Tree-Adjoining Grammars as Abstract Categorial Grammars

Philippe de Groote
LORIA UMR n° 7503 – INRIA
Campus Scientifique, B.P. 239
54506 Vandœuvre lès Nancy Cedex – France
e-mail: Philippe.de.Groote@loria.fr

1. Introduction

We recently introduced abstract categorial grammars (ACGs) (de Groote, 2001) as a new categorial formalism based on Girard linear logic (Girard, 1987). This formalism, which derives from current type-logical grammars (Carpenter, 1996; Moortgat, 1997; Morrill, 1994; Oehrle, 1994), offers some novel features:

- Any ACG generates two languages, an abstract language and an object language. The abstract language may be thought as a set of abstract grammatical structures, and of the object language as the set of concrete forms generated from these abstract structures. Consequently, one has a direct control on the parse structures of the grammar.
- The langages generated by the ACGs are sets of linear λ-terms. This may be seen as a generalization of both string-langages and tree-langages.
- ACGs are based on a small set of mathematical primitives that combine via simple composition rules. Consequently, the ACG framework is rather flexible.

Abstract categorial grammars are not intended as yet another grammatical formalism that would compete with other established formalisms. It should rather be seen as the kernel of a grammatical framework — in the spirit of (Ranta, 2002) — in which other existing grammatical models may be encoded. This paper illustrates this fact by showing how tree-adjoining grammars (Joshi and Schabes, 1997) may be embedded in abstract categorial grammars.

This embedding exemplifies several features of the ACG framework:

- The fact that the basic objects manipulated by an ACG are λ-terms allows higher-order operations to be defined. Typically, tree-adjunction is such a higher-order operation (Abrusci, Fouqueré and Vauzeilles, 1999; Joshi and Kulick, 1997; Mönnich, 1997).
- The flexibility of the framework allows the embedding to be defined in two stages. A first ACG allows the tree langage of a given TAG to be generated. The abstract language of this first ACG corresponds to the derivation trees of the TAG. Then, a second ACG allows the corresponding string language to be extracted. The abstract language of this second ACG corresponds to the object language of the first one.

2. Abstract Categorial Grammars

This section defines our notion of an abstract categorial grammar. We first introduce the notions of *linear implicative types, higher-order linear signature, linear λ-terms* built upon a higher-order linear signature, and *lexicon*.

Let A be a set of atomic types. The set $\mathscr{T}(A)$ of *linear implicative types* built upon A is inductively defined as follows:

1. if $a \in A$, then $a \in \mathscr{T}(A)$;
2. if $\alpha, \beta \in \mathscr{T}(A)$, then $(\alpha \multimap \beta) \in \mathscr{T}(A)$.

A *higher-order linear signature* consists of a triple $\Sigma = \langle A, C, \tau \rangle$, where:

1. A is a finite set of atomic types;

2. C is a finite set of constants;

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

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

1. if $c \in C$, then $c \in \Lambda(\Sigma)$;

2. if $x \in X$, then $x \in \Lambda(\Sigma)$;

3. if $x \in X$, $t \in \Lambda(\Sigma)$, and x occurs free in t exactly once, then $(\lambda x.\, t) \in \Lambda(\Sigma)$;

4. if $t, u \in \Lambda(\Sigma)$, and the sets of free variables of t and u are disjoint, then $(t\, u) \in \Lambda(\Sigma)$.

$\Lambda(\Sigma)$ is provided with the usual notion of capture avoiding substitution, α-conversion, and β-reduction (Barendregt, 1984).

