrm{c}}\right]\right], \mathsf{like'}\, x\, \mathsf{Chris'} : s, \pi \dashv$ we can derive, using the G rule, $\vdash (\mathrm{S}_S\left[\mathsf{Sandy\ top}_{\mathsf{in\text{-}situ}}{}^{\mathrm{a}}\right])_t\left[^{\mathrm{s}}\mathsf{Chris}\ \left[\mathsf{liked}\ t^{\,\mathrm{c}}\right]\right], (\mathsf{top'\ Sandy'})_x(\mathsf{like'}\, x\, \mathsf{Chris'}) : s, \pi \dashv$ Note that:

$$\begin{array}{rl}
(\mathrm{S}_S\left[\mathsf{Sandy\ top}_{\mathsf{in\text{-}situ}}{}^{\mathrm{a}}\right])_t(\left[^{\mathrm{s}}\mathsf{Chris}\ \left[\mathsf{liked}\ t^{\,\mathrm{c}}\right]\right]) = & ((\lambda e\, P.P\, e)\left[\mathsf{Sandy\ top}_{\mathsf{in\text{-}situ}}{}^{\mathrm{a}}\right]) \\
& (\lambda t.\left[^{\mathrm{s}}\mathsf{Chris}\ \left[\mathsf{liked}\ t^{\,\mathrm{c}}\right]\right]) \\
=_{\beta} & \left[^{\mathrm{s}}\mathsf{Chris}\ \left[\mathsf{liked}\ \left[\mathsf{Sandy\ top}_{\mathsf{in\text{-}situ}}{}^{\mathrm{a}}\right]^{\,\mathrm{c}}\right]\right]
\end{array}$$

[12] It correspond to the requirement that the Shift rule occurs just before the G rule in the modeling the interface C and R rule with the the G rule.

In order to map this derivation to an ACG term, we use the constant $\textsc{top}_{\textsc{in-situ}} : \langle np, \iota\rangle \multimap \langle np, \iota^{\pi}_{\pi}\rangle$ and the constant that will simulate the G rule and the Shift rule together $G^{\mathrm{S}}_{\langle np,\iota^{\pi}_{\pi}\rangle} : \langle np, \iota^{\pi}_{\pi}\rangle \multimap (\langle np, \iota\rangle \multimap \langle s, \pi\rangle) \multimap \langle s, \pi\rangle$ such that, according to what precedes: $\mathscr{L}_{\mathrm{Syn}}(G^{\mathrm{S}}_{\langle np,\iota^{\pi}_{\pi}\rangle}) = \lambda P\, Q.Q\, P$ and $\mathscr{L}_{\mathrm{Log}}(G^{\mathrm{S}}_{\langle np,\iota^{\pi}_{\pi}\rangle}) = \lambda P\, Q.P\, Q$. Then the previous CVG derivation corresponds to the following term of $\Lambda(\Sigma_{I(G)})$: $t = G^{\mathrm{S}}_{\langle np,\iota^{\pi}_{\pi}\rangle}(\textsc{top}_{\textsc{in-situ}}\ \textsc{Sandy})(\lambda x.\textsc{liked}\, x\, \textsc{Chris})$ and its expected realizations as syntactic and semantic terms are:

$$\begin{array}{rl} \mathscr{L}_{\mathrm{Syn}}(t) = (\lambda P\, Q.Q\, P)(\left[\mathsf{Sandy\ top}_{\mathsf{in\text{-}situ}}{}^{\mathsf{a}}\right]) & \mathscr{L}_{\mathrm{Log}}(t) = (\lambda P\, Q.P\, Q)(\mathsf{top'\ Sandy'}) \\ \quad (\lambda x.\left[{}^{\mathsf{s}}\mathsf{Chris}\ \left[\mathsf{liked}\ x\ {}^{\mathsf{c}}\right]\right]) & \quad (\lambda x, \mathsf{like'}\, x\, \mathsf{Chris'}) \\ = \left[{}^{\mathsf{s}}\mathsf{Chris}\ \left[\mathsf{liked}\ \left[\mathsf{Sandy\ top}_{\mathsf{in\text{-}situ}}{}^{\mathsf{a}}\right]\ {}^{\mathsf{c}}\right]\right] & = (\mathsf{top'\ Sandy'})(\lambda x.\mathsf{like'}\, x\, \mathsf{Chris'}) \end{array}$$

Finally the $G_{\langle\alpha,\beta\rangle}$ and $G^{\mathrm{S}}_{\langle\alpha,\beta\rangle}$ are the only constants of the abstract signature having higher-order types. Hence, they are the only ones that will possibly trigger abstractions, fulfilling the CVG requirement.

