Fundamenta Informaticae 106 (2011) 211–231
DOI 10.3233/FI-2011-384
IOS Press

About Parallel and Syntactocentric Formalisms: A Perspective from the Encoding of Convergent Grammar into Abstract Categorial Grammar

Philippe de Groote[*], **Sylvain Pogodalla**[*†]
LORIA/INRIA Nancy – Grand Est
615 rue du Jardin Botanique
54602 Villers-lès-Nancy, France
philippe.degroote@loria.fr; sylvain.pogodalla@loria.fr

Carl Pollard[*]
The Ohio State University
202 Oxley Hall Columbus
OH 43210, United States
pollard@ling.ohio-state.edu

Abstract. Recent discussions of grammatical architectures have distinguished two competing approaches to the syntax-semantics interface: *syntactocentrism*, wherein syntactic structures are mapped or transduced to semantics (and phonology), vs. *parallelism*, wherein semantics (and phonology) communicates with syntax via a nondirectional (or relational) interface. This contrast arises for instance in dealing with *in situ* operators. The aim of this paper is threefold: first, we show how the essential content of a parallel framework, convergent grammar (CVG), can be encoded within abstract categorial grammar (ACG), a generic framework which has mainly been used, until now, to encode syntactocentric architectures. Second, using such a generic framework allows us to relate the mathematical characterization of parallelism in CVG with that of syntactocentrism in mainstream categorial grammar (CG), suggesting that the distinction between parallel and syntactocentric formalisms is superficial in nature. More generally, it shows how to provide mildly context sensitive

[*]The authors wish to acknowledge support from the Conseil Régional de Lorraine.
[†]Address for correspondence: LORIA/INRIA Nancy – Grand Est, 615 rue du Jardin Botanique, 54602 Villers-lès-Nancy, France

languages (MCSL), which are a clearly defined class of languages in terms of ACG, with a relational syntax-semantics interface. Finally, while most of the studies on the generative power of ACG have been related to formal languages, we show that ACG can illuminate a linguistically motivated framework such as CVG.

Keywords: Grammatical formalism, type theory, linear logic, lambda calculus, mathematics of language, syntax-semantics interface.

Introduction

Analyzing the evolution of generative grammars (GG), [16] uses the term *syntactocentric* in reference to grammar formalisms in which the syntactic structures (for instance syntactic proofs/derivations) are mapped or transduced to semantics (and phonology), while advocating a different, *parallel* architecture where semantics (and phonology) communicates with syntax via a nondirectional (or relational) interface. Syntactocentric formalisms are exemplified by such frameworks as categorial grammar (CG) [23] and (though in a weaker sense) principles-and-parameters (P&P), while parallel frameworks include HPSG [29], LFG [19], and Simpler Syntax [7].

The emphasis placed on the nature of the syntax-semantics interface relates to the long-standing challenge for designers of NL grammar frameworks 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 [3] and some versions of CG [1] and phrase structure grammar [6, 29] 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 [21]) except that sentences which differ only with respect to the scope of in-situ operators have identical syntactic derivations.[1] 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.

Storage is easily grasped on an intuitive level, and its effect can be simulated as in [4], which builds on [22]'s logical reconstruction of [14]. However, the functional mapping between syntax and semantics of this account makes syntactic ambiguity a requirement for semantic ambiguity. Aiming at preserving a relational correspondence between syntactic terms and semantic terms truer to Cooper's original conception, recent work [28, 27] within the CVG framework provided a partial logical clarification by encoding storage and retrieval rules within a somewhat nonstandard natural-deduction semantic calculus.

In this paper, we first provide a logical characterization of storage/retrieval free of nonstandard features. To that end, we give an explicit transformation of CVG interface derivations (parallel syntax-semantic derivations) into a framework (ACG [11]) that employs no logical resources beyond those of standard (linear) natural deduction.

Second, by relating the encoding of CVG into ACG to the encoding of CG into ACG as in [26], we show how the modelling of covert movement in the CVG *interface* calculus strongly relates to the

[1] In CMG, by contrast, 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.

way it is modeled in the so-called *syntactic* calculus of CG. This underlines the interest of distinguishing between the mathematical apparatus required to achieve some purpose (the encoding of a relation) and the name it is given in different grammatical formalisms (interface or syntax). A generic grammatical framework such as ACG is helpful to make this distinction. Moreover, because the mildly context-sensitive languages (MCSL) are generated by a clearly defined class of ACG, this mathematical apparatus can easily be applied to this class of languages and provide them with a parallel architecture, though at first sight they seem inherently syntactocentric in nature.

The remainder of the paper is organized as follows. In Sect. 1, we introduce the basics of CVG. Section 2 provides a preliminary conversion of CVG by showing how to replace the nonstandard storage and retrieval rules of the semantic calculus by, respectively, standard hypotheses and another rule already present in CVG (analogous to Gazdar's [10] rule for unbounded dependencies). This conversion requires the addition to the CVG interface calculus of a *Shift* rule that raises the *syntactic* type of an in-situ operator to a type similar to that of an 'overtly moved' interrogative wh-expression). Section 3 provides an overview of the target framework ACG. In Sect. 4, we lay out the details of the transformation of a (pre-converted) CVG into an ACG and illustrate it with an example. Further examples are given in Sect. 5.Finally, Sect. 6 gives a more general interpretation of our encoding with respect to the way it could be applied to model semantic ambiguity in multiple context-free grammars and with respect to the role of higher-order types.

1. Convergent Grammar

1.1. Introduction

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.[2]

1.2. CVG Syntactic Calculus

In the **syntactic calculus**, given in Table 1, types are syntactic categories, constants (non-logical 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 analysis 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 ($\multimap_{\mathrm{s}}$), complements ($\multimap_{\mathrm{c}}$), or hosts of phrasal affixes ($\multimap_{\mathrm{a}}$). Additionally, there is a ternary type constructor G (written A^C_B) for the category of 'overtly moved' phrases that bind an A-trace in a B, resulting in a C.

Contexts (to the 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 for the ternary constructor G, inspired by Gazdar's ([10]) rule

[2]To handle phonology, ignored here, a fourth calculus would be needed; and then the interface would specify phonology/syntax/semantics triples.

for discharging traces[3] 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 introduced by lexical items ('overt movement triggers' such as wh-expressions, or the prosodically realized topicalizer).

Table 1. CVG syntactic calculus

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

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

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

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

1.3. CVG Semantic Calculus

In the CVG **semantic calculus**, given in Table 2, as in familiar semantic λ-calculi, terms correspond to meanings, constants to word meanings, and implication elimination to function application. But just as the CVG syntactic calculus has no introduction rule for the ternary G constructor, correspondingly the semantic calculus lacks the familiar rule for hypothetical proof (λ-abstraction). Instead, binding of semantic variables is effected by either (1) a semantic 'twin' of the syntactic G 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

Table 2. CVG semantic calculus

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

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

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

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

[3]With the context corresponding to the value of Gazdar's SLASH feature, the major premiss to the "filler" constituent, and the minor premiss to the "gappy" constituent.

('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).

1.4. CVG Interface Calculus

The CVG **interface calculus**, given Table 3, 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. Example 1.1 demonstrates the use of this R rule in scope ambiguity modelling.

