Controlling Extraction in Abstract Categorial Grammars

Sylvain Pogodalla[1] and Florent Pompigne[2]

[1] sylvain.pogodalla@inria.fr
LORIA/INRIA Nancy – Grand Est
[2] florent.pompigne@loria.fr
LORIA/Nancy Université

Abstract. This paper proposes an approach to control extraction in the framework of Abstract Categorial Grammar (ACG). As examples, we consider embedded wh-extraction, multiple wh-extraction and tensed-clauses as scope islands. The approach relies on an extended type system for ACG that introduces dependent types and advocates for a treatment at a rather abstract (tectogrammatical) level. Then we discuss approaches that put control at the object (phenogrammatical) level, using appropriate calculi.

1 Introduction

In pursuing [1]'s program of separating the combinatorial part of grammars, the *tectogrammatical* level, from the one that realizes the operations on the surface structures, the *phenogrammatical* level, the two independently formulated frameworks of Lambda Grammar (LG) [2, 3] and Abstract Categorial Grammar (ACG) [4] propose to consider the implicative fragment of linear logic as the underlying tectogrammatical calculus. While interleaving the phenogrammatical and the tectogrammatical levels as in standard Categorial Grammar and Lambek calculus (CG) [5, 6] leads to using a *directed* (or non-commutative) calculus, both LG and ACG rather rely on a *non-directed* (or commutative) calculus.

As immediate outcome of this choice, extraction is easily available, in particular from medial position whereas CG permits only for peripheral extraction. So even if CG and Lambek grammars are known for their powerful treatment of extraction, LG and ACG extend these capabilities.

However, it is a common observation that extractions are not completely free in natural language in general. The power of hypothetical reasoning of Lambek calculus based grammars itself is sometimes too strong [7, p. 207]. Directionality of the calculus is not sufficient to model all kinds of *islands* to extraction, for instance with coordinate structures, and it *overgenerates*. Because of the presence of hypothetical reasoning in the LG and ACG frameworks, the question arises whether those frameworks overgenerate as well and, because they do, how to control extraction in those frameworks.

This paper aims at providing some solution to control extraction in the framework of ACG for various cases, including tensed-clauses as scope islands, embedded wh-extraction and multiple wh-extraction. The solution relies on an extended type system for ACG that Sect. 2 presents together with the ACG basics. We emphasize there the compositional[3] flexibility of ACG and present a treatment at a rather abstract (tectogrammatical) level. Then Sect. 3 describes the examples and the solutions we provide. Our account focuses on using *dependent types*, both in a rather limited and in a more general setting. Section 4 compares our approach with related works. We first discuss approaches that put control at the phenogrammatical level, using appropriate calculi, then discuss other ACG models that use the same kind of architectures as the one we propose. We also discuss ways of importing solutions developed in the the CG frameworks.

2 ACG: Definitions and Properties

The ACG formalism lies within the scope of type-theoretic grammars [5, 1, 8, 9]. In addition to relying on a small set of mathematical primitives from type-theory and λ-calculus, an important property concerns the direct control it provides over the parse structures of the grammar. This control is at the heart of the present proposal.

2.1 Definitions

The definitions we provide here follow [4] together with the type-theoretic extension of [10, 11] providing the dependent product[4].

Definition 1. *The set of kinds $\mathscr{K}$, the set of types $\mathscr{T}$ and the set of terms T are defined as:*

$$\begin{aligned} \mathscr{K} &::= \texttt{type} \,|\, (\mathscr{T})\mathscr{K} \\ \mathscr{T} &::= a \,|\, (\lambda x.\mathscr{T}) \,|\, (\mathscr{T}\,T) | (\mathscr{T} \multimap \mathscr{T}) \,|\, (\Pi\, x : \mathscr{T})\mathscr{T} \\ T &::= c \,|\, x \,|\, (\lambda^0 x.T) \,|\, (\lambda x.T) \,|\, (T\,T) \end{aligned}$$

where a ranges over atomic types and c over constants [5].

Assume for instance a type *Gender* and the three terms masc, fem and neut of this type. We then can define *np* with kind $(\textit{Gender})\texttt{type}$ that derives three types: *np* masc, *np* fem and *np* neut. *np* can be seen as a feature structure whose gender value is still missing while *John* can be seen as a term of type *np* masc, *i.e.* a feature structure where the value of the *Gender* feature has been set to masc. On the other hand, an intransitive verb accepts as subject a noun phrase with any gender. So its type is typically $(\Pi x : \textit{Gender})\,(\textit{np}\,x \multimap s)$.

Definition 2 (Signature). *A* raw signature *is a sequence of declarations of the form 'a : K'or of the form 'c : α', where a ranges over atomic types, c over constants, K over kinds and α over types.*

Let Σ be a raw signature. We write A_Σ (resp. C_Σ) for the set of atomic types (resp. constants) declared in Σ and write $\mathscr{K}_\Sigma$ (resp. $\mathscr{T}_\Sigma$ and Λ_Σ) for the set of well-formed

[3] As in functional composition, not as in the compositionality principle.

[4] We don't use the record and variant types they introduced.

[5] $\lambda^0 x.T$ denotes the linear abstraction and $\lambda x.T$ the non-linear one. $(\Pi x : \alpha)$ denotes a universal quantification over variables of type α.

kinds (resp. well-kinded types and well-typed terms). In case Σ correctly introduces well-formed kinds and well-kinded types, it is said to be a well-formed signature.

We also define κ_Σ (resp. τ_Σ) the function that assigns kinds to atomic types (resp. that assigns types to constants).

There is no room here to give the typing rules detailed in [10, 11], but the ones used in the next sections are quite straightforward. They all are instances of the following derivation (the sequent $\vdash_\Sigma (\textsc{sleeps}\,\mathsf{masc})\textsc{John} : s$ is said to be *derivable*) assuming the raw signature Σ of Table 1:

$$\dfrac{\dfrac{\dfrac{}{\vdash_\Sigma \textsc{sleeps} : (\Pi x : \mathit{Gender})\,(\mathit{np}\,x \multimap s)} \qquad \dfrac{}{\vdash_\Sigma \mathsf{masc} : \mathit{Gender}}}{\vdash_\Sigma \textsc{sleeps}\,\mathsf{masc} : \mathit{np}\,\mathsf{masc} \multimap s} \qquad \dfrac{}{\vdash_\Sigma \textsc{John} : \mathit{np}\,\mathsf{masc}}}{\vdash_\Sigma (\textsc{sleeps}\,\mathsf{masc})\textsc{John} : s}$$

$\Sigma :$	$\mathit{Gender} : \mathtt{type}$	$\mathsf{masc}, \mathsf{fem} : \mathit{Gender}$	$\textsc{John} : \mathit{np}\,\mathsf{masc}$
	$\mathit{np} \quad : (\mathit{Gender})\mathtt{type}$	$\textsc{sleeps} \quad : (\Pi x : \mathit{Gender})\,(\mathit{np}\,x \multimap s)$	