When used in quantifier modeling, ambiguities are dealt with in CVG by the non determinism of the order in which semantic operators are retrieved from the store. It corresponds to the (reverse) order in which their ACG encoding are applied in the final term. However, by themselves, both accounts don't provide control on this order. Hence, when several quantifiers occur in the same sentence, all the relative orders of the quantifiers are possible.

Conclusion

We have shown how to encode a linguistically motivated *parallel* formalism, CVG, into a framework, ACG, that has mainly been used to encode syntactocentric formalisms until now. In addition to providing a logical basis for the CVG store mechanism, this encoding also sheds light on the various components (such as higher-order signatures) that are used in the interface calculus. It is noteworthy that the signature used to generate the interface proof terms relate to what is usually called *syntax* in mainstream categorial grammar, whereas the CVG *simple syntax* calculus is not expressed in such frameworks (while it can be using ACG, see [15]).

References

1. Blackburn, P., Bos, J.: Representation and Inference for Natural Language. A First Course in Computational Semantics. CSLI (2005)
2. Bach, E., Partee, B.H.: Anaphora and semantic structure. (1980) Reprinted in Barbara H. Partee, Compositionality in Formal Semantics (Blackwell), pp. 122-152.
3. Cooper, R.: Quantification and Syntactic Theory. Reidel, Dordrecht (1983)
4. Pollard, C., Sag, I.A.: Head-Driven Phrase Structure Grammar. CSLI Publications, Stanford, CA (1994) Distributed by University of Chicago Press.

5. Cooper, R.: Montague's Semantic Theory and Transformational Syntax. PhD thesis, University of Massachusetts at Amherst (1975)
6. Montague, R.: The proper treatment of quantification in ordinary english. In Hintikka, J., Moravcsik, J., Suppes, P., eds.: Approaches to natural language: proceedings of the 1970 Stanford workshop on Grammar and Semantics, Dordrecht, Reidel (1973)
7. Pollard, C.: Covert movement in logical grammar. Submitted
8. Pollard, C.: The calculus of responsibility and commitment. Submitted
9. de Groote, P.: Towards abstract categorial grammars. In: Association for Computational Linguistics, 39th Annual Meeting and 10th Conference of the European Chapter, Proceedings of the Conference. (2001) 148–155
10. Gazdar, G.: Unbounded dependencies and coordinate structure. Linguistic Inquiry **12** (1981) 155–184
11. Lambek, J.: The mathematics of sentence structure. Amer. Math. Monthly **65** (1958) 154–170
12. Curry, H.: Some logical aspects of grammatical structure. In Jakobson, R., ed.: Studies of Language and its Mathematical Aspects, Providence, Proc. of the 12th Symp. Appl. Math.. (1961) 56–68
13. de Groote, P., Pogodalla, S.: On the expressive power of abstract categorial grammars: Representing context-free formalisms. Journal of Logic, Language and Information **13**(4) (2004) 421–438 http://hal.inria.fr/inria-00112956/fr/.
14. de Groote, P., Maarek, S.: Type-theoretic extensions of abstract categorial grammars. In: New Directions in Type-Theoretic Grammars, proceedings of the workshop. (2007) 18–30 http://let.uvt.nl/general/people/rmuskens/ndttg/ndttg2007.pdf.
15. Pogodalla, S.: Generalizing a proof-theoretic account of scope ambiguity. In Geertzen, J., Thijsse, E., Bunt, H., Schiffrin, A., eds.: Proceedings of the 7th International Workshop on Computational Semantics - IWCS-7, Tilburg University, Deparment of Communication and Information Sciences (2007) 154–165 http://hal.inria.fr/inria-00112898.
16. Hinderer, S.: Automatisation de la construction smantique dans TYn. PhD thesis, Université Henri Poincaré – Nancy 1 (2008)