Given a higher-order linear signature $\Sigma = \langle A, C, \tau \rangle$, each linear λ-term in $\Lambda(\Sigma)$ may be assigned a linear implicative type in $\mathscr{T}(A)$. This type assignment obeys an inference system whose judgements are sequents of the following form:

$$\Gamma \vdash_{\Sigma} t : \alpha$$

where:

1. Γ is a finite set of λ-variable typing declarations of the form '$x : \beta$' (with $x \in X$ and $\beta \in \mathscr{T}(A)$), such that any λ-variable is declared at most once;

2. $t \in \Lambda(\Sigma)$;

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

The axioms and inference rules are the following:

$$\vdash_{\Sigma} c : \tau(c) \quad \text{(cons)}$$

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

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

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

Given two higher-order linear signatures $\Sigma_1 = \langle A_1, C_1, \tau_1 \rangle$ and $\Sigma_2 = \langle A_2, C_2, \tau_2 \rangle$, a lexicon $\mathscr{L} : \Sigma_1 \to \Sigma_2$ is a realization of Σ_1 into Σ_2, i.e., an interpretation of the atomic types of Σ_1 as types built upon A_2 together with an interpretation of the constants of Σ_1 as linear λ-terms built upon Σ_2. These two interpretations must be such that their homomorphic extensions commute with the typing relations. More formally, a *lexicon* $\mathscr{L}$ from Σ_1 to Σ_2 is defined to be a pair $\mathscr{L} = \langle F, G \rangle$ such that:

1. $F : A_1 \to \mathscr{T}(A_2)$ is a function that interprets the atomic types of Σ_1 as linear implicative types built upon A_2;

2. $G : C_1 \to \Lambda(\Sigma_2)$ is a function that interprets the constants of Σ_1 as linear λ-terms built upon Σ_2;

3. the interpretation functions are compatible with the typing relation, *i.e.*, for any $c \in C_1$, the following typing judgement is derivable:
$$\vdash_{\Sigma_2} G(c) : \hat{F}(\tau_1(c)),$$
where $\hat{F}$ is the unique homomorphic extension of F.

We are now in a position of defining the notion of abstract categorial grammar. An *abstract categorial grammar* is a quadruple $\mathscr{G} = \langle \Sigma_1, \Sigma_2, \mathscr{L}, s \rangle$ where:

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

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

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

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

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

In words, the abstract language generated by $\mathscr{G}$ is the set of closed linear λ-terms, built upon the abstract vocabulary Σ_1, whose type is the distinguished type s. On the other hand, the *object language* generated by $\mathscr{G}$ ($\mathcal{O}(\mathscr{G})$) is defined to be the image of the abstract language by the term homomorphism induced by the lexicon $\mathscr{L}$:

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

3. Representing Tree-Adjoining Grammars

In this section, we explain how to construct an abstract categorial grammar that generates the same tree langage as a given tree-adjoining grammar.

Let $G = \langle \Sigma, N, I, A, S\rangle$ be a tree-adjoining grammar, where Σ, N, I, A, and S are the set of terminal symbols, the set of non-terminal symbols, the set of initial trees, the set of auxiliary tree, and the distinguished non-terminal symbol, respectively. We associate to G an ACG $\mathscr{G}^G = \langle \Sigma_1^G, \Sigma_2^G, \mathscr{L}^G, s^G\rangle$ as follows.

The set of atomic types of Σ_1^G is made of two copies of the set of non-terminal symbols. Given $\alpha \in N$, we write α_S and α_A for the two corresponding atomic types. Then, we associate a constant

$$c_T : \gamma_{1A} \multimap \cdots \gamma_{mA} \multimap \beta_{1S} \multimap \cdots \beta_{nS} \multimap \alpha_S$$

to each initial tree T whose root node is labelled by α, whose substitution nodes are labeled by $\beta_1, \ldots, \beta_n$, and whose interior nodes are labeled by $\gamma_1, \ldots, \gamma_m$. Similarly, we associate a constant

$$c_{T'} : \gamma_{1A} \multimap \cdots \gamma_{mA} \multimap \beta_{1S} \multimap \cdots \beta_{nS} \multimap \alpha_A \multimap \alpha_A$$

to each auxiliary tree T' whose root node is labelled by α, whose substitution nodes are labeled by $\beta_1, \ldots, \beta_n$, and whose interior nodes are labeled by $\gamma_1, \ldots, \gamma_m$. Finally, we also associate to each non-terminal symnbol $\alpha \in N$, a constant I_α of type α_A. This concludes the specification of the abstract vocabulary.