Table 1. Raw signature example

Definition 3 (Lexicon). *A* lexicon *from Σ_A to Σ_O is a pair $\langle \eta, \theta \rangle$ where:*

- *η is a morphism form A_{Σ_A} to $\mathscr{T}_{\Sigma_O}$ (we also note η its unique extension to $\mathscr{T}_{\Sigma_A}$);*
- *θ is a morphism form C_{Σ_A} to Λ_{Σ_O} (we also note θ its unique extension to Λ_{Σ_A});*
- *for every $c \in C_{\Sigma_A}$, $\theta(c)$ is of type $\eta(\tau_{\Sigma_A}(c))$;*
- *for every $a \in A_{\Sigma_A}$, the kind of $\eta(a)$ is $\tilde{\eta}(\kappa_{\Sigma_A}(a))$ where $\tilde{\eta} : \mathscr{K}_{\Sigma_A} \to \mathscr{K}_{\Sigma_O}$ is defined by $\tilde{\eta}(\mathtt{type}) = \mathtt{type}$ and $\tilde{\eta}((\alpha)K) = (\eta(\alpha))\tilde{\eta}(K)$.*

Definition 4 (ACG). *An* abstract categorial grammar *is a quadruple $\mathscr{G} = \langle \Sigma_A, \Sigma_O, \mathscr{L}, s \rangle$ where:*

1. *Σ_A and Σ_O are two well-formed signatures: the* abstract vocabulary *and the* object vocabulary*, respectively;*
2. *$\mathscr{L} : \Sigma_A \to \Sigma_O$ is a lexicon from the abstract vocabulary to the object vocabulary;*
3. *$s \in \mathscr{T}_{\Sigma_A}$ (in the abstract vocabulary) is the* distinguished type *of the grammar.*

While the object vocabulary specifies the surface structures of the grammars (*e.g.* strings or trees), the abstract vocabulary specifies the parse structures (*e.g.* trees, but more generally proof trees as in CG). The lexicon specifies how to map the parse structures to the surface structures.

Definition 5 (Languages). *An ACG $\mathscr{G} = \langle \Sigma_A, \Sigma_O, \mathscr{L}, s \rangle$ defines two languages:*

- *the* abstract language*: $\mathcal{A}(\mathscr{G}) = \{t \in \Lambda_{\Sigma_A} \mid \vdash_{\Sigma_A} t : s \text{ is derivable}\}$*
- *the* object language*, which is the image of the abstract language by the lexicon: $\mathcal{O}(\mathscr{G}) = \{t \in \Lambda_{\Sigma_O} \mid \exists u \in \mathcal{A}(\mathscr{G}).\, t = \mathscr{L}(u)\}$*

The expressive power and the complexity of ACG have been intensively studied, in particular for 2nd-order ACG. This class of ACG corresponds to a subclass of the ACG where linear implication ($\multimap$) is the unique type constructor (*core* ACG). While the parsing problem for the latter reduces to provability in the Multiplicative Exponential fragment of Linear Logic (MELL) [12], which is still unknown, parsing of 2nd-order ACG is polynomial and the generated languages correspond to mildly context-sensitive languages [13, 12, 14][6].

Extending the typing system with dependent products results in a Turing-complete formalism. The problem of identifying interesting and tractable fragments for this extended type system is ongoing work that we don't address in this paper. However, a signature where types only depend on finitely inhabited types (as in the former example, *np* depends on the finitely inhabited type *Gender*) can be expressed in core ACG and complexity results can be transfered. The model we propose in Sect. 3.3 has this property. For the other cases where the number of inhabitants is infinite, an actual implementation could take into account an upper bound for the number of extractions in the same spirit as [15, 16] relate the processing load with the number of unresolved dependencies while processing a sentence, and could reduce these cases to the finite one.

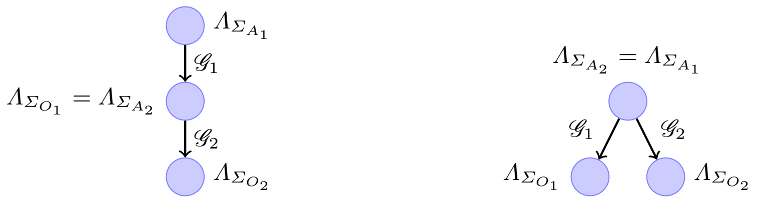
2.2 Grammatical Architecture

Since they both are higher-order signatures, the abstract vocabulary and the object one don't show any structural difference. This property makes ACG composition a quite natural operation. Figure 1(a) exemplifies the first way to compose two ACG: the object vocabulary of the first ACG $\mathscr{G}_1$ is the abstract vocabulary of the second ACG $\mathscr{G}_2$. Its objectives are twofold:

- either a term $u \in \mathcal{A}(\mathscr{G}_2)$ has at least one antecedent by the lexicon of $\mathscr{G}_1$ in $\mathcal{A}(\mathscr{G}_1)$ (or even two or more antecedents) and $\mathscr{G}_2 \circ \mathscr{G}_1$ provides more analysis to a same object term of $\mathcal{O}(\mathscr{G}_2)$ than $\mathscr{G}_2$ does. [17, 18] use this architecture to model scope ambiguity using higher-order types for quantified noun phrases at the level of Σ_{A_1} while their type remains low at the level of Σ_{A_2};
- or a term $u \in \mathcal{A}(\mathscr{G}_2)$ has no antecedent by the lexicon of $\mathscr{G}_1$ in $\mathcal{A}(\mathscr{G}_1)$. It means that $\mathscr{G}_2 \circ \mathscr{G}_1$ somehow *discards* some analysis given by $\mathscr{G}_2$ of an object term of Λ_{O_2}. We have chosen this architecture in this paper for that purpose: while some constructs are accepted by $\mathscr{G}_{\mathrm{Syn}}$ (to be defined in Sect. 3.1), an additional control at a more abstract level discard them.

Figure 1(b) illustrates the second way to compose two ACG: $\mathscr{G}_1$ and $\mathscr{G}_2$ share the same abstract vocabulary, hence define the same abstract language. This architecture arises in particular when one of the ACG specifies the syntactic structures and the other one specifies the semantic structures. The shared abstract vocabulary hence specifies the syntax-semantics interface. [19, 18] precisely consider this architecture with that aim. Note that this architecture for the syntax-semantics interface corresponds to the presentation of synchronous TAG as a bi-morphic architecture [20].

[6] There are other decidable classes we don't discuss here.