Table 3. CVG interface calculus

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

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

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

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

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

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

Example 1.1. Let's assume the following CVG lexicon:

liked, like'	: $np \multimap_{\mathrm{c}} np \multimap_{\mathrm{s}} s, \iota \multimap \iota \multimap \pi$		
everyone, ev'	: np, ι^π_π	someone, so'	: np, ι^π_π

We can build the following derivations:

$$\pi''_{\text{quant}} = \dfrac{\dfrac{}{\vdash \mathsf{liked}, \mathsf{like'} : np \multimap_{\mathrm{c}} np \multimap_{\mathrm{s}} s, \iota \multimap \iota \multimap \pi \dashv}\,\text{Lex} \quad \dfrac{\dfrac{}{\vdash \mathsf{someone}, \mathsf{so'} : np, \iota^{\pi}_{\pi} \dashv}\,\text{Lex}}{\vdash \mathsf{someone}, x : np, \iota \dashv \mathsf{so'}_x : \iota^{\pi}_{\pi}}\,\text{C}}{\vdash \left[\mathsf{liked\ someone}^{\,c}\right], \mathsf{like'}\, x : np \multimap_{\mathrm{s}} s, \iota \multimap \pi \dashv \mathsf{so'}_x : \iota^{\pi}_{\pi}}\,\text{C}$$

$$\pi'_{\text{quant}} = \dfrac{\begin{array}{c}\vdots \pi''_{\text{quant}} \\ \vdash \left[\mathsf{liked\ someone}^{\,c}\right], \mathsf{like'}\, x : np \multimap_{\mathrm{s}} s, \iota \multimap \pi \dashv \mathsf{so'}_x : \iota^{\pi}_{\pi}\end{array} \quad \dfrac{\dfrac{}{\vdash \mathsf{everyone}, \mathsf{ev'} : np, \iota^{\pi}_{\pi} \dashv}\,\text{Lex}}{\vdash \mathsf{everyone}, y : np, \iota \dashv \mathsf{ev'}_y : \iota^{\pi}_{\pi}}\,\text{C}}{\vdash \left[^{\mathrm{s}}\mathsf{everyone}\ \left[\mathsf{liked\ someone}^{\,c}\right]\right], \mathsf{like'}\, x\, y : s, \pi \dashv \mathsf{so'}_x : \iota^{\pi}_{\pi}, \mathsf{ev'}_y : \iota^{\pi}_{\pi}}\,\text{C}$$

Then we can either have:

$$\pi_{\text{Subj. wide scope}} = \dfrac{\dfrac{\begin{array}{c}\vdots \pi'_{\text{quant}} \\ \vdash \left[^{\mathrm{s}}\mathsf{everyone}\ \left[\mathsf{liked\ someone}^{\,c}\right]\right], \mathsf{like'}\, x\, y : s, \pi \dashv \mathsf{so'}_x : \iota^{\pi}_{\pi}, \mathsf{ev'}_y : \iota^{\pi}_{\pi}\end{array}}{\vdash \left[^{\mathrm{s}}\mathsf{everyone}\ \left[\mathsf{liked\ someone}^{\,c}\right]\right], \mathsf{so'}_x\, (\mathsf{like'}\, x\, y) : s, \pi \dashv \mathsf{ev'}_y : \iota^{\pi}_{\pi}}\,\text{R}}{\vdash \left[^{\mathrm{s}}\mathsf{everyone}\ \left[\mathsf{liked\ someone}^{\,c}\right]\right], \mathsf{ev'}_y(\mathsf{so'}_x(\mathsf{like'}\, x\, y)) : s, \pi \dashv}\,\text{R}$$

Or:

$$\pi_{\text{Obj. wide scope}} = \dfrac{\dfrac{\begin{array}{c}\vdots \pi'_{\text{quant}} \\ \vdash \left[^{\mathrm{s}}\mathsf{everyone}\ \left[\mathsf{liked\ someone}^{\,c}\right]\right], \mathsf{like'}\, x\, y : s, \pi \dashv \mathsf{so'}_x : \iota^{\pi}_{\pi}, \mathsf{ev'}_y : \iota^{\pi}_{\pi}\end{array}}{\vdash \left[^{\mathrm{s}}\mathsf{everyone}\ \left[\mathsf{liked\ someone}^{\,c}\right]\right], \mathsf{ev'}_y\, (\mathsf{like'}\, x\, y) : s, \pi \dashv \mathsf{so'}_x : \iota^{\pi}_{\pi}}\,\text{R}}{\vdash \left[^{\mathrm{s}}\mathsf{everyone}\ \left[\mathsf{liked\ someone}^{\,c}\right]\right], \mathsf{so'}_x(\mathsf{ev'}_y(\mathsf{like'}\, x\, y)) : s, \pi \dashv}\,\text{R}$$

corresponding to the two readings. Note that the syntactic structure is the same in both cases ($\left[^{\mathrm{s}}\mathsf{everyone}\ \left[\mathsf{liked\ someone}^{\,c}\right]\right]$).

2. About the Commitment and Retrieve Rules

The rules C and R are the only rules of the CVG semantic calculus that manipulate the store (co-context). Although it is quite clear, in a purely mechanical sense, how these rules are supposed to work, it is considerably less clear how they are to be explicated or justified in terms of well-understood logical notions. In this section we show that, in the presence of a new Shift (syntactic type-raising) rule, they can actually be derived from the other rules, in particular from the G rule.

Proposition 2.1. Any CVG semantic derivation π can be transformed into a CVG semantic derivation where all C and R pairs of rule have been replaced by the G rule, 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. □

Then, derivations using C and R rules such as the one one on the left can be replaced by the one on the right:

$$\dfrac{\dfrac{\dfrac{\vdots\,\pi_1}{\Gamma \vdash a : A_B^C \dashv \Delta}}{\Gamma \vdash x : A \dashv a_x : A_B^C, \Delta}\,\mathrm{C} \quad \dfrac{\vdots\,\pi_2}{\Gamma, \Gamma' \vdash b : B \dashv a_x : A_B^C, \Delta', \Delta}}{\Gamma, \Gamma' \vdash a_x\, b : C \dashv \Delta', \Delta}\,\mathrm{R} \quad \rightsquigarrow \quad \dfrac{\dfrac{\vdots\,\pi_1}{\Gamma \vdash a : A_B^C \dashv \Delta} \quad \dfrac{\overline{x : A \vdash x : A \dashv}}{\vdots\,\pi_2}}{\Gamma, \Gamma' \vdash a_x\, b : C \dashv \Delta, \Delta'}\,\mathrm{G}$$

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 a consequence of Proposition 2.1.

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 calculus, this technique for eliminating 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_β^γ. This difficulty is overcome by adding the following Shift rule to the interface calculus:

$$\dfrac{\Gamma \vdash a, b : A, B_C^D \dashv \Delta}{\Gamma \vdash \mathrm{S}_E\, a, b : A_E^E, B_C^D \dashv \Delta}\,\mathrm{Shift}_E$$

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

$$\dfrac{\dfrac{\dfrac{\vdots\,\pi_1}{\Gamma \vdash a, b : A, B_C^D \dashv \Delta}}{\Gamma \vdash a, x : A, B \dashv b_x : B_C^D, \Delta}\,\mathrm{C}}{\Gamma, \Gamma' \vdash e, c : E, C \dashv b_x : B_C^D, \Delta, \Delta'}\,\mathrm{R}$$

to:

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

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 4.3 to encode CVG into ACG. It involves the same 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) that was pioneered by [24].

3. Abstract Categorial Grammar

3.1. Motivations

Abstract Categorial Grammars (ACGs) [11], which derive from type-theoretic grammars in the tradition of Lambek [20], Curry [8], and Montague [21], 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.

3.2. 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.

Definition 3.1. Given a set of atomic type A, the *order* $o(\alpha)$ of a type α of $\mathscr{T}_A$ is inductively defined as:

- $o(\alpha) = 1$ if $\alpha \in A$;
- $o(\alpha \multimap \beta) = \max o(\beta, \alpha + 1)$

The *order* of a typed term is the order of its type.

Definition 3.2. A *higher-order linear signature* is 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 take 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.[4]

Let $\hat{\theta} : \Lambda_\Sigma \to \Lambda_\Xi$ be the unique λ-term homomorphism that extends θ.[5] 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 3.3. 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 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:

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

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. One 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 that of the second ACG corresponding to the semantic structures of the grammar. Then, the common abstract vocabulary corresponds to the syntax/semantics interface. This is precisely the architecture that the next section will exemplify.