The object vocabulary Σ_2^G allows labelled trees to be represented. Its set of atomic types contains only one element : τ (for *tree*). Then, its set of constants consists in:

1. constants of type τ corresponding to the terminal symbols of G;

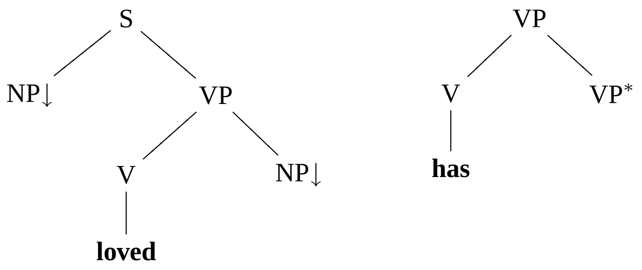
2. for each non-terminal symbol α, constants

$$\alpha_i : \underbrace{\tau \multimap \cdots \tau}_{i \text{ times}} \multimap \tau$$

for $1 \leq i \leq k$, where k is the maximal branching of the interior nodes labelled with α that occur in the initial and auxiliary trees of G.

Clearly, the terms of type τ that can be built by means of the above set of constants correspond to trees whose frontier nodes are terminal symbols and whose interior nodes are labelled with non-terminal symbols.

It remains to define the lexicon $\mathscr{L}^G$. The rough idea is to represent the initial trees as trees (i.e., terms of type τ) and the auxiliary trees as functions over trees (i.e., terms of type $\tau \multimap \tau$). Consequently, for each $\alpha \in N$, we let $\mathscr{L}^G(\alpha_S) = \tau$ and $\mathscr{L}^G(\alpha_A) = \tau \multimap \tau$. Accordingly, the susbstitution nodes will be represented as first-order λ-variables of type τ , and the adjunction nodes as second-order λ-variables of type $\tau \multimap \tau$. The object representation of the elementary trees is then straightforward. Consider, for instance, the following initial tree and auxiliary tree:



According to our construction, the two abstract constants corresponding to these trees have the following types:

$$C_{\text{loved}} : \text{S}_A \multimap \text{VP}_A \multimap \text{V}_A \multimap \text{NP}_S \multimap \text{NP}_S \multimap \text{S}_S \quad \text{and} \quad C_{\text{has}} : \text{VP}_A \multimap \text{V}_A \multimap \text{VP}_A \multimap \text{VP}_A$$

Then, the realization of these two constants is as follows:

$$\begin{array}{rcl} \mathscr{L}^G(C_{\text{loved}}) & = & \lambda F.\, \lambda G.\, \lambda H.\, \lambda x.\, \lambda y.\, F\,(\text{S}_2\, x\, (G\, (\text{VP}_2\, (H\, (\text{V}_1\, \textbf{loved}))\, y))) \\ \mathscr{L}^G(C_{\text{has}}) & = & \lambda F.\, \lambda G.\, \lambda H.\, \lambda x.\, F\,(\text{VP}_2\, (G\, (\text{V}_1\, \textbf{has}))\, (H\, x)) \end{array}$$

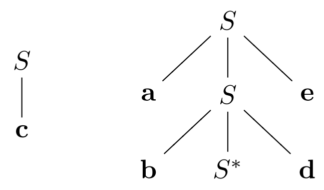
In order to derive actual trees, the second-order variables should eventually disappear. The abstract constants I_α have been introduced to this end. Consequently they are realized by the identity function, i.e., $\mathscr{L}^G(I_\alpha) = \lambda x.\, x$.

Finally, the distinguished type of $\mathscr{G}^G$ is defined to be S_S. This completes the definition of the ACG $\mathscr{G}^G$ associated to a TAG G. Then, the following proposition may be easily established.