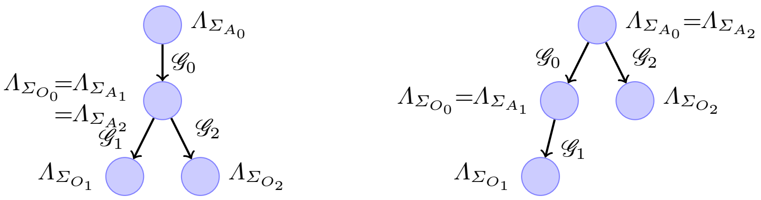
(a) First composition mode (b) Second composition mode

Fig. 1. Various ways of composing ACG

Finally, mixing the two ways of composition is also possible, as Fig. 2 illustrates. Because the ACG for the semantics is linked at the highest level in Fig. 2(b), this architecture has been used in [17] and [18] to model semantic ambiguity while keeping at an intermediate level a non-ambiguous syntactic type for quantifiers. Indeed the semantics needs in that case to attach to the place where ambiguity already arised.

On the other hand, if the syntax-semantics interface takes place at an intermediate level such as in Fig. 2(a) the highest ACG can provide further control on the acceptable structures: while some syntactic constructs could be easily given a semantics, it might happen that they're forbidden in some languages. Hence the need of another control that discards those constructs. This paper uses such an architecture and we show first how to set a fairly standard syntax-semantics interface and second how to provide additional control without changing anything to this interface.

Note that in both cases, because the composition of two ACG is itself an ACG, these architectures boil down to the one of Fig. 1(b). However, keeping a multi-level architecture helps in providing some modularity for grammatical engineering, either by reusing components as in Fig. 2(a) (where the syntax-semantics interface is not affected by the supplementary control provided by the most abstract ACG) or by providing intermediate components as in Fig. 2(b) (such as the low-order type for quantifiers, contrary to CG)[7].



(a) First combination (b) Second combination

Fig. 2. Mixing composition modes

[7] However, for sake of simplicity, we don't use this intermediate level here and directly adopt the standard higher-order type for quantified noun-phrases.

3 Examples

3.1 The Syntax-Semantics Interface

Following the architecture presented in Sect. 2.2, we first briefly define the two ACG sharing the same abstract language defining the general syntax-semantics interface we use. Since the scope of this paper is rather the control of this interface, we don't enter the details here. It's enough to say that we basically follow standard categorial grammar approaches except that the linear non-directional implication replaces the two directional implications[8]. We define $\mathscr{G}_{\mathrm{Syn}} = \langle \varSigma_{\mathrm{Syn}}, \varSigma_{\mathrm{String}}, \mathscr{L}_{\mathrm{Syn}}, s\rangle$ the ACG that relates syntactic structures together with their surface realization. Table 2 presents $\varSigma_{\mathrm{Syn}}$ the signature for the parse structures, $\varSigma_{\mathrm{String}}$ the signature for surface realization, and $\mathscr{L}_{\mathrm{Syn}}$ the lexicon that relates them.

$$
\begin{array}{lllllll}
\varSigma_{\mathrm{Syn}} : & & & & & & \\
 & s, np, n & : \texttt{type} & C_{\mathrm{so}}, C_{\mathrm{ev}} & : (np \multimap s) \multimap s & C_{\mathrm{loves}} & : np \multimap np \multimap s \\
 & C_{\mathrm{Mary}}, C_{\mathrm{John}} & : np & C_{\mathrm{who}} & : (np \multimap s) \multimap n \multimap n & C_{\mathrm{says}} & : s \multimap np \multimap s
\end{array}
$$

$$
\begin{array}{lll}
\varSigma_{\mathrm{String}} : & & \\
 & \sigma & : \texttt{type} \\
 & /\mathrm{Mary}/, /\mathrm{John}/, /\mathrm{someone}/, \epsilon, /\mathrm{everyone}//\mathrm{loves}/, /\mathrm{who}/, /\mathrm{says}/ & : \sigma \\
 & + & : \sigma \multimap \sigma \multimap \sigma
\end{array}
$$

$$
\begin{array}{lllll}
\mathscr{L}_{\mathrm{Syn}} : & & & & \\
 & s, np, n & :=_{\mathrm{Syn}} \sigma & C_{\mathrm{Mary}} & :=_{\mathrm{Syn}} /\mathrm{Mary}/ \\
 & C_{\mathrm{so}} & :=_{\mathrm{Syn}} \lambda^0 p.p\,/\mathrm{someone}/ & C_{\mathrm{loves}} & :=_{\mathrm{Syn}} \lambda^0 os.s + /\mathrm{loves}/ + o \\
 & C_{\mathrm{who}} & :=_{\mathrm{Syn}} \lambda^0 pn.n + /\mathrm{who}/ + (p\,\epsilon) & C_{\mathrm{says}} & :=_{\mathrm{Syn}} \lambda^0 cs.s + /\mathrm{says}/ + c
\end{array}
$$

Table 2. $\varSigma_{\mathrm{Syn}}$, $\varSigma_{\mathrm{String}}$ (σ stands for the type of string, $+$ for the concatenation operation and ϵ for the empty string) and $\mathscr{L}_{\mathrm{Syn}}$ (obvious interpretations are omitted)

In situ operators such as quantifiers have the property to (semantically) take scope over complex (surface) expressions they are part of. In (1) for instance, the quantified noun phrase (QNP), while subpart of the whole sentence, has the existential quantifier of its semantic contribution taking scope over the whole proposition as in (1-a).

(1) Mary loves someone
 a. $\exists x.\mathbf{love}\,\mathbf{m}\,x$
 b. $C_{\mathrm{so}}(\lambda^0 x.C_{\mathrm{loves}}\,x\,C_{\mathrm{Mary}})$

The way CG model these phenomena is to type QNP with the higher-order type $(np \multimap s) \multimap s$, whose first argument is a sentence missing an NP. Such an argument can be represented by a λ-term starting with an abstraction $\lambda^0 x.t$ with x occurring (free) in t that plays the role of any non quantified NP having the surface position of the QNP. So, in the previous example, t would represent the expression *Mary loves* x, and

[8] ACG manages word order at the surface level. For discussion on relations between ACG and CG, see [21].

the representation of (1) is (1-b). We leave it to the reader to check that the string representation is indeed the image by $\mathscr{L}_{\text{Syn}}$ of (1-b).

The case of wh-words where the movement is overt is dealt with in almost the same way: the first argument is a sentence missing an NP. The difference (overt *vs.* covert) rests in what is provided to this first argument to get the surface form: in the case of covert movements, there is an actual realization with the QNP form (see $\mathscr{L}_{\text{Syn}}(C_{\text{so}})$) while there is no realization of overt movements (see $\mathscr{L}_{\text{Syn}}(C_{\text{who}})$). However, in both cases, the abstract structure contains a variable that is abstracted over. In the sequel of this paper, we refer to the variable as the *extracted* variable, or as the variable *available for extraction*.

