An ACG Analysis of the G-TAG Generation Process
Laurence Danlos, Aleksandre Maskharashvili, Sylvain Pogodalla

To cite this version:

HAL Id: hal-00999595
https://inria.hal.science/hal-00999595
Submitted on 3 Jun 2014

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
An ACG Analysis of the G-TAG Generation Process

Laurence Danlos, Aleksandre Maskharashvili, and Sylvain Pogodalla

Université Paris Diderot (Paris 7) INRIA Villers-lès-Nancy, France
ALPAGE, INRIA Paris–Rocquencourt Université de Lorraine,
Institut Universitaire de France, Paris, France CNRS, LORIA, UMR 7503
laurence.danlos@inria.fr Vandœuvre-lès-Nancy, France
aleksandre.maskharashvili@inria.fr
sylvain.pogodalla@inria.fr

Abstract

This paper presents an encoding of Generation-TAG (G-TAG) within Abstract Categorial Grammars (ACG). We show how the key notions of G-TAG have a natural interpretation in ACG, allowing us to use its reversibility property for text generation. It also offers solutions to several limitations of G-TAG.

1 Motivations

G-TAG (Danlos, 1998; Danlos, 2000) is a formalism based on the Tree Adjoining Grammar (TAG) formalism (Joshi et al., 1975; Joshi and Schabes, 1997) dedicated to text generation. It focuses on providing several notions to support useful data structures, such as g-derivation trees or lexical databases, to effectively relate a surface form (a derived tree or a string) to a conceptual representation. An actual implementation in ADA was first provided for French (Meunier, 1997), and it has recently been implemented in the .NET framework as the EasyText NLG system and is operational at Kantar Media, a French subsidiary company of TNS-Sofres (Danlos et al., 2011).

The G-TAG proposal can be seen as a result of the observation of the mismatch between the derivation tree notion of TAG and the expected semantic dependencies (Schabes and Shieber, 1994) from a generation perspective. Several approaches that extend the derivation tree notion of TAG have been proposed to overcome this difficulty. Other approaches showed that the derivation trees still could be used without additional modifications. Such approaches rely on unification (Kallmeyer and Romero, 2004; Kallmeyer and Romero, 2007) or a functional approach to TAG (Pogodalla, 2009).

We propose then to study G-TAG under the ACG perspective. We show that the key notion of g-derivation tree naturally express itself in this framework. The surface form construction from a conceptual representation can then use the general algorithms of ACG, the very same ones that can be used in parsing to analyze mildly context sensitive languages (TAG generated language, LCFRS) (de Groote and Pogodalla, 2004), following (Kanazawa, 2007)’s proposal here applied to give an ACG account of G-TAG. We do not consider here the G-TAG treatment of preferences between the different realizations of the same input. Similarly, we do not consider the generation of pronouns used in G-TAG and we will work on integrating a theory of generation of referring expressions.

2 Sketching G-TAG

G-TAG deals with the How to say it? task of generation. The input is a conceptual representation. A G-TAG grammar includes elementary trees, as any TAG grammar. But it also makes g-derivation trees primary objects, relating them to the elementary trees and considering them as pivot to the conceptual representation level.

Conceptual Representation G-TAG conceptual