PROPOSITION *Let G be a TAG. The tree-language generated by G is isomorphic to the object language of the ACG $\mathscr{G}^G$ associated to G.* □

4. Example

Consider the TAG with the following initial tree and auxiliary tree:



It generates a non context-free language whose intersection with the regular language $a^*b^*c\,d^*e^*$ is $a^nb^nc\,d^ne^n$. According to the construction of Section 3, this TAG may be represented by the ACG, $\mathscr{G} = \langle \Sigma_1, \Sigma_2, \mathscr{L}, S\rangle$, where:

$$\begin{array}{l} \Sigma_1 = \langle\ \{S_S, S_A\}, \{c_i, c_a, I\}, \\ \quad \{c_i \mapsto (S_A \multimap S_S), \\ \quad\ c_a \mapsto (S_A \multimap (S_A \multimap (S_A \multimap S_A))), \\ \quad\ I \mapsto S_A\}\ \rangle \end{array}$$

$$\begin{array}{l} \Sigma_2 = \langle\ \{\tau\}, \{\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}, \mathbf{e}, S_1, S_3\}, \\ \quad \{\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}, \mathbf{e} \mapsto \tau, \\ \quad\ S_1 \mapsto (\tau \multimap \tau), \\ \quad\ S_3 \mapsto (\tau \multimap (\tau \multimap (\tau \multimap \tau)))\}\ \rangle \end{array}$$

$$\begin{array}{l} \mathscr{L} = \langle\ \{S_S \mapsto \tau, \\ \quad\ S_A \mapsto (\tau \multimap \tau)\}, \\ \quad \{c_i \mapsto \lambda f.\, f\,(S_1\, \mathbf{c}), \\ \quad\ c_a \mapsto \lambda f.\, \lambda g.\, \lambda h.\, \lambda x.\, f\,(S_3\, \mathbf{a}\, (g\, (S_3\, \mathbf{b}\, (h\, x)\, \mathbf{d}))\, \mathbf{e}), \\ \quad\ I \mapsto \lambda x.\, x\}\ \rangle \end{array}$$

5. Extracting the string languages

There is a canonical way of representing strings as linear λ-terms. It consists of encoding a string of symbols as a composition of functions. Consider an arbitrary atomic type σ, and define the type '*string*' to be $(\sigma \multimap \sigma)$. Then, a string such as '$abbac$' may be represented by the linear λ-term:

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

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

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

and concatenation is defined to be functional composition:

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

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

This allows a second ACG, $\mathscr{G}'^{G}$, to be defined. Its abstract vocabulary is the object vocabulary Σ_2^G of $\mathscr{G}^G$. Its object vocabulary allows string of terminal symbols to be represented. Its lexicon interprets each constant of type τ as an atomic string, and each constant α_i as a concatenation operator. This second ACG, $\mathscr{G}'^{G}$, extracts the yields of the trees. Then, by composing $\mathscr{G}^G$ with $\mathscr{G}'^{G}$, one obtains an ACG which generates the same string-language as G.

Let us continue the example of Section 4. The second ACG, $\mathscr{G}' = \langle \Sigma_1', \Sigma_2', \mathscr{L}', S' \rangle$, is defined as follows:

$$\Sigma_1' = \Sigma_2$$

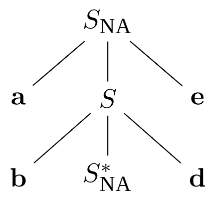
$$\begin{aligned}\Sigma_2' = \langle\ & \{\sigma\}, \{a, b, c, d, e\},\\ & \{a, b, c, d, e \mapsto (\sigma \multimap \sigma)\}\ \rangle\end{aligned}$$

$$\begin{aligned}\mathscr{L}' = \langle\ & \{\tau \mapsto (\sigma \multimap \sigma)\},\\ & \{\mathbf{a} \mapsto \lambda x.\, a\, x,\\ & \ \mathbf{b} \mapsto \lambda x.\, b\, x,\\ & \ \mathbf{c} \mapsto \lambda x.\, c\, x,\\ & \ \mathbf{d} \mapsto \lambda x.\, d\, x,\\ & \ \mathbf{e} \mapsto \lambda x.\, e\, x,\\ & \ S_1 \mapsto \lambda f.\, \lambda x.\, f\, x,\\ & \ S_3 \mapsto \lambda f.\, \lambda g.\, \lambda h.\, f\,(g\,(h\, x))\}\ \rangle\end{aligned}$$