We also define $\mathscr{G}_{\text{Sem}} = \langle \Sigma_{\text{Syn}}, \Sigma_{\text{Sem}}, \mathscr{L}_{\text{Sem}}, s\rangle$ the ACG that relates syntactic structures together with their *semantic* interpretation. As expected, $\mathscr{G}_{\text{Syn}}$ and $\mathscr{G}_{\text{Sem}}$ share the abstract vocabulary Σ_{Syn} presented in Table 2. Table 3 presents Σ_{Sem} the signature for logical formulas and $\mathscr{L}_{\text{Sem}}$ the lexicon that relates them. This lexicon associates (1-b) with its meaning (1-a).

Σ_{Sem} :						
	e,t	: `type`	$\mathbf{m},\mathbf{j}$	: e	$\forall,\exists$	: $(e\rightarrow t)\multimap t$
	$\wedge,\Rightarrow$	: $t\multimap t\multimap t$	**love**	: $e\multimap e\multimap t$	**say**	: $t\multimap e\multimap t$
$\mathscr{L}_{\text{Sem}}$:						
	s	$:=_{\text{Sem}} t$	np	$:=_{\text{Sem}} e$		
	n	$:=_{\text{Sem}} e\multimap t$	C_{Mary}	$:=_{\text{Sem}} \mathbf{m}$		
	C_{so}	$:=_{\text{Sem}} \lambda^0 p.\forall x.p\,x$	C_{loves}	$:=_{\text{Sem}} \lambda^0 os.s(\lambda^0 x.o(\lambda^0 y.\mathbf{love}\,x\,y))$		
	C_{who}	$:=_{\text{Sem}} \lambda^0 pn.\lambda x.(n\,x)\wedge(p\,x)$	C_{says}	$:=_{\text{Sem}} \lambda^0 cs.\mathbf{say}\,x\,c$		

Table 3. Σ_{Sem} and $\mathscr{L}_{\text{Sem}}$

Because $\mathscr{G}_{\text{Syn}}$ is a straightforward adaptation of standard treatments of quantification and relativization in CG, it overgenerates as well. Indeed, when building a term using free variables, those variables can be arbitrarily deep in the term, and can be abstracted over in any order (resulting in particular in scope ambiguity), as close of the top level as we want. However, natural languages are not completely free with that respect, and the next sections are devoted to deal with some of these cases and to show how to introduce some control.

The principle we adopt is based on the following observation: operators triggering extractions get the general pattern $(\alpha \multimap \beta) \multimap \gamma$ for their type. However, not all elements of a same type α can be extracted. For instance, if α is *np*, it is required sometimes to be nominative and sometimes to be accusative. These constraints can be accomodated adding feature structures (here dependent types) to the syntactic type.

But this is not enough since β might also express some additional constaints. For instance, if β is *s*, extraction is sometimes possible under the assumption that no other extraction occured. This can also be expressed using feature structures added to *s*.

Finally, it might happen that not all combinations for the constraints on α and β are possible, meaning that the extraction constraints are described by a *relation*, distinct from the cartesian product, between their feature structures. For instance extraction of the subject inside a clause is possible provided this is the very subject of that clause. Dependent types allows us to implement such relations. This approach shares a lot of similarities with [22]s' usage of first order linar logic where first order variables also implements some kind of relation between constituents.

3.2 Tensed Clauses as Scope Islands for Quantifiers

(2) is a first example of such a constraint. It is indeed sometimes considered that in such sentences, the QNP *everyone* should not be able to take scope over *someone*, or even *says* as in (2-b) and (2-c): the QNP *everyone* cannot take its scope outside its minimal tensed sentence[9].

(2) Someone said everyone loves Mary
  a. $C_{\text{so}}(\lambda^0 x.C_{\text{says}}\,(C_{\text{ev}}(\lambda^0 y.C_{\text{loves}}\,C_{\text{Mary}}\,y))\,x)$
  $\exists x.\mathbf{say}\,x\,(\forall y.\mathbf{love}\,y\,\mathbf{m})$
  b. $*C_{\text{so}}(\lambda^0 x.C_{\text{ev}}(\lambda^0 y.C_{\text{says}}\,(C_{\text{loves}}\,C_{\text{Mary}}\,y)\,x))$
  $*\exists x.\forall y.\mathbf{say}\,x\,(\mathbf{love}\,y\,\mathbf{m})$
  c. $*C_{\text{ev}}(\lambda^0 y.C_{\text{so}}(\lambda^0 x.C_{\text{says}}\,(C_{\text{loves}}\,C_{\text{Mary}}\,y)\,x))$
  $*\forall y.\exists x.\mathbf{say}\,x\,(\mathbf{love}\,y\,\mathbf{m})$

The fact that a QNP cannot take its scope outside its minimal tensed sentence means that whenever such a sentence is argument of a verb like *says*, it should not contain any free variable, hence any variable available for extraction, anymore. To model that, we decorate the *s* and *np* types with an integer feature that contains the actual number of free variables of type *np* occurring in it. Because any *np* introduced by the lexicon is decorated by 0, *np* with a feature strictly greater than 0 can only be introduced by hypothetical reasoning, hence by free variables. A clause without any left free variable is then of type *s* decorated with 0: this is required for the first argument of the verb *says* for instance.

In order to avoid changing the syntax-semantics interface we defined in Sect. 3.1, we implement the control using a more abstract level. This level introduces the counter feature using a new signature Σ_{Cont_1}, as Table 4 shows. The new types are very similar to the ones of Σ_{Syn} (Table 2) except that they now depend on an integer meant to denote the number of free variables occurring in the subterms. We then define $\mathscr{G}_{\text{Cont}_1} = \langle \Sigma_{\text{Cont}_1}, \Sigma_{\text{Syn}}, \mathscr{L}_{\text{Cont}_1}, s\,0\rangle$ the ACG that realizes the control over the syntactic structures. $\mathscr{L}_{\text{Cont}_1}$ (Table 4) basically removes the dependent product and transforms Σ_{Cont_1} into Σ_{Syn}.

Having constants producing terms of type $s\,i$ like D_{loves}, where the feature indicates the number of current free variables that can be abstracted over in the subterms they are the head of, we are now in position of controlling the scope of QNP. Because the sentence argument of D_{says} is required to carry 0 free variables, all the quantified variables must have met their scope-taking operator before the resulting term is passed as argument, preventing them from escaping the scope island.

[9] This is arguable, and the tensed clauses island may be less straightforward, but this point is not ours here.