A The CVG calculi

A.1 The CVG syntactic calculus

$$\frac{}{\vdash a : A}\ \text{Lex} \qquad \frac{}{t : A \vdash t : A}\ \text{T}\ (t \text{ fresh})$$

$$\frac{\Gamma \vdash b : A \multimap_{\mathrm{s}} B \quad \Delta \vdash a : A}{\Gamma, \Delta \vdash \left[^{\mathrm{s}} a\ b\right] : B}\ \mathrm{M}_s \qquad \frac{\Gamma \vdash b : A \multimap_{\mathrm{c}} B \quad \Delta \vdash a : A}{\Gamma, \Delta \vdash \left[b\ a\,^{\mathrm{c}}\right] : B}\ \mathrm{M}_c$$

$$\frac{\Gamma \vdash b : A \multimap_{\mathrm{a}} B \quad \Delta \vdash a : A}{\Gamma, \Delta \vdash \left[b\ a\,^{\mathrm{a}}\right] : B}\ \mathrm{M}_a$$

$$\frac{\Gamma \vdash a : A^C_B \quad t : A; \Gamma' \vdash b : B}{\Gamma; \Gamma' \vdash a_t\, b : C}\ \mathrm{G}$$

A.2 The CVG semantic calculus

$$\frac{}{\vdash a : A \dashv}\ \text{Lex} \qquad \frac{}{x : B \vdash x : B \dashv}\ \text{T } (x \text{ fresh})$$

$$\frac{\vdash f : A \multimap B \dashv \Delta \quad \vdash a : A \dashv \Delta'}{\vdash (f\, a) : B \dashv \Delta; \Delta'}\ \text{M}$$

$$\frac{\Gamma \vdash a : A^C_B \dashv \Delta \quad x : A; \Gamma' \vdash b : B \dashv \Delta'}{\Gamma; \Gamma' \vdash a_x\, b : C \dashv \Delta; \Delta'}\ \text{G}$$

$$\frac{\vdash a : A^C_B \dashv \Delta}{\vdash x : A \dashv a_x : A^C_B; \Delta}\ \text{C } (x \text{ fresh}) \qquad \frac{\vdash b : B \dashv a_x : A^C_B; \Delta}{\Gamma \vdash (a_x\, b) : C \dashv \Delta}\ \text{R}$$

A.3 The CVG interface calculus

$$\frac{}{\vdash w, c : A, B \dashv}\ \text{Lex} \qquad \frac{}{x, t : A, B \vdash x, t : A, B \dashv}\ \text{T}$$

$$\frac{\Gamma \vdash f, v : A \multimap_{\text{s}} B, C \multimap D \dashv \Delta \quad \Gamma' \vdash a, c : A, C \dashv \Delta'}{\Gamma; \Gamma' \vdash \left[^{\text{s}} a\ f\right], (v\, c) : B, D \dashv \Delta; \Delta'}\ \text{M}_s$$

$$\frac{\Gamma \vdash f, v : A \multimap_{\text{c}} B, C \multimap D \dashv \Delta \quad \Gamma' \vdash a, c : A, C \dashv \Delta'}{\Gamma; \Gamma' \vdash \left[f\ a^{\text{c}}\right], (v\, c) : B, C \dashv \Delta; \Delta'}\ \text{M}_c$$

$$\frac{\Gamma \vdash f, v : A \multimap_{\text{a}} B, C \multimap D \dashv \Delta \quad \Gamma' \vdash a, c : A, C \dashv \Delta'}{\Gamma; \Gamma' \vdash \left[f\ a^{\text{a}}\right], (v\, c) : B, C \dashv \Delta; \Delta'}\ \text{M}_c$$