An equally important notion related to the ACG definition is the *order* of an ACG.

Definition 3.4. The *order* of an ACG is the maximum of the order of its abstract constants.

The *order* of the *lexicon* of an ACG is the maximum of the order of the realizations of its atomic types.

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

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

The class of 2nd order ACG has been extensively studied, and an important result is that 2nd order ACG characterize the class of multiple context-free-languages [30, 17, 18] for which polynomial parsing algorithms exist [31].

Terms of the abstract language of a 2nd order ACG cannot have an abstraction as subterm. Hence, they can indeed be seen as trees. With that respect, they are closely related to CVG where λ-abstraction is forbidden, so that CVG syntactic structures share this property with parse structures of multiple context-free-languages.

4. ACG Encoding of CVG

4.1. 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 denote by $\Sigma_{I(G)}$ 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 ACG encoding 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 ACG encoding 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 [12] . So we focus here on the (simple) syntax-semantics interface only, which requires only linear terms.

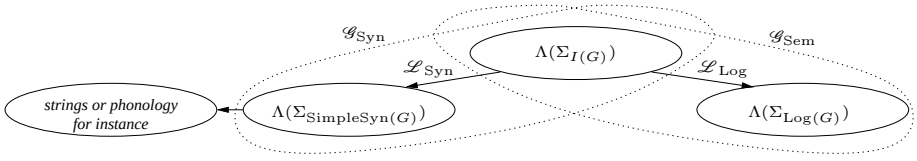

Figure 1. Overall architecture of the ACG encoding of a CVG

We begin by providing an example of a CVG lexicon (Table 4). 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:

- $\overline{\alpha, \beta}^{\tau} = \langle \alpha, \beta \rangle$ 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 $\Sigma_{I(G)}$;

- $\overline{\alpha \multimap \beta, \alpha' \multimap \beta'}^{\tau} = \overline{\alpha, \alpha'}^{\tau} \multimap \overline{\beta, \beta'}^{\tau}$[6].

When inductively ranging over the set of types provided by the CVG lexicon, 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 $\overline{w, f}^{\mathrm{c}} = \mathrm{w}$ of type $\overline{\alpha, \beta}^{\tau}$ to the signature $\Sigma_{I(G)}$.

The application of $\overline{\cdot}^{\mathrm{c}}$ and $\overline{\cdot}^{\tau}$ to the lexicon of Table 4 yields the signature $\Sigma_{I(G)}$ of Table 5. 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. 4.2.

Table 4. CVG lexicon for topicalization

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

Table 5. 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 :	$\langle np, \iota \rangle \multimap \langle np, \iota \rangle \multimap \langle s, \pi \rangle$	$\mathrm{TOP}_{\mathrm{IN\text{-}SITU}}$:	$\langle np, \iota \rangle \multimap \langle np, \iota^{\pi}_{\pi} \rangle$

Constants and types in $\Sigma_{\mathrm{SimpleSyn}(G)}$ and $\Sigma_{\mathrm{Log}(G)}$ 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
- $[\![\alpha^{\gamma}_{\beta}]\!] = ([\![\alpha]\!] \multimap [\![\beta]\!]) \multimap [\![\gamma]\!]$
- $[\![\alpha \multimap_{\mathrm{x}} \beta]\!] = [\![\alpha]\!] \multimap [\![\beta]\!]$

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

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

So the lexicon of Table 4 gives[7]:

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

[6] 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 $\mathcal{O}(\mathscr{G}') = \{t \in \mathcal{A}(\mathscr{G}) | t \text{ consists only in applications}\}$ (it's enough to transform $\mathscr{G}$ into $\mathscr{G}'$ as in [15, Chap. 7] where $\mathscr{G}'$ is a second order ACG), we can assume without loss of generality that we here deal only with second order terms.

[7] 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 implication in CVG. While we don't give the proof here, flavors on implication can be simulated with suitable (flavored) types using for all flavors f the translation $\lceil \cdot \rceil_{\mathrm{f}}$ such that $\lceil B \rceil_{\mathrm{f}} = B_{\mathrm{f}}$ if B is an atomic type and $\lceil \beta \multimap_{\mathrm{g}} \gamma \rceil_{\mathrm{f}} = \lceil \beta \rceil_{\mathrm{g}} \multimap \lceil \gamma \rceil_{\mathrm{f}}$.

And we get the trivial translations:

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

4.2. On the Encoding of CVG Rules

4.2.1. Abstraction and Modus Ponens

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 $\vdash_{\Sigma_{I(G)}}$ LIKED SANDY CHRIS : $\langle s, \pi\rangle$ in $\Lambda(\Sigma_{I(G)})$:

$$\dfrac{\dfrac{\dfrac{}{\vdash \textsc{liked} : \langle np \multimap np \multimap s, \iota \multimap \iota \multimap \pi\rangle}\,\text{cons} \quad \dfrac{}{\vdash \textsc{Sandy} : \langle np, \iota\rangle}\,\text{cons}}{\vdash \textsc{liked Sandy} : \langle np \multimap s, \iota \multimap \pi\rangle}\,\text{app} \quad \dfrac{}{\vdash \textsc{Chris} : \langle np, \iota\rangle}\,\text{cons}}{\vdash \textsc{liked Sandy Chris} : \langle s, \pi\rangle}\,\text{app}$$

is isomorphic to $\vdash \left[^{\mathrm{s}}\mathsf{Chris}\ \left[\mathsf{liked\ Sandy}^{\mathrm{c}}\right]\right], \mathsf{like'\ Sandy'\ C} : s, \pi \dashv$ as a CVG interface derivation:

$$\dfrac{\dfrac{\vdots\,\pi}{\vdash \left[\mathsf{liked\ Sandy}^{\mathrm{c}}\right], \mathsf{like'\ Sandy'} : np \multimap_{\mathrm{s}} s, \iota \multimap \pi \dashv} \quad \dfrac{}{\vdash \mathsf{Chris}, \mathsf{Chris} : np, \iota \dashv}\,\text{Lex}}{\vdash \left[^{\mathrm{s}}\mathsf{Chris}\ \left[\mathsf{liked\ Sandy}^{\mathrm{c}}\right]\right], \mathsf{like'\ Sandy'\ C} : s, \pi \dashv}\,\text{s app}$$

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

But the CVG G rule has no counterpart in the ACG type system. So it needs to be introduced using constants in $\Sigma_{I(G)}$.

4.2.2. The G Rule

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

$$\dfrac{\vdots\,\pi_1 \quad \Gamma \vdash a, d : A^C_B, D^F_E \dashv \Delta \qquad \vdots\,\pi_2 \quad t, x : A, D; \Gamma' \vdash b, e : B, E \dashv \Delta'}{\Gamma; \Gamma' \vdash a_t\, b, d_x\, e : C, F \dashv \Delta; \Delta'}\,\mathrm{G}$$

and that we are able to build two terms (or two ACG derivations) $\mathrm{T}_1 : \langle A^C_B, D^F_E\rangle$ and $\mathrm{T}_2 : \overline{B, E}^{\tau}$ 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 $G_{\langle A^C_B, D^F_E\rangle}\ \mathrm{T}_1\ (\lambda y.\mathrm{T}_2) : \overline{C, F}^{\tau} \in \Lambda(\Sigma_{I(G)})$. The use of this constant $G_{\langle A^C_B, D^F_E\rangle}$ is somehow triggered by a term of type $\langle A^C_B, D^F_E\rangle$ (for instance an in-situ operator) and results in a term with a higher-order type reminiscent the one given in CG for the same operators.