Σ_{Cont_1} :				
	int	$: \texttt{type}$	s, np, n	$: (\mathit{int})\,\texttt{type}$
	next	$: \mathit{int} \multimap \mathit{int}$	D_{loves}	$: (\Pi i, j : \mathit{int})\ (\mathit{np}\, i \multimap \mathit{np}\, j \multimap s\,(i+j))$
	$+$	$: \mathit{int} \multimap \mathit{int} \multimap \mathit{int}$	$D_{\text{so}}, D_{\text{ev}}$	$: (\Pi i : \mathit{int})\ ((\mathit{np}\, 1 \multimap s\,(\mathsf{next}\, i)) \multimap s\, i)$
	$D_{\text{John}}, D_{\text{Mary}}$	$: \mathit{np}\, 0$	D_{says}	$: (\Pi i : \mathit{int})\ (s\, 0 \multimap \mathit{np}\, i \multimap s\, i)$
$\mathscr{L}_{\mathrm{Cont}_1}$:				
	$s :=_{\mathrm{Cont}_1} \lambda x.\, s$		$\mathit{np}\ :=_{\mathrm{Cont}_1} \lambda x.\, \mathit{np}$	
	$n :=_{\mathrm{Cont}_1} \lambda x.\, n$		$D_{\text{x}} :=_{\mathrm{Cont}_1} C_{\text{x}}$	

Table 4. Σ_{Cont_1} and $\mathscr{L}_{\mathrm{Cont}_1}$

(3) is a well-typed term (of type $s\,0$) of $\Lambda_{\Sigma_{\mathrm{Cont}_1}}$. It has the same structure as (2-a) which, indeed, is its image by $\mathscr{L}_{\mathrm{Cont}_1}$. On the other hand, the type $\mathit{np}\,0 \multimap s\,0$ of (4) (that would be the counterpart of the subterm of (2-c)) prevents it from being argument of a quantifier. Here, D_{says} requires y to be of type $\mathit{np}\,0$ in order to have its argument $D_{\text{love}}\,0\,0\,D_{\text{Mary}}\,y$ of type $s\,0$.

$$(3)\quad D_{\text{so}}\,0\ \underbrace{(\lambda^0 x.\, D_{\text{says}}\,1\,\underbrace{(D_{\text{ev}}\,0\,\underbrace{(\lambda^0 y.\, \overbrace{D_{\text{love}}\,0\,1\,D_{\text{Mary}}\,\overbrace{y}^{\mathit{np}\,1}}^{s\,1})}_{\mathit{np}\,1\multimap s\,1})}_{s\,0}\ \overbrace{x}^{\mathit{np}\,1})}_{\mathit{np}\,1\multimap s\,1}$$

$$(4)\quad \lambda^0 y.\ \underbrace{D_{\text{so}}\,0\,\underbrace{(\lambda^0 x.\, D_{\text{says}}\,1\,\overbrace{(D_{\text{love}}\,0\,0\,D_{\text{Mary}}\,\overbrace{y}^{\mathit{np}\,0})}^{s\,0}\ \overbrace{x}^{\mathit{np}\,1})}_{\mathit{np}\,1\multimap s\,1}}_{s\,0}$$

This example could be easily adapted to other tensed clauses, as if-clauses or relative clauses. The next examples use the same principle: all types depend on a feature that expresses whether some free variables in the subterms are available for extraction. Then, wh-words put the condition on how many of them are simultaneously possible for extraction to take place while islands still require this number to be set to 0.

Note that in each case, we introduce a new feature for the particular phenomenon under study. Using record types (that *np*, *n* and *s* would depend on) with a proper field for each of them makes the different solutions work together without any interaction. Feature structures for each type might of course become complex, however this complexity can be dealt with in a very modular way.

3.3 Rooted and Embedded Wh-Extraction

We now focus on extractions in relative clauses, in which a distinction should be made between rooted extractions and embedded extractions: while an embedded object can be extracted by a relative pronoun, embedded subjects cannot. Only main-clause subjects (rooted subjects) can be extracted. This is illustrated in:

(5) *The man who$_1$ John said that t$_1$ loves Mary sleeps
$*C_{\text{sleep}}\,(C_{\text{the}}\,(C_{\text{who}}(\lambda^0 x.C_{\text{say that}}\,(C_{\text{love}}\,C_{\text{Mary}}\,x)\,C_{\text{John}})\,C_{\text{man}}))$

(6) The man whom$_1$ John said that Mary loves t$_1$ sleeps
$C_{\text{sleep}}\,(C_{\text{the}}\,(C_{\text{whom}}(\lambda^0 x.C_{\text{say that}}\,(C_{\text{love}}\,x\,C_{\text{Mary}})\,C_{\text{John}})\,C_{\text{man}}))$

Relative clauses are extraction islands, so we know that acceptable terms should never have more than one free variable available to extraction in the same clause. Hence we don't need an unbound counter for them and we use instead a 3-valued type that distinguishes: the absence of extraction, the existence of a rooted extraction, and the existence of an embedded one. The new abstract signature is given in Table 5 (for the sake of clarity, *who* will only refer to subject extraction and case is omitted). The corresponding ACG $\mathscr{G}_{\mathrm{Cont}_2} = \langle \Sigma_{\mathrm{Cont}_2}, \Sigma_{\mathrm{Syn}}, \mathscr{L}_{\mathrm{Cont}_2}, s\,\mathsf{no}\rangle$ is built in the same way as the previous example.

$$
\begin{array}{llll}
\multicolumn{4}{l}{\Sigma_{\mathrm{Cont}_2} :} \\
\quad \mathit{value}, \mathit{extraction} & : \texttt{type} & D_{\text{sleeps}} & : (\Pi x : \mathit{value})\,(\mathit{np}\,x \multimap s\,(f\,x\,\mathsf{cst})) \\
\quad \mathsf{var}, \mathsf{cst} & : \mathit{value} & D_{\text{loves}} & : (\Pi x, y : \mathit{value})\,(\mathit{np}\,x \multimap \mathit{np}\,y \multimap s\,(f\,x\,y)) \\
\quad \mathsf{no}, \mathsf{root}, \mathsf{emb} & : \mathit{extraction} & D_{\text{the}} & : (\Pi x : \mathit{value})\,(n\,x \multimap \mathit{np}\,x) \\
\quad \mathit{np}, n & : (\mathit{value})\,\texttt{type} & D_{\text{says that}} & : (\Pi x : \mathit{extraction}, y : \mathit{value})\,(s\,x) \\
\quad s & : (\mathit{extraction})\,\texttt{type} & & \qquad\qquad \multimap \mathit{np}\,y \multimap s\,(g\,x\,y) \\
\quad D_{\text{man}} & : n\,\mathsf{cst} & D_{\text{who}} & : (\mathit{np}\,\mathsf{var} \multimap s\,\mathsf{root}) \multimap n\,\mathsf{cst} \multimap n\,\mathsf{cst} \\
\quad D_{\text{John}}, D_{\text{Mary}} & : \mathit{np}\,\mathsf{cst} & D_{\text{whom}} & : (\mathit{np}\,\mathsf{var} \multimap s\,\mathsf{root}) \multimap n\,\mathsf{cst} \multimap n\,\mathsf{cst} \\
 & & D'_{\text{whom}} & : (\mathit{np}\,\mathsf{var} \multimap s\,\mathsf{emb}) \multimap n\,\mathsf{cst} \multimap n\,\mathsf{cst}
\end{array}
$$

$$
\text{with } f : \begin{cases} \mathsf{var}\ x \longrightarrow \mathsf{root} \\ \mathsf{cst}\ \mathsf{var} \longrightarrow \mathsf{root} \\ \mathsf{cst}\ \mathsf{cst} \longrightarrow \mathsf{no} \end{cases}
\qquad
g : \begin{cases} \mathsf{no}\ \ \mathsf{var} \longrightarrow \mathsf{root} & \mathsf{root}\ \mathsf{cst} \longrightarrow \mathsf{emb} \\ \mathsf{no}\ \ \mathsf{cst} \longrightarrow \mathsf{no} & \mathsf{emb}\ \mathsf{var} \longrightarrow \mathsf{emb} \\ \mathsf{root}\ \mathsf{var} \longrightarrow \mathsf{root} & \mathsf{emb}\ \mathsf{cst} \longrightarrow \mathsf{emb} \end{cases}
$$