$$\frac{\Gamma \vdash a, d : A^C_B, D^F_E \dashv \Delta \quad t, x : A, D; \Gamma' \vdash b, e : B, E \dashv \Delta'}{\Gamma; \Gamma' \vdash a_t\, b, d_x\, e : C, F \dashv \Delta; \Delta'}\ \text{G}$$

$$\frac{\Gamma \vdash a, b : A, B^D_C \dashv \Delta}{\Gamma \vdash a, x : A, B \dashv b_x : B^D_C; \Delta}\ \text{C } (x \text{ fresh}) \qquad \frac{\vdash e, c : E, C \dashv b_x : B^D_C; \Delta}{\Gamma \vdash e, (b_x\, c) : E, D \dashv \Delta}\ \text{R}$$

Example of a simple interface derivation:

$$\frac{\begin{array}{c}\vdots \pi \\ \vdash \left[\mathsf{liked\ Sandy}^{\text{c}}\right], \mathsf{like'\ Sandy'} : np \multimap_{\text{s}} s, \iota \multimap \pi \dashv\end{array} \quad \frac{}{\vdash \mathsf{Chris}, \mathsf{Chris} : np, \iota \dashv}\ \text{Lex}}{\vdash \left[^{\text{s}} \mathsf{Chris}\ \left[\mathsf{liked\ Sandy}^{\text{c}}\right]\right], \mathsf{like'\ Sandy'\ Chris'} : s, \pi \dashv}\ \text{M}_s$$

$$\pi = \frac{\frac{}{\vdash \mathsf{liked}, \mathsf{like'} : np \multimap_{\text{c}} np \multimap_{\text{s}} s, \iota \multimap \iota \multimap \pi \dashv}\ \text{Lex} \quad \frac{}{\vdash \mathsf{Sandy}, \mathsf{Sandy'} : np, \iota \dashv}\ \text{Lex}}{\vdash \left[\mathsf{liked\ Sandy}^{\text{c}}\right], \mathsf{like'\ Sandy'} : np \multimap_{\text{s}} s, \iota \multimap \pi \dashv}\ \text{M}_c$$

Example using the G rule

$$\frac{\begin{array}{c}\vdots \pi_1 \\ \vdash \left[\mathsf{Sandy\ top}^{\text{a}}\right], \mathsf{top'\ Sandy'} : np^t_s, \iota^\pi_\pi \dashv\end{array} \quad \begin{array}{c}\vdots \pi_2 \\ t, x : np, \iota \vdash \left[^{\text{s}} \mathsf{Chris}\ \left[\mathsf{liked}\ t^{\text{c}}\right]\right], \mathsf{like'}\ x\ \mathsf{Chris'} : s, \pi \dashv\end{array}}{\vdash \left[\mathsf{Sandy\ top}^{\text{a}}\right](\lambda t.\left[^{\text{s}} \mathsf{Chris}\ \left[\mathsf{liked}\ t^{\text{c}}\right]\right]), (\mathsf{top'\ Sandy'})(\lambda x.\mathsf{like'}\ x\ \mathsf{Chris'}) : t, \pi \dashv}\ \text{G}$$

with trivial derivations for π_1 and π_2.

B On CVG derivations

Proposition 1. *Let π be a CVG semantic derivation. It can be turned into a CVG semantic derivation where all C and R pairs of rule have been replaced by the above schema, and which derives the same term.*

Proof. This is proved by induction on the derivations. If the derivation stops on a Lexicon, Trace, Modus Ponens, G or C rule, this is trivial by application of the induction hypothesis.

If the derivation stops on a R rule, the C and R pair has the above schema. Note that nothing can be erased from Γ in π_2 because every variable in Γ occur (freely) only in a and Δ. So using a G rule (the only one that can delete material from the left hand side of the sequent) would leave variables in the store that could not be bound later. The same kind of argument shows that nothing can be retrieved from Δ before a_x had been retrieved. This means that no R rule can occur in π_2 whose corresponding C rule is in π_1 (while there can be a R rule with a corresponding C rule introduced in π_2). Hence we can make the transform and apply the induction hypothesis to the two premises of the new G rule.