It is then up to the lexicons to provide the interpretations of $G_{\langle A_B^C, D_E^F\rangle}$ so that if:

- $\mathscr{L}_{\mathrm{Syn}}(\mathrm{T}_1) = a$,
- $\mathscr{L}_{\mathrm{Log}}(\mathrm{T}_1) = d$,
- $\mathscr{L}_{\mathrm{Syn}}(\mathrm{T}_2) = b$,
- and $\mathscr{L}_{\mathrm{Log}}(\mathrm{T}_2) = e$

then

- $\mathscr{L}_{\mathrm{Syn}}(G_{\langle A_B^C, D_E^F\rangle}\ \mathrm{T}_1\ (\lambda y.\mathrm{T}_2)) = a\ (\lambda y.b)$
- and $\mathscr{L}_{\mathrm{Log}}(G_{\langle A_B^C, D_E^F\rangle}\ \mathrm{T}_1\ (\lambda y.\mathrm{T}_2)) = d\ (\lambda y.e)$.

This is realized when $\mathscr{L}_{\mathrm{Syn}}(G_{\langle A_B^C, D_E^F\rangle}) = \mathscr{L}_{\mathrm{Log}}(G_{\langle A_B^C, D_E^F\rangle}) = \lambda Q\, R.Q\, R$.

A CVG derivation using the (not in-situ) topicalization lexical item and the G rule could for instance be[8]:

$$\frac{\vdots\,\pi_{\text{Sandy top}} \qquad \vdots\,\pi_{\text{Chris liked}}}{\vdash \left[\mathsf{Sandy\ top}^{\,\mathrm{a}}\right], \mathsf{top'\ Sandy'} : np^{t}_{S}, \iota^{\pi}_{\pi} \dashv \quad t, x : np, \iota \vdash \left[{}^{\mathrm{s}}\mathsf{Chris}\ \left[\mathsf{liked}\ t^{\,\mathrm{c}}\right]\right], \mathsf{like'}\ x\ \mathsf{C} : s, \pi \dashv}{\vdash \left[\mathsf{Sandy\ top}^{\,\mathrm{a}}\right](\lambda t.\left[{}^{\mathrm{s}}\mathsf{Chris}\ \left[\mathsf{liked}\ t^{\,\mathrm{c}}\right]\right]), (\mathsf{top'\ Sandy'})(\lambda x.\mathsf{like'}\ x\ \mathsf{C}) : t, \pi \dashv}$$

This is also isomorphic to the derivation in $\Lambda(\Sigma_{I(G)})$ proving:
$\vdash_{\Sigma_{I(G)}} G_{\langle np^{t}_{S}, \iota^{\pi}_{\pi}\rangle}(\textsc{top Sandy})(\lambda x.\textsc{liked}\ x\ \textsc{Chris}) : \langle t, \pi\rangle$. Indeed we have the derivation:

$$\frac{\vdots\,\pi'_{\text{G Sandy top}} \qquad \vdots\,\pi'_{\text{Chris liked}}}{\vdash G_{\langle np^{t}_{S}, \iota^{\pi}_{\pi}\rangle}(\textsc{top Sandy}) : (\langle np, \iota\rangle \multimap \langle s, \pi\rangle) \multimap \langle t, \pi\rangle \quad \vdash \lambda x.\textsc{liked}\ x\ \textsc{Chris} : \langle np, \iota\rangle \multimap \langle s, \pi\rangle}{\vdash G_{\langle np^{t}_{S}, \iota^{\pi}_{\pi}\rangle}(\textsc{top Sandy})(\lambda x.\textsc{liked}\ x\ \textsc{Chris}) : \langle t, \pi\rangle}$$

where:

$$\pi'_{\text{G Sandy top}} = \frac{\dfrac{}{\vdash G_{\langle np^{t}_{S}, \iota^{\pi}_{\pi}\rangle} : \langle np^{t}_{S}, \iota^{\pi}_{\pi}\rangle \multimap (\langle np, \iota\rangle \multimap \langle s, \pi\rangle) \multimap \langle t, \pi\rangle}\ \text{cons} \qquad \vdots\,\pi'_{\text{Sandy top}}\ \vdash \textsc{top Sandy} : \langle np^{t}_{S}, \iota^{\pi}_{\pi}\rangle}{\vdash G_{\langle np^{t}_{S}, \iota^{\pi}_{\pi}\rangle}(\textsc{top Sandy}) : (\langle np, \iota\rangle \multimap \langle s, \pi\rangle) \multimap \langle t, \pi\rangle}$$

$$\pi'_{\text{Sandy top}} = \frac{\dfrac{}{\vdash \textsc{top} : \langle np, \iota\rangle \multimap \langle np^{t}_{S}, \iota^{\pi}_{\pi}\rangle}\ \text{cons} \qquad \dfrac{}{\vdash \textsc{Sandy} : \langle np, \iota\rangle}\ \text{cons}}{\vdash \textsc{top Sandy} : \langle np^{t}_{S}, \iota^{\pi}_{\pi}\rangle}$$

$$\pi'_{\text{Chris liked}} = \frac{\dfrac{\dfrac{\dfrac{}{\vdash \textsc{liked} : \langle np, \iota\rangle \multimap \langle np, \iota\rangle \multimap \langle s, \pi\rangle}\ \text{cons} \qquad \dfrac{}{x : \langle np, \iota\rangle \vdash x : \langle np, \iota\rangle}\ \text{var}}{x : \langle np, \iota\rangle \vdash \textsc{liked}\ x : \langle np, \iota\rangle \multimap \langle s, \pi\rangle}\ \text{app} \qquad \dfrac{}{\vdash \textsc{Chris} : \langle np, \iota\rangle}\ \text{cons}}{x : \langle np, \iota\rangle \vdash \textsc{liked}\ x\ \textsc{Chris} : \langle s, \pi\rangle}\ \text{app}}{\vdash \lambda x.\textsc{liked}\ x\ \textsc{Chris} : \langle np, \iota\rangle \multimap \langle s, \pi\rangle}$$

[8] With trivial derivations for $\pi_{\text{Sandy top}}$ and $\pi_{\text{Chris liked}}$.

Let be $\text{T} = G_{\langle np^t_S, \iota^\pi_\pi \rangle}(\text{TOP SANDY})(\lambda x.\text{LIKED}\, x\, \text{CHRIS}) : \langle t, \pi \rangle$. Then with

- $\mathscr{L}_{\text{Syn}}(\text{TOP}) = \lambda x.\big[\!\!\big[\mathsf{top}\ x\ ^{\text{a}}\big]\!\!\big] : [\![np \multimap_{\text{a}} np^t_S]\!] = np \multimap (np \multimap s) \multimap t$,
- $\mathscr{L}_{\text{Log}}(\text{TOP}) = \mathsf{top'} : [\![\iota \multimap \iota^\pi_\pi]\!] = \iota \multimap (\iota \multimap \pi) \multimap \pi$,
- and $\mathscr{L}_{\text{Syn}}(G_{\langle np^t_S, \iota^\pi_\pi \rangle}) = \mathscr{L}_{\text{Log}}(G_{\langle np^t_S, \iota^\pi_\pi \rangle}) = \lambda P\, Q.P\, Q$,

we have the expected result:

$$\begin{aligned}\mathscr{L}_{\text{Syn}}(t) &= \big[\!\!\big[\mathsf{Sandy\ top}\ ^{\text{a}}\big]\!\!\big](\lambda x.\big[\!\!\big[^{\text{s}}\mathsf{Chris}\ \big[\!\!\big[\mathsf{liked}\ x\ ^{\text{c}}\big]\!\!\big]\ \big]\!\!\big]) \\ \mathscr{L}_{\text{Log}}(t) &= (\mathsf{top'\ Sandy'})(\lambda x.\mathsf{like'}\ x\ \mathsf{C})\end{aligned}$$

4.3. 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 $\mathscr{L}_{\text{Syn}}$ should be $\text{S} = \lambda e\, P.P\, e$ and its realization in the semantics by $\mathscr{L}_{\text{Log}}$ should be the identity.