R

Table 5. Σ_{Cont_2}

The behavior of a transitive verb such as D_{loves} is to percolate the information that a free variable occurs in its parameters. So the resulting type depends on no only when both the subject and the object don't themselves depend on a var term. Function f in Table 5 implements it.

Verbs requiring subordinate clauses as $D_{\text{says that}}$ also needs to percolate the information as to whether a free variable occurs in the main clause and/or if a free variable occurs in the subordinate clause (in that case, the extraction is embedded). Function g in Table 5 implements these conditions.

Finally, relative pronouns need to check the type of their argument. In particular subject extractor can't accept an argument clause with type $(\mathit{np}\,\mathsf{var} \multimap s\,\mathsf{emb})$ while other pronouns can. This prevents extractions of embedded subject from being generated while extraction of embedded objects can, as shown with the abstract term (7) of type $s\,\mathsf{no}$ associated to (6).

(7) $$D_{\text{sleep}}\,\mathsf{cst}\,D_{\text{the}}\,\mathsf{cst}\,(D'_{\text{whom}}\,\underbrace{(\lambda^0 x.\,\overbrace{D_{\text{says that}}\,\mathsf{root\,cst}\,\underbrace{(D_{\text{love}}\,\mathsf{var\,cst}\,\overbrace{x}^{np\,\mathsf{var}}\,D_{\text{Mary}})}_{s\,\mathsf{root}}\,D_{\text{John}})}^{s\,\mathsf{emb}}}_{np\,\mathsf{var}\multimap s\,\mathsf{emb}}\,D_{\text{man}})$$

On the other hand, $D_{\text{says that}}\,(D_{\text{love}}\,D_{\text{Mary}}\,x)\,D_{\text{John}}$ is typable only with type $s\,\mathsf{emb}$ or $s\,\mathsf{no}$ (because D_{John} is of type $np\,\mathsf{cst}$), hence $\lambda^0 x.D_{\text{says that}}\,(D_{\text{love}}\,x\,D_{\text{Mary}})\,D_{\text{John}}$ cannot be of type $np\,\mathsf{var} \multimap s\,\mathsf{root}$ and cannot be an argument of D_{who}. Then (5) cannot get an antecedent by $\mathscr{L}_{\text{Cont}_2}$[10].

The same technique can be used to model the fact that a nominative interrogative pronoun can form a root question with a sentence that is missing its main clause subject as in (8) but not with one that is missing an embedded subject as in (9).

(8) Who left?

(9) *Who$_1$ Mary said that t_1 left?

3.4 Multiple Extraction

Nested-dependencies constraints, exemplified in (10) and (11), specify that only the leftmost trace can be bound (for sake of clarity, we forget here about the control verb nature of *know*).

(10) Which$_1$ problems does John know whom$_2$ to talk to t$_2$ about t$_1$?

a. $C_{\text{which?}}\,C_{\text{problems}}\,(\lambda^0 x.\,C_{\text{know}}\,(C_{\text{whom?}}\,(\lambda^0 y.\,C_{\text{to talk to about}}\,y\,x))\,C_{\text{John}})$

(11) *Whom$_1$ does John know which$_2$ problems to talk to t$_1$ about t$_2$?

a. $*C_{\text{whom?}}\,(\lambda^0 y.\,C_{\text{know}}\,(C_{\text{which?}}\,C_{\text{problems}}\,(\lambda^0 x.\,C_{\text{to talk to about}}\,y\,x))\,C_{\text{John}})$

The interrogative extraction follows a first in last out pattern. Despite the close relation of this pattern to the linear order of the sentence, we again implement control at the abstract level. As in Sect. 3.2, extractions are associated with counters that reflect the argument position in the canonical form. Table 6 describes the abstract signature for modelling these cases and $\mathscr{G}_{\text{Cont}_3} = \langle \Sigma_{\text{Cont}_3}, \Sigma_{\text{Syn}}, \mathscr{L}_{\text{Cont}_3}, s\,0\rangle$ is defined the usual way.

Basically, pronouns and their traces get the same counter value. The type of the interrogative pronouns requires sequences of them to have increasing values, greater numbers being abstracted first.

Let us consider a term $t = D_{\text{to talk to about}}\,i\,j\,y\,x$ (to be read as *to talk to* y *about* x) of type $q(h\,i\,j)$ with y of type $np\,i$ and x of type $np\,j$. We show that in order to extract both x and y (and bind them with interrogative pronouns), y has to be extracted first:

[10] The felicity of *The man who John said loves Mary sleeps*, without the complementizer, suggests a type assignment to D_{says} that does not switch the dependant product to emb the way $D_{\text{says that}}$ does.

int	$: \mathtt{type}$	$\mathit{np}, \mathit{n}, \mathit{s}, \mathit{q}$	$: (\mathit{int})\, \mathtt{type}$
D_{John}	$: \mathit{np}\, 0$	$D_{\text{to talk to about}}$	$: (\Pi i, j : \mathit{int})\ (\mathit{np}\, i \multimap \mathit{np}\, j \multimap q\, (h\, i\, j))$
D_{problems}	$: n\, 0$	D_{know}	$: (\Pi i, j : \mathit{int})\ (q\, i \multimap \mathit{np}\, j \multimap q\, (h\, i\, j))$
next	$: \mathit{int} \multimap \mathit{int}$	$D_{\text{whom?}}$	$: (\Pi i : \mathit{int})\ ((\mathit{np}\, (\mathsf{next}\, i) \multimap q\, (\mathsf{next}\, i)) \multimap q\, i)$
		$D_{\text{which?}}$	$: (\Pi i : \mathit{int})\ (n\, 0 \multimap (\mathit{np}\, (\mathsf{next}\, i) \multimap q\, (\mathsf{next}\, i)) \multimap q\, i)$

$$h : \begin{cases} i & 0 \longrightarrow i \\ 0 & j \longrightarrow j \\ \mathsf{next}\, i & j \longrightarrow \mathsf{next}\, i \end{cases}$$