C How to build an applicative ACG

Let $\Sigma_{\text{HO}} = \langle A_{\text{HO}}, C_{\text{HO}}, \tau_{\text{HO}} \rangle$. This section section shows how to build an ACG $\mathscr{G} = \langle \Sigma_{2\text{nd}}, \Sigma_{\text{HO}}, \mathscr{L}, s' \rangle$ such that $\mathcal{O}(\mathscr{G})$ is the set of $t : s \in \Lambda_{\Sigma_{\text{HO}}}$ such that there exists π a proof of $\vdash_{\Sigma_{\text{HO}}} t : s$ and π does not use the abstraction rule. This construction is very similar to the one given in [16, Chap. 7].

Definition 2. *Let α be a type. We inductively define the set $\mathsf{Decompose}(\alpha)$ as:*

- *if α is atomic, $\mathsf{Decompose}(\alpha) = \{\alpha\}$;*
- *if $\alpha = \alpha_1 \multimap \alpha_2$, $\mathsf{Decompose}(\alpha) = \{\alpha\} \cup \{\alpha_1\} \cup \mathsf{Decompose}(\alpha_2)$.*

Let T be a set of types. We then define:

- $\mathsf{Base}(T) = \cup_{\alpha \in T} \mathsf{Decompose}(T)$;
- $\mathsf{At}(T)$ *a set of fresh atomic types that is in a one to one correspondence with* $\mathsf{Base}(T)$. *We note* $:=$ *one of the correspondence from* $\mathsf{At}(T)$ *to* $\mathsf{Base}(t)$ *(we also note* $:=$ *its unique homomorphic extension that is compatible with* $\multimap$. *The later is not necessarily a bijection);*
- *let* $\alpha \in \mathsf{Base}(T)$. *The set* $\mathsf{AtP}_T(\alpha)$ *of its* atomic profiles *is inductively defined as:*
  - *if α is atomic, $\mathsf{AtP}_T(\alpha) = \{\alpha'\}$ such that α' is the unique element of $\mathsf{At}(T)$ and $\alpha' := \alpha$;*
  - *if $\alpha = \alpha_1 \multimap \alpha_2$, $\mathsf{AtP}_T(\alpha) = \{\alpha'\} \cup \{\alpha'_1 \multimap \alpha'_2 \mid \alpha'_2 \in \mathsf{AtP}_T(\alpha_2)\}$ where:*
    - *α' is uniquely defined in $\mathsf{At}(T)$ and $\alpha' := \alpha$;*
    - *α'_1 is uniquely defined in $\mathsf{At}(T)$ and $\alpha'_1 := \alpha_1$. There exists such an α'_1 because $\alpha_1 \in \mathsf{Decompose}(\alpha)$ and $\mathsf{Decompose}(\alpha) \subset \mathsf{Base}(T)$ when $\alpha \in \mathsf{Base}(T)$.*

Note that for the same reason, α'_2 is well defined.

Note that for any $\alpha \in \mathsf{Base}(T)$, The types in $\mathsf{AtP}_T(\alpha)$ are of order at most 2.

Proposition 2. *Let T be a set of types and $\alpha \in \mathsf{Base}(T)$ with $\alpha = \alpha_1 \multimap \ldots \multimap \alpha_k \multimap \alpha_0$ such that α_0 is atomic. Then $|\mathsf{AtP}_T(\alpha)| = k + 1$.*

Proof. By induction.

Proposition 3. *Let T be a set of types and $\alpha \in \mathsf{Base}(T)$. Then for all $\alpha' \in \mathsf{AtP}_T(\alpha)$ we have $\alpha' := \alpha$.*

Proof. By induction.