6. Expressing Adjoining constraints

Adjunction, which is enabled by second-order variables at the object level, is explicitly controlled at the abstract level by means of types. This typing discipline may be easily refined in order to express adjoining constraints such as selective, null, or obligatory adjunction.

Consider again the TAG given in Section 4. By adding the following null adjunction constraints on its auxiliary tree:



one obtains a grammar that generates exactly the non context-free language $a^n b^n c\, d^n e^n$. These constraints may be expressed in a simple and natural way. It suffices to exclude the constrained nodes from the arguments of the λ-term corresponding to the auxiliary tree. This gives the following modified ACG:

$$\begin{aligned}\Sigma_1 = \langle\ & \{S_S, S_A\}, \{c_i, c_a, I\},\\ & \{c_i \mapsto (S_A \multimap S_S),\\ & \ c_a \mapsto (S_A \multimap S_A),\\ & \ I \mapsto S_A\}\ \rangle\end{aligned}$$

$$\begin{aligned}\Sigma_2 = \langle\ & \{\tau\}, \{\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}, \mathbf{e}, S_1, S_3\},\\ & \{\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}, \mathbf{e} \mapsto \tau,\\ & \ S_1 \mapsto (\tau \multimap \tau),\\ & \ S_3 \mapsto (\tau \multimap (\tau \multimap (\tau \multimap \tau)))\}\ \rangle\end{aligned}$$

$$\begin{aligned}\mathscr{L} = \langle\ & \{S_S \mapsto \tau,\\ & \ S_A \mapsto (\tau \multimap \tau)\},\\ & \{c_i \mapsto \lambda f.\, f\,(S_1\,\mathbf{c}),\\ & \ c_a \mapsto \lambda f.\, \lambda x.\, S_3\,\mathbf{a}\,(f\,(S_3\,\mathbf{b}\,x\,\mathbf{d}))\,\mathbf{e},\\ & \ I \mapsto \lambda x.\, x\}\ \rangle\end{aligned}$$

The other kinds of adjunction constraints may be expressed in a similar way.

References

Abrusci, M., C. Fouqueré and J. Vauzeilles. 1999. Tree-adjoining grammars in a fragment of the Lambek calculus. *Computational Linguistics*, 25(2):209–236.

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

Carpenter, B. 1996. *Type-Logical Semantics*. Cambridge, Massachussetts and London England: MIT Press.

de Groote, Ph. 2001. Towards Abstract Categorial Grammars. In *Association for Computational Linguistics, 39th Annual Meeting and 10th Conference of the European Chapter, Proceedings of the Conference*, pages 148–155.

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

Joshi, A. K. and S. Kulick. 1997. Partial Proof Trees as Building Blocks for a Categorial Grammar. *Linguistic & Philosophy*, 20:637–667.

Joshi, A. K. and Y. Schabes. 1997. Tree-adjoining grammars. In G. Rozenberg and A. Salomaa, editors, *Handbook of formal languages*, volume 3. Springer, chapter 2.

Mönnich, U. 1997. Adjunction as substitution. In G.-J. Kruijff, G. Morrill and D. Oehrle, editors, *Formal Grammar*, pages 169–178. FoLLI.

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

Morrill, G. 1994. *Type Logical Grammar: Categorial Logic of Signs*. Dordrecht: Kluwer Academic Publishers.

Oehrle, R. T. 1994. Term-labeled categorial type systems. *Linguistic & Philosophy*, 17:633–678.

Ranta, A. 2002. Grammatical Framework, a type-theoretical grammar formalism. Working paper, submitted for publication.