Table 6. Σ_{Cont_3}

- let's assume x is extracted first. The type of the result is $\mathit{np}\, j \multimap q\, i$. Making it a suitable argument of an interrogative pronoun requires $i = j$. But the application results in a term of type $q\,(i-1)$. Then an abstraction of y would result in a term of type $\mathit{np}\, i \multimap q\,(i-1)$ that cannot be argument of another interrogative pronoun. Hence (11-a) can't have an antecedent by $\mathscr{L}_{\mathrm{Cont}_3}$;
- let's now assume that y is extracted first. The type of the result is $\mathit{np}\, i \multimap q\, i$, and when argument of an interrogative pronoun, it results in a term of type $q\,(i-1)$. The result of abstracting then over x is a term of type $\mathit{np}\, j \multimap q\,(i-1)$. To have the latter a suitable argument for an interrogative pronoun requires that $j = i - 1$, or $i = \mathsf{next}\, j$.
  Then, provided $i \geq 2$,

  $$\begin{aligned} &D_{\text{which?}}\,(i-2)\,D_{\text{problems}} \\ &\qquad (\lambda^0 x.D_{\text{know}}\,(i-1)\,0\,(D_{\text{whom?}}\,(i-1)\,(\lambda^0 y.D_{\text{to talk to about}}\, i\,(i-1)\, y\, x))\, D_{\text{John}}) \end{aligned}$$

  is typable (of type $q\,(i-2)$) and is an antecedent of (10-a) by $\mathscr{L}_{\mathrm{Cont}_3}$.

4 Related Approaches

4.1 Parallel Architectures

In this section, we wish to contrast our approach that modifies the abstract level with approaches in which control comes from a specific calculus at the object level. One of this approach specifically relates to the LG framework [23] and aims at introducing Multimodal Categorial Grammar (MMCG) [6] analysis at the phenogrammatical level. The other approach [24] also builds on MMCG analysis. It can actually bee seen as a parallel framework where the both the tectogrammatical level and the phenogrammatical level are MMCG. What is of interest to us is the proposal permitting phonological changes at the phenogrammatical level while the tectogrammatical one is unchanged.

In order to compare the three approaches, it is convenient to introduce the following notations:

Definition 6 (Signs and languages). *A* sign $s = \langle a, o, m \rangle$ *is a triple where:*

- *a is a term belonging to the tectogrammatical level*

- *o is a term belonging to the phenogrammatical level describing the* surface *form associated to a*
- *m is a term belonging to the phenogrammatical level describing the* logical *form associated to a*

In the case of LG and ACG, a is a linear λ-term whereas it is a MMCG proof term in [24].

In all frameworks, a sign $s = \langle a, o, m\rangle$ belong to the language whenever a is of a distinguished type s. Following [23], we call it a generated *sign.*

It is easy to see that in ACG and the approach we developed, o is a λ-term, possibly using the string concatenation operation.

On the other hand, [23] makes o be a multimodal logical formula build from constants and (unary and binary) logical connectives. It not only includes a special binary connective $\circ$ basically representing concatenation, but also any other required connective, in particular families of $\Diamond_i$ and $\Box_i$ operators. Then, the phenogrammatical level can be provided with a consequence relation $\sqsubseteq$ and also, as is standard in MMCG, with proper axioms, or *postulates*. It can then inherit all models of this framework such as [25]'s one for controlling extraction.

Hence, for any sign $s = \langle a, o, m\rangle$, it is possible to define a notion of derivability:

Definition 7 (Derivable and string-meaning signs). *Let $s = \langle a, o, m\rangle$ be a generated sign and o' a logical formula such that $o \sqsubseteq o'$. Then $s' = \langle a, o', m\rangle$ is called a* derivable *sign.*

Let $s = \langle a, o, m\rangle$ be a sign such that o is made only from constants and $\circ$. Then o is said to be readable[11] *and s is said to be a* string-meaning *sign.*

From that perspective, what is now of interest is not the generated signs as such but rather the string-meaning signs. In particular, if $s = \langle a, o, m\rangle$ is a generated sign, the interesting question is whether there exist some o' with $o \sqsubseteq o'$ and o' *readable*. If such an o' exists, then s is expressible, otherwise it is not.

[23, example (35)] is very similar to Example (2). Its analysis is as follows: (2-a), (2-b) and (2-c) are all possible abstract terms so that $s_a = \langle \text{(2-a)}, o_a, m_a\rangle$, $s_b = \langle \text{(2-b)}, o_b, m_b\rangle$ and $s_c = \langle \text{(2-c)}, o_c, m_c\rangle$ are all generated signs. However, there is no readable o such that $o_b \sqsubseteq o$ or $o_c \sqsubseteq o$ because o_b and o_c make use of different kinds of modalities that don't interact through postulates. Hence s_b and s_c can be generated but don't have any readable (or pronounceable) form and only s_a gives rise to a string-meaning sign and is expressible. The approach of [24] is very similar except that the phenogrammatical level is an algebra with a preorder whose maximal elements are the only pronounceable ones.

4.2 Continuation Semantics

In order to take into account constraints on scope related to scope ambiguity and polar sensitivity, [26] uses control operators, in particular delimited continuations with **shift** and **reset** operators in the semantic calculus.

[11] [24] defines *pronounceable* because it deals with phonology rather than with strings.

Parallel architecture such as LG or ACG could also make use of such operators in the syntactic calculus, achieving some of the effects we described. However, applying the continuation-passing style (CPS) transform to those constructs results in a significant increase of the order of types. The impact on the parsing complexity should then be studied carefully in order to get tractable fragments.

4.3 TAG and Lambek Grammars in ACG

We also wish to relate our proposal with similar architectures that have been proposed to model other grammatical formalisms, namely Minimalist Grammars (MG) [27], Tree Adjoining Grammar (TAG) [28], and non-associative Lambek grammars (NL) [29] .

In order to study MG from a logical point of view, [30] studies MG derivations in the ACG framework. Derivations are described at an abstract level (using **move** and **merge** operations) and are further interpreted to get the syntactic representation and the semantic representation at object levels. But rather than giving a direct translation, it is possible to add an intermediate level that corresponds to what is shared between syntax and semantics, but that contains much more than only MG derivations. This is reminiscent of the architecture of Fig. 2(a).

An other example where such an architecture takes place is given in [31] where a first abstract level specifies a syntax-semantics interface for TAG. However, this interface is not constrained enough and accept more than just TAG derivations. Then more abstract levels are added to control the derivations and accept only TAG, local MCTAG and non-local MCTAG.

The encoding of NL into ACG [21] also involves such an architecture. It defines a syntax-semantics interface very close to the one proposed here, and a more abstract level controls in turn this interface in order to discard derivations that are not NL derivations. This last result gives another interesting link to MMCG at a tectogrammatical level rather than at a phenogrammatical one as described in Sect. 4.1, in particular in the case of extraction because of the relation between NL and the calculus with the bracket operator of [25] to deal with islands.