In the following, we always consider $T = \cup_{c \in C_{\mathrm{HO}}} \tau_{\mathrm{HO}}(c)$. We then can define $\Sigma_{2\mathrm{nd}} = \langle A_{2\mathrm{nd}}, C_{2\mathrm{nd}}, \tau_{2\mathrm{nd}} \rangle$ with:

- $A_{2\mathrm{nd}} = \mathsf{At}(T)$
- $s' \in A_{2\mathrm{nd}}$ the unique term such that $s' := s$
- $C_{2\mathrm{nd}} = \cup_{c \in C_{\mathrm{HO}}} \{\langle c, \alpha' \rangle | \alpha' \in \mathsf{AtP}_T(\tau_{\mathrm{HO}}(c))\}$ ($\mathsf{AtP}_T(\tau_{\mathrm{HO}}(c))$ is well defined because $\tau_{\mathrm{HO}}(c) \in \mathsf{Base}(T)$)
- for every $c' = \langle c, \alpha' \rangle \in C_{2\mathrm{nd}}$, $\tau_{2\mathrm{nd}}(c') = \alpha'$

Note that according to Proposition 2, for every constant c of C_{HO} of arity k (i.e. $\tau_{\mathrm{HO}}(c) = \alpha_1 \multimap \ldots \multimap \alpha_k \multimap \alpha_0$), there are $k + 1$ constants in $C_{2\mathrm{nd}}$.

Finally, in order to completely define $\mathscr{G}$, we need to define $\mathscr{L}$:

- for $\alpha' \in A_{2\mathrm{nd}}$, there exists a unique $\alpha \in \mathsf{Base}(T)$ such that $\alpha' := \alpha$ by construction of $\mathsf{At}(T)$. We set $\mathscr{L}(\alpha') = \alpha$.
- for $c' = \langle c, \alpha' \rangle \in C_{2\mathrm{nd}}$, we set $\mathscr{L}(c') = c$

According to Proposition 3, we have $\mathscr{L}(\tau_{2\mathrm{nd}}(c')) = \alpha$ where α is the type of $\mathscr{L}(c')$ so $\mathscr{L}$ is well defined.

Proposition 4. *There exists $t : \alpha \in \Lambda_{\Sigma_{\mathrm{HO}}}$ build using only applications if and only if there exists $t' : \alpha'$ a closed term of $\Lambda_{\Sigma_{2\mathrm{nd}}}$ with α' the unique element of $\mathsf{At}(T)$ such that $\alpha' := \alpha$ and $\mathscr{L}(t') = t$.*

Proof. $\Rightarrow$ We prove it by induction on t. If t is a constant, we take $t' = \langle t, \alpha' \rangle$ with α' the unique element of $\mathsf{At}(T)$ such that $\alpha' := \alpha$. By definition, $\mathscr{L}(t') = t$.
If $t = c\, u_1 \ldots u_k$, then $c \in C_{\mathrm{HO}}$ is of type $\alpha_1 \multimap \ldots \multimap \alpha_k \multimap \alpha$ and for all $i \in [1, k]$ u_k is of type α_i. We know there exist $c' = \langle c, \beta' \rangle \in \Sigma_{2\mathrm{nd}}$ such that $\beta' = \alpha'_1 \multimap \ldots \alpha'_k \multimap \alpha'$ with for all $i \in [1, k]$, α'_i is the unique element of $\mathsf{At}(T)$ such that $\alpha'_i := \alpha_i$ and α' the unique element of $\mathsf{At}(T)$ such that $\alpha' := \alpha$. By induction hypothesis, we also have for all $i \in [1, k]$ a term $u'_i : \alpha'_i$ with α'_i the unique element of $\mathsf{At}(T)$ such that $\alpha'_i := \alpha_i$ and $\mathscr{L}(u'_i) = u_i$.
If we take $t' = \langle c, \beta' \rangle\, u'_1 \ldots u'_k$, we have $\mathscr{L}(t') = \mathscr{L}(\langle c, \beta' \rangle\, u'_1 \ldots u'_k) = \mathscr{L}(\langle c, \beta' \rangle)\, \mathscr{L}(u'_1) \ldots \mathscr{L}(u'_k) = c\, u_1 \ldots u_k = t$ which completes the proof.

$\Leftarrow$ If $\alpha' \in \mathsf{At}(T)$ and t' is a closed term then because $\Sigma_{2\mathrm{nd}}$ is of order 2, then t' is build only using applications. Hence its image by $\mathscr{L}$ is also only build using applications.