Technically, this would amount to have a new constant $\text{S}_{\langle A, B^D_C \rangle} : \langle a, B^D_C \rangle \multimap \langle A^E_E, B^D_C \rangle$ such that:

- $\mathscr{L}_{\text{Log}}(\text{S}_{\langle A, B^D_C \rangle}) = \lambda x.x : [\![B^D_C]\!] \multimap [\![B^D_C]\!]$ (this rule does not change the semantics)
- and $\mathscr{L}_{\text{Syn}}(\text{S}_{\langle A, B^D_C \rangle}) = \lambda x\, P.P\, x : [\![A]\!] \multimap ([\![A]\!] \multimap [\![E]\!]) \multimap [\![E]\!]$ (this rule shifts 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: $t = G_{\langle A^E_E, B^D_C \rangle}(\text{S}_{\langle A, B^D_C \rangle}\, x)\, Q$ 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:

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

So basically, $\mathscr{L}_{\text{Log}}(\lambda x\, Q.t) = \mathscr{L}_{\text{Log}}(G_{\langle A^E_E, B^D_C \rangle})$, and this expresses that nothing new happens on the semantic side, while $\mathscr{L}_{\text{Syn}}(\lambda x\, Q.t) = \lambda x\, Q.Q\, x$ 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[9]. 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 $G^{\text{S}}_{\langle A, B^D_C \rangle} : \langle A, B^D_C \rangle \multimap (\overline{\langle A, B \rangle}^{\tau} \multimap \overline{\langle E, C \rangle}^{\tau}) \multimap \overline{\langle E, D \rangle}^{\tau}$ (basically the above term t) whose interpretations are: $\mathscr{L}_{\text{Syn}}(G^{\text{S}}_{\langle A, B^D_C \rangle}) = \lambda P\, Q.Q\, P$ and $\mathscr{L}_{\text{Syn}}(G^{\text{S}}_{\langle A, B^D_C \rangle}) = \lambda P\, Q.P\, Q$.

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

For instance, if we now use the in-situ topicalizer of Table 4 (prosodically realized by a contrastive pitch accent for instance) we can have the following CVG derivation:

$$\dfrac{\vdots\,\pi_{\text{Santy top in-situ}} \qquad \vdots\,\pi_{\text{Chris liked}}}{\dfrac{\vdash \mathrm{S}_S\,\big[\mathsf{Sandy\ top}_{\text{in-situ}}{}^{\mathrm{a}}\big], \mathsf{top'\,Sandy'} : np^S_S, \iota^\pi_\pi \dashv \quad t, x : np, \iota \vdash \big[{}^{\mathrm{s}}\mathsf{Chris}\ \big[\mathsf{liked}\ t^{\,\mathrm{c}}\big]\,\big], \mathsf{like'}\,x\,\mathsf{C} : s, \pi \dashv}{\vdash (\mathrm{S}_S\,\big[\mathsf{Sandy\ top}_{\text{in-situ}}{}^{\mathrm{a}}\big])(\lambda t.\big[{}^{\mathrm{s}}\mathsf{Chris}\ \big[\mathsf{liked}\ t^{\,\mathrm{c}}\big]\,\big]), (\mathsf{top'\,Sandy'})(\lambda x.\mathsf{like'}\,x\,\mathsf{C}) : s, \pi \dashv}$$

with $\pi_{\text{Santy top in-situ}} =$

$$\dfrac{\dfrac{\dfrac{\vdash \mathsf{top}_{\text{in-situ}}, \mathsf{top'} : np \multimap_{\mathrm{a}} np, \iota \multimap \iota^\pi_\pi \dashv}{\text{Lex}} \quad \dfrac{\vdash \mathsf{Sandy}, \mathsf{Sandy'} : np, \iota \dashv}{\text{Lex}}}{\vdash \big[\mathsf{Sandy\ top}_{\text{in-situ}}{}^{\mathrm{a}}\big], \mathsf{top'\,Sandy'} : np, \iota^\pi_\pi \dashv}}{\vdash \mathrm{S}_S\,\big[\mathsf{Sandy\ top}_{\text{in-situ}}{}^{\mathrm{a}}\big], \mathsf{top'\,Sandy'} : np^S_S, \iota^\pi_\pi \dashv}\ \text{Shift}_S$$

Note that:

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

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 $\mathscr{L}_{\mathrm{Log}}(G^{\mathrm{S}}_{\langle np, \iota^\pi_\pi\rangle}) = \lambda P\,Q.P\,Q.$

These syntactic and semantic linearizations of in-situ operators are analogous to the one used in [26] to provide a simple syntax to CG.

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:

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

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. This corresponds to the (reverse) order in which their ACG encoding are applied in the final term. However, as they stand, neither account constrains this order. Hence, when several quantifiers occur in the same sentence, all the relative orders of the quantifiers are possible as the next section exemplifies.

5. Examples

5.1. *De Re* and *De Dicto* Readings

Quantification is a typical example of covert movement where the semantic scope of QNP is underdetermined by their syntactic position. Incidentally, when two (or more) of these expressions occur in the same sentence, the underdetermination gives rise to several possible relative semantic scope. Table 6 gives a straightforward example of how QNP are encoded in CVG: while the syntactic type is *np*, as expected, the semantic type using a Gazdar type constructor ι_π^π.

Table 6. CVG lexicon for quantifier scope fragment

everyone, ev' :	np, ι_π^π	someone, so' :	np, ι_π^π
thought, think' :	$s \multimap np \multimap s, \pi \multimap \iota \multimap \pi$	Kim, K :	np, ι

Let $\pi''_{\text{de dicto}}$ be the following derivation:

$$\dfrac{\dfrac{}{\vdash \mathsf{Kim}, \mathsf{K} : np, \iota \dashv}\,\text{Lex} \quad \dfrac{\dfrac{}{\vdash \mathsf{liked}, \mathsf{like'} : np \multimap_{\mathrm{c}} np \multimap_{\mathrm{s}} s, \iota \multimap \iota \multimap \pi \dashv}\,\text{Lex} \quad \dfrac{\dfrac{}{\vdash \mathsf{everyone}, \mathsf{ev'} : np, \iota_\pi^\pi \dashv}\,\text{Lex}}{\vdash \mathsf{everyone}, x : np, \iota \dashv \mathsf{ev'}_x : \iota_\pi^\pi}\,\text{C}}{\vdash \left[\mathsf{liked}\ \mathsf{everyone}^{\,\mathrm{c}}\right], \mathsf{like'}\, x : np \multimap_{\mathrm{s}} s, \iota \multimap \pi \dashv \mathsf{ev'}_x : \iota_\pi^\pi}}{\vdash \left[^{\mathrm{s}}\mathsf{Kim}\ \left[\mathsf{liked}\ \mathsf{everyone}^{\,\mathrm{c}}\right]\right], \mathsf{like'}\, x\, \mathsf{K} : s, \pi \dashv \mathsf{ev'}_x : \iota_\pi^\pi}\,\text{R}$$

and $\pi'_{\text{de dicto}}$ be:

$$\dfrac{\dfrac{}{\vdash \mathsf{thought}, \mathsf{think'} : s \multimap np \multimap s, \pi \multimap \iota \multimap \pi \dashv}\,\text{Lex} \quad \vdots\,\pi''_{\text{de dicto}}}{\vdash \left[\mathsf{thought}\ \left[^{\mathrm{s}}\mathsf{Kim}\ \left[\mathsf{liked}\ \mathsf{everyone}^{\,\mathrm{c}}\right]\right]^{\mathrm{c}}\right], \mathsf{think'}\,(\mathsf{ev'}_x\,(\mathsf{like'}\, x\, \mathsf{K})) : np \multimap s, \iota \multimap \pi \dashv}$$

and $\pi_{\text{de dicto}}$ be:

$$\dfrac{\dfrac{}{\vdash \mathsf{Chris}, \mathsf{Chris} : np, \iota \dashv}\,\text{Lex} \quad \vdots\,\pi'_{\text{de dicto}}}{\vdash \left[^{\mathrm{s}}\mathsf{Chris}\ \left[\mathsf{thought}\ \left[^{\mathrm{s}}\mathsf{Kim}\ \left[\mathsf{liked}\ \mathsf{everyone}^{\,\mathrm{c}}\right]\right]^{\mathrm{c}}\right]\right], \mathsf{think'}\,(\mathsf{ev'}_x\,(\mathsf{like'}\, x\, \mathsf{K}\mathsf{R}))\,\mathsf{C} : s, \pi \dashv}$$

Then $\pi_{\text{de dicto}}$ is a CVG interface derivation for the narrow scope reading of the sentence *Chris thought Kim liked everyone*.

Let now $\pi''_{\text{de re}}$ be the following derivation:

$$\dfrac{\dfrac{}{\vdash \mathsf{Kim}, \mathsf{K} : np, \iota \dashv}\,\text{Lex} \quad \dfrac{\dfrac{}{\vdash \mathsf{liked}, \mathsf{like'} : np \multimap_{\mathrm{c}} np \multimap_{\mathrm{s}} s, \iota \multimap \iota \multimap \pi \dashv}\,\text{Lex} \quad \dfrac{\dfrac{}{\vdash \mathsf{everyone}, \mathsf{ev'} : np, \iota_\pi^\pi \dashv}\,\text{Lex}}{\vdash \mathsf{everyone}, x : np, \iota \dashv \mathsf{ev'}_x : \iota_\pi^\pi}\,\text{C}}{\vdash \left[\mathsf{liked}\ \mathsf{everyone}^{\,\mathrm{c}}\right], \mathsf{like'}\, x : np \multimap_{\mathrm{s}} s, \iota \multimap \pi \dashv \mathsf{ev'}_x : \iota_\pi^\pi}}{\vdash \left[^{\mathrm{s}}\mathsf{Kim}\ \left[\mathsf{liked}\ \mathsf{everyone}^{\,\mathrm{c}}\right]\right], \mathsf{like'}\, x\, \mathsf{K} : s, \pi \dashv \mathsf{ev'}_x : \iota_\pi^\pi}$$

and $\pi'_{\text{de re}}$:

$$\dfrac{\dfrac{}{\vdash \mathsf{thought}, \mathsf{think'} : s \multimap np \multimap s, \pi \multimap \iota \multimap \pi \dashv}\,\text{Lex} \quad \vdots\,\pi''_{\text{de re}}}{\vdash \left[\mathsf{thought}\ \left[^{\mathrm{s}}\mathsf{Kim}\ \left[\mathsf{liked}\ \mathsf{everyone}^{\,\mathrm{c}}\right]\right]^{\mathrm{c}}\right], \mathsf{think'}\,(\mathsf{like'}\, x\, \mathsf{K}) : np \multimap s, \iota \multimap \pi \dashv \mathsf{ev'}_x : \iota_\pi^\pi}$$

and $\pi_{\text{de re}}$:

$$\dfrac{\dfrac{}{\vdash \mathsf{Chris}, \mathsf{Chris} : np, \iota \dashv}\ \text{Lex} \quad \vdash \left[\mathsf{thought}\ \left[^{\mathrm{s}}\mathsf{Kim}\ \left[\mathsf{liked\ everyone}^{\mathrm{c}}\right]\right]^{\mathrm{c}}\right], \mathsf{think'}\,(\mathsf{like'}\,x\,\mathsf{K}) : np \multimap s, \iota \multimap \pi \dashv \mathsf{ev'}_x : \iota_\pi^\pi \quad \vdots\ \pi'_{\text{de re}}}{\vdash \left[^{\mathrm{s}}\mathsf{Chris}\ \left[\mathsf{thought}\ \left[^{\mathrm{s}}\mathsf{Kim}\ \left[\mathsf{liked\ everyone}^{\mathrm{c}}\right]\right]^{\mathrm{c}}\right]\right], \mathsf{think'}\,(\mathsf{ev'}_x\,(\mathsf{like'}\,x\,\mathsf{KR}))\,\mathsf{C} : s, \pi \dashv \mathsf{ev'}_x : \iota_\pi^\pi}\ \text{R}$$

Then $\pi_{\text{de re}}$ provides the wide scope reading of the sentence *Chris thought Kim liked everyone*.

Using the ACG translation of this CVG fragment given in Table 7, the two derivations $\pi_{\text{de dicto}}$ and $\pi_{\text{de re}}$ correspond to the following terms of type $\langle s, \pi\rangle$ respectively:

- $t_{\text{de dicto}} = \textsc{thought}\,(G^{\mathrm{S}}_{\langle np,\iota_\pi^\pi\rangle}\ \textsc{everyone}\,(\lambda x.\textsc{like}\,x\,\textsc{Kim}))\,\textsc{Chris}$
- $t_{\text{de re}} = G^{\mathrm{S}}_{\langle np,\iota_\pi^\pi\rangle}\ \textsc{everyone}\,(\lambda x.\textsc{thought}\,(\textsc{like}\,x\,\textsc{Kim})\,\textsc{Chris})$

We can check that:

$$\begin{aligned}
\mathscr{L}_{\mathrm{Syn}}(t_{\text{de dicto}}) &= \left[^{\mathrm{s}}\mathsf{Chris}\ \left[\mathsf{though}\ ((\lambda x.\left[^{\mathrm{s}}\mathsf{Kim}\ \left[\mathsf{liked}\ x^{\mathrm{c}}\right]\right])\,\mathsf{everyone})^{\mathrm{c}}\right]\right] \\
&\to_\beta \left[^{\mathrm{s}}\mathsf{Chris}\ \left[\mathsf{though}\ \left[^{\mathrm{s}}\mathsf{Kim}\ \left[\mathsf{liked\ everyone}^{\mathrm{c}}\right]\right]^{\mathrm{c}}\right]\right] \\
\mathscr{L}_{\mathrm{Syn}}(t_{\text{de re}}) &= (\lambda x.\left[^{\mathrm{s}}\mathsf{Chris}\ \left[\mathsf{thought}\ \left[^{\mathrm{s}}\mathsf{Kim}\ \left[\mathsf{like}\ x^{\mathrm{c}}\right]\right]^{\mathrm{c}}\right]\right])\,\mathsf{everyone} \\
&\to_\beta \left[^{\mathrm{s}}\mathsf{Chris}\ \left[\mathsf{though}\ \left[^{\mathrm{s}}\mathsf{Kim}\ \left[\mathsf{liked\ everyone}^{\mathrm{c}}\right]\right]^{\mathrm{c}}\right]\right]
\end{aligned}$$

while

$$\begin{aligned}
\mathscr{L}_{\mathrm{Log}}(t_{\text{de dicto}}) &= \mathsf{think'}(\mathsf{ev'}(\lambda x.\mathsf{like'}\,x\,\mathsf{K})\mathsf{C} \\
\mathscr{L}_{\mathrm{Log}}(t_{\text{de re}}) &= \mathsf{ev'}(\lambda x.\mathsf{think'}(\mathsf{like'}\,x\,\mathsf{K})\mathsf{C})
\end{aligned}$$

Table 7. ACG translation of the CVG lexicon for quantifier scope fragment

EVERYONE	$: \langle np, \iota_\pi^\pi\rangle$	SOMEONE	$: \langle np, \iota_\pi^\pi\rangle$
$\mathscr{L}_{\mathrm{Syn}}(\textsc{everyone})$	$= \mathsf{everyone} : np$	$\mathscr{L}_{\mathrm{Syn}}(\textsc{someone})$	$= \mathsf{someone} : np$
$\mathscr{L}_{\mathrm{Log}}(\textsc{everyone})$	$= \mathsf{ev'} : (\iota \multimap \pi) \multimap \pi$	$\mathscr{L}_{\mathrm{Log}}(\textsc{someone})$	$= \mathsf{so'} : (\iota \multimap \pi) \multimap \pi$
THOUGHT	$: \langle s, \pi\rangle \multimap \langle np, \iota\rangle \multimap \langle s, \pi\rangle$	KIM	$: \langle np, \iota\rangle$
$\mathscr{L}_{\mathrm{Syn}}(\textsc{thought})$	$= \lambda os.\left[^{\mathrm{s}}s\ \left[\mathsf{thought}\ o^{\mathrm{c}}\right]\right] : s \multimap np \multimap s$	$\mathscr{L}_{\mathrm{Syn}}(\textsc{Kim})$	$= \mathsf{Kim} : np$
$\mathscr{L}_{\mathrm{Log}}(\textsc{thought})$	$= \lambda os.\mathsf{think'}\,o\,s : \iota \multimap \iota \multimap \pi$	$\mathscr{L}_{\mathrm{Log}}(\textsc{Kim})$	$= \mathsf{K} : \iota$
$G^{\mathrm{S}}_{\langle np,\iota_\pi^\pi\rangle}$	$: \langle np, \iota_\pi^\pi\rangle \multimap (\langle np, \iota\rangle \multimap \langle s, \pi\rangle) \multimap \langle s, \pi\rangle$		
$\mathscr{L}_{\mathrm{Syn}}(G^{\mathrm{S}}_{\langle np,\iota_\pi^\pi\rangle})$	$= \lambda PQ.Q\,P$		
$\mathscr{L}_{\mathrm{Log}}(G^{\mathrm{S}}_{\langle np,\iota_\pi^\pi\rangle})$	$= \lambda PQ.P\,Q$		

5.2. Quantification

The previous section presents an example of scope ambiguity arising from where in the derivation the R rule occurs. We now consider scope ambiguity corresponding to different orders in retrieving the components from the store. Lexicon of Table 6 allows us to build the derivations (detailed in Example 1.1 and not repeated here) $\pi_{\text{Subj. wide scope}}$ and $\pi_{\text{Obj. wide scope}}$, which are isomorphic to $t_{\text{Subj. wide scope}}$ and $t_{\text{Obj. wide scope}}$ where:

- $t_{\text{Subj. wide scope}} = G^{\mathrm{S}}_{\langle \mathit{np}, \iota^{\pi}_{\pi} \rangle}\ \textsc{everyone}\ (\lambda y. G^{\mathrm{S}}_{\langle \mathit{np}, \iota^{\pi}_{\pi} \rangle}\ \textsc{someone}\ (\lambda x. \textsc{like'}\ x\ y))$
- $t_{\text{Obj. wide scope}} = G^{\mathrm{S}}_{\langle \mathit{np}, \iota^{\pi}_{\pi} \rangle}\ \textsc{someone}\ (\lambda x. G^{\mathrm{S}}_{\langle \mathit{np}, \iota^{\pi}_{\pi} \rangle}\ \textsc{everyone}\ (\lambda y. \textsc{like'}\ x\ y))$

We can check that:

$$\begin{aligned}
\mathscr{L}_{\mathrm{Syn}}(t_{\text{Subj. wide scope}}) &= (\lambda y.(\lambda x.\left[^{\mathrm{s}} y\ \left[\mathsf{liked}\ x\ ^{\mathrm{c}}\right]\right])\,\mathsf{someone})\,\mathsf{everyone} \\
&\rightarrow_{\beta} \left[^{\mathrm{s}}\mathsf{everyone}\ \left[\mathsf{liked}\ \mathsf{someone}\ ^{\mathrm{c}}\right]\right] \\
\mathscr{L}_{\mathrm{Syn}}(t_{\text{Obj. wide scope}}) &= (\lambda x.(\lambda y.\left[^{\mathrm{s}} y\ \left[\mathsf{liked}\ x\ ^{\mathrm{c}}\right]\right])\,\mathsf{everyone})\,\mathsf{someone} \\
&\rightarrow_{\beta} \left[^{\mathrm{s}}\mathsf{everyone}\ \left[\mathsf{liked}\ \mathsf{someone}\ ^{\mathrm{c}}\right]\right]
\end{aligned}$$

while

$$\begin{aligned}
\mathscr{L}_{\mathrm{Log}}(t_{\text{Subj. wide scope}}) &= \mathsf{ev'}(\lambda y.\mathsf{so'}(\lambda x.\mathsf{like'}\ x\ y)) \\
\mathscr{L}_{\mathrm{Log}}(t_{\text{Obj. wide scope}}) &= \mathsf{so'}(\lambda x.\mathsf{ev'}(\lambda y.\mathsf{like'}\ x\ y))
\end{aligned}$$

6. Comments and Perspectives

Since we now are in position of encoding CVG into ACG, we can look again at the general architecture we used (depicted in Fig. 1). In CVG, the semantic effect of in situ operators is achieved using an interface calculus. This calculus, the interface calculus, explicitly encodes the *relation* between syntactic terms and semantic terms. In particular, it means that whenever a single syntactic term u of a CVG G has two semantic interpretations v_1 and v_2, there are two interface derivations d_1 and d_2 proving $\vdash u, v_1 : \alpha, \beta \dashv$ and $\vdash u, v_2 : \alpha, \beta \dashv$.

In the ACG settings, this means there are two abstract terms D_1 and D_2 of $\Lambda(\Sigma_{I(G)})$ (corresponding to d_1 and d_2) such that:

- $\mathscr{L}_{\mathrm{Syn}}(D_1) = u = \mathscr{L}_{\mathrm{Syn}}(D_2)$
- $\mathscr{L}_{\mathrm{Log}}(D_1) = v_1$ and $\mathscr{L}_{\mathrm{Log}}(D_2) = v_2$

In other words, $\mathscr{L}_{\mathrm{Syn}}$ *is not an injection* and we encode the *relation between syntactic terms and semantic terms using composition of the inverse of a function and a function* (resp. $\mathscr{L}_{\mathrm{Syn}}$ and $\mathscr{L}_{\mathrm{Log}}$).

It makes explicit that CVG interface derivations are structures controlling the (simple) syntactic structures (and, of course, the semantic structures as well). And the fact that CVG derivations are expressed at the abstract level of $\mathscr{G}_{\mathrm{Syn}}$ exactly relates to the ACG key feature of providing direct access to parse structures[10] at the abstract level.

So the first outcome of the ACG encoding of CVG is to give the same mathematical status to the interface terms (set of λ-terms) as the one of the syntactic terms and the semantic terms. It is also to assign them the role of an abstract structure that controls the syntactic and the semantic ones. Let's call this more abstract (in the ACG sense) structure the *interface* structure, which is probably more neutral than for instance *syntactic* (vs. simple syntactic) structure, or *deep* structure.

[10]Here, *parse structure* has to be understood as the underlying, or abstract, structure that generates some object term, the latter being in this case a (simple) syntactic term. Exactly as a (simple) syntactic term is the underlying structure of some phonological (or string) term.

The second outcome is to show that such an interface structure is able to give an account of in situ operators as soon as the function mapping those interface structures to the (simple) syntactic ones is not injective. The encoding we propose achieve this using *higher-order types*. It is interesting to note that both type-logical grammars, LFG [9], and [2] (while hiding it the tower notation) use them at some place to get the same results.

Then, while CVG forbid λ-abstraction in its syntactic calculus (at least to model in situ operators), the interface calculus can be seen as providing it. It gives a general principle to provide an account of in situ operators that give rise to semantic ambiguity to any grammar whose parse structures avoid λ-abstraction, namely to any 2nd order ACG, *i.e.* any multiple context-free grammar. [25] proposes an account of quantification ambiguity along these lines for Tree Adjoining Grammars. It has to be contrasted with approaches where getting a semantic ambiguity for a single syntactic representation requires the use of an underspecified representation language. In those approaches, a single syntactic structure is mapped onto a single underspecified formula (a description) which is in turn solved in two (or more) logical formulas.

Finally, this encoding echoes the ACG perspective on CG of [26] pinpointing the fact that what is called a *syntactic* derivation in CG rather relates to what we call here an *interface* derivation and that CG somehow lacks what we call here the (simple) syntax structures.

A similar comparison could be worked out in the case of LFG and of [2]. In particular, the tower nation of the latter is very close to interface notation of CVG, except that the types driving the derivations is explicitly given as element of the tuple and that the underlying calculus is rather Combinatory Categorial Grammar [32]. In an ACG setting, the “Lift” and “Lower” rules of [2] could be explicitely modelled by abstract constants and interpreted by the different lexicons just as the CVG G rule is.

Conclusion

We have shown how to encode a linguistically motivated *parallel* formalism, CVG, into a framework, ACG, that up until now has mainly been used to encode syntactocentric formalisms. 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 relates to what is usually called *syntax* in mainstream categorial grammar, whereas the CVG *simple syntax* calculus is not expressed in such frameworks. This also suggests a general method to provide multiple context-free grammars with a mechanism to model semantic ambiguity using higher-order types rather than underspecified representation formalisms. In that respect, the encoding of these different formalisms in a common generic framework, here ACG, can help to illuminate differences between formalisms, but also to make them share the modelling of some linguistic phenomena related to the syntax-semantics interface.

References

[1] Bach, E., Partee, B. H.: Anaphora and semantic structure, 1980, Reprinted in Barbara H. Partee, Compositionality in Formal Semantics (Blackwell), pp. 122-152.

[2] Barker, C., Shan, C.-c.: Donkey Anaphora is In-Scope Binding, *Semantics and Pragmatics*, **1**(1), June 2008, 1–46.

[3] Blackburn, P., Bos, J.: *Representation and Inference for Natural Language. A First Course in Computational Semantics*, CSLI, 2005.

[4] Carpenter, B.: *Type-Logical Semantics*, The MIT Press, 1997.

[5] Cooper, R.: *Montague's Semantic Theory and Transformational Syntax*, Ph.D. Thesis, University of Massachusetts at Amherst, 1975.

[6] Cooper, R.: *Quantification and Syntactic Theory*, Reidel, Dordrecht, 1983.

[7] Culicover, P. W., Jackendoff, R.: *Simpler Syntax*, Oxford University Press, 2005.

[8] Curry, H.: Some logical aspects of grammatical structure, *Studies of Language and its Mathematical Aspects* (R. Jakobson, Ed.), Proc. of the 12th Symp. Appl. Math.., Providence, 1961.

[9] Dalrymple, M.: *Lexical Functional Grammar*, vol. 42 of *Syntax and Semantics series*, Academic Press, 2001.

[10] Gazdar, G.: Unbounded dependencies and coordinate structure, *Linguistic Inquiry*, **12**, 1981, 155–184.

[11] de Groote, P.: Towards Abstract Categorial Grammars, *Association for Computational Linguistics, 39th Annual Meeting and 10th Conference of the European Chapter, Proceedings of the Conference*, 2001.

[12] de Groote, P., Maarek, S.: Type-theoretic extensions of Abstract Categorial Grammars, *New Directions in Type-Theoretic Grammars, proceedings of the workshop*, 2007, `http://let.uvt.nl/general/people/rmuskens/ndttg/ndttg2007.pdf`.

[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] Hendriks, H.: *Studied Flexibility: Categories and Types in Syntax and Semantics*, Ph.D. Thesis, University of Amsterdam, 1993.

[15] Hinderer, S.: *Automatisation de la construction sémantique dans TYn*, Ph.D. Thesis, Université Henri Poincaré – Nancy 1, 2008, `http://tel.archives-ouvertes.fr/docs/00/35/30/65/PDF/these-hinderer.pdf`.

[16] Jackendoff, R.: *Foundations of Language: Brain, Meaning, Grammar, Evolution*, Oxford University Press, 2002.

[17] Kanazawa, M.: Second-Order Abstract Categorial Grammars as Hyperedge Replacement Grammars, *Journal of Logic, Language, and Information*, **19**(2), 2009, 137–161, `http://www.springerlink.com/content/p644605651088uv6/`.

[18] Kanazawa, M., Salvati, S.: Generating Control Languages with Abstract Categorial Grammars, *Proceedings of The 12th conference on Formal Grammar FG 2007* (G. Penn, Ed.), CSLI Publications, 2007, `http://www.dei.unipd.it/~fgrammar/fg07/fg07preproc/0005/paper_10.pdf`.

[19] Kaplan, R. M., Bresnan, J.: Lexical-Functional Grammar: a formal system for grammatical representation, in: *The Mental Representation of Grammatical Relations* (J. Bresnan, Ed.), MIT Press, 1982, 173–281.

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

[21] Montague, R.: The proper treatment of quantification in ordinary English, *Approaches to natural language: proceedings of the 1970 Stanford workshop on Grammar and Semantics* (J. Hintikka, J. Moravcsik, P. Suppes, Eds.), Reidel, Dordrecht, 1973.

[22] Moortgat, M.: Generalized quantifiers and discontinuous type constructors, in: *Discontinuous constituency* (W. Sijtsma, A. van Horck, Eds.), De Gruyter, 1991.

[23] Moortgat, M.: Categorial Type Logics, in: *Handbook of Logic and Language* (J. van Benthem, A. ter Meulen, Eds.), Elsevier Science Publishers, Amsterdam, 1996, 93–177.

[24] Oehrle, R. T.: Term-Labeled Categorial Type Systems, *Linguistic and Philosophy*, **17**(6), December 1994, 633–678.

[25] Pogodalla, S.: Ambiguïté de portée et approche fonctionnelle des TAG, *Actes de TALN'07*, 2007, `http://hal.inria.fr/inria-00141913`.

[26] Pogodalla, S.: Generalizing a proof-theoretic account of scope ambiguity, *Proceedings of the 7th International Workshop on Computational Semantics - IWCS-7* (J. Geertzen, E. Thijsse, H. Bunt, A. Schiffrin, Eds.), Tilburg University, Deparment of Communication and Information Sciences, 2007, `http://hal.inria.fr/inria-00112898`.

[27] Pollard, C.: The Calculus of Responsibility and Commitment, Submitted.

[28] Pollard, C.: Covert movement in logical grammar, Submitted.

[29] Pollard, C., Sag, I. A.: *Head-Driven Phrase Structure Grammar*, CSLI Publications, Stanford, CA, 1994, Distributed by University of Chicago Press.

[30] Salvati, S.: Encoding second order string ACG with Deterministic Tree Walking Transducers, *Proceedings of The 11th conference on Formal Grammar FG 2006* (Shuly Wintner, Ed.), FG Online Proceedings, CSLI Publications, Malaga Espagne, 2006, `http://cslipublications.stanford.edu/FG/2006/salvati.pdf`.

[31] Salvati, S.: On the complexity of Abstract Categorial Grammars, *Proceedings of the 10th Conference on Mathematics of Language, MOL 10*, 2007, `http://wwwhomes.uni-bielefeld.de/mkracht/mol10/abstracts/acg_complexity.pdf`.

[32] Steedman, M.: *Surface Structure and Interpretation*, MIT Press, 1996.