5 Conclusion

Studying constraints related to extraction phenomena, we propose to use dependent types to implement them at an abstract level in the ACG framework. Using dependent types allows us to get finer control on derivations and to discard overgenerating ones. The same methodology has been used to model constraints related to bounding scope displacement, wh-extraction and multiple wh-extraction. This approach, where what appears as constraints at the surface level are rendered at an abstract level, contrasts with other approaches where a derivability notion on surface forms is introduced, and where some of the surface forms get the special status of *readable*.

Interestingly, these two ways to introduce or relax control on derivations are completely orthogonal, hence they could be used together. This gives rise to the question of determining the most appropriate approach given one particular phenomena. Answers could come both from linguistic considerations and from tractability issues of the underlying calculi. Another question is whether the relational semantics behind MMCG could be used, together with the dependent types, to model MMCG derivations within the ACG framework.

Acknowledgments. We would like to thank Carl Pollard and Philippe de Groote for fruitful discussions on earlier versions of this paper.

References

1. Curry, H.B.: Some logical aspects of grammatical structure. In Jakobson, R., ed.: Structure of Language and its Mathematical Aspects: Proceedings of the Twelfth Symposium in Applied Mathematics, American Mathematical Society (1961) 56–68
2. Muskens, R.: Lambda Grammars and the Syntax-Semantics Interface. In van Rooy, R., Stokhof, M., eds.: Proceedings of the Thirteenth Amsterdam Colloquium, Amsterdam (2001) 150–155
3. Muskens, R.: Lambdas, Language, and Logic. In Kruijff, G.J., Oehrle, R., eds.: Resource Sensitivity in Binding and Anaphora. Studies in Linguistics and Philosophy. Kluwer (2003) 23–54
4. 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
5. Lambek, J.: The mathematics of sentence structure. American Mathematical Monthly **65**(3) (1958) 154–170
6. Moortgat, M.: Categorial type logics. In van Benthem, J., ter Meulen, A., eds.: Handbook of Logic and Language. Elsevier Science Publishers, Amsterdam (1996) 93–177
7. Carpenter, B.: Type-Logical Semantics. The MIT Press (1997)
8. Montague, R.: The proper treatment of quantification in ordinary english. In: Formal Philosophy: Selected Papers of Richard Montague. Yale University Press (1974) Re-edited in ”Formal Semantics: The Essential Readings”, Paul Portner and Barbara H. Partee, editors. Blackwell Publishers, 2002.
9. Ranta, A.: Type Theoretical Grammar. Oxford University Press (1994)
10. 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.
11. de Groote, P., Yoshinaka, R., Maarek, S.: On two extensions of abstract categorial grammars. In Dershowitz, N., Voronkov, A., eds.: Logic for Programming, Artificial Intelligence, and Reasoning, 14th International Conference, LPAR 2007, Yerevan, Armenia, October 15-19, 2007, Proceedings. Volume 4790 of Lecture Notes in Computer Science., Springer (2007) 273–287
12. Salvati, S.: Problèmes de filtrage et problèmes d'analyse pour les grammaires catégorielles abstraites. PhD thesis, Institut National Polytechnique de Lorraine (2005)
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. Kanazawa, M.: Parsing and generation as datalog queries. In: Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics (ACL), Prague, Czech Republic, Association for Computational Linguistics (June 2007) 176–183 http://www.aclweb.org/anthology/P/P07/P07-1023.
15. Johnson, M.: Proof nets and the complexity of processing center embedded constructions. Journal of Logic, Language and Information **7**(4) (1998)
16. Morrill, G.V.: Incremental processing and acceptability. Computational Linguistics **26**(3) (September 2000) 319–338
17. 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.
18. de Groote, P., Pogodalla, S., Pollard, C.: On the syntax-semantics interface: From convergent grammar to abstract categorial grammar. In Kanazawa, M., Ono, H., de Queiroz, R., eds.: 16th Workshop on Logic, Language, Information and Computation. Volume 5514., Japon Tokyo, Springer (2009) 182–196 http://hal.inria.fr/inria-00390490/en/.
19. Pogodalla, S.: Computing semantic representation: Towards ACG abstract terms as derivation trees. In: Proceedings of the Seventh International Workshop on Tree Adjoining Grammar and Related Formalisms (TAG+7). (May 2004) 64–71 http://www.cs.rutgers.edu/TAG+7/papers/pogodalla.pdf.
20. Shieber, S.M.: Unifying synchronous tree-adjoining grammars and tree transducers via bimorphisms. In: Proceedings of the 11th Conference of the European Chapter of the Association for Computational Linguistics (EACL-06), Trento, Italy (3–7 April 2006) http://www.aclweb.org/anthology-new/E/E06/E06-1048.pdf.
21. Retoré, C., Salvati, S.: A faithful representation of non-associative lambek grammars in abstract categorial grammars. Journal of Logic, Language and Information **19**(2) (2010) 185–200 http://www.springerlink.com/content/f48544n414594gw4/.
22. Moot, R., Piazza, M.: Linguistic applications of first order intuitionistic linear logic. Journal of Logic, Language and Information **10** (2001) 211–232
23. Muskens, R.: Separating syntax and combinatorics in categorial grammar. Research on Language and Computation **5**(3) (September 2007) 267–285
24. Kubota, Y., Pollard, C.: Phonological interpretation into preordered algebras. In: Proceedings of the 11th Meeting of the Association for Mathematics of Language (MOL'11). (2009) http://www.ling.ohio-state.edu/~kubota/papers/mol11_proc.pdf.
25. Morrill, G.: Categorial formalisation of relativisation: Islands, extraction sites and pied piping. Technical Report LSI-92-23-R,, Departament de Llenguatges i Sistemes Informàtics, Universitat Politècnica de Catalunya (1992)
26. Shan, C.C.: Delimited continuations in natural language : Quantification and polarity sensitivity. In Thielecke, H., ed.: Proceedings of the 4th continuations workshop, School of Computer Science, University of Birmingham (2004) 55–64
27. Stabler, E.: Derivational minimalism. In Retoré, C., ed.: Logical Aspects of Computational Linguistics, LACL'96. Volume 1328 of LNCS/LNAI., Springer-Verlag (1997) 68–95
28. Joshi, A.K., Schabes, Y.: Tree-adjoining grammars. In Rozenberg, G., Salomaa, A., eds.: Handbook of formal languages. Springer (1997)
29. Lambek, J.: On the calculus of syntactic types. In Jacobsen, R., ed.: Structure of Language and its Mathematical Aspects. Proceedings of Symposia in Applied Mathematics, XII. American Mathematical Society (1961)
30. Salvati, S.: Minimalist grammars in the light of logic. (to appear)
31. Kanazawa, M., Pogodalla, S.: Advances in abstract categorial grammars: Language theory and linguistic modelling. ESSLLI 2009 Lecture Notes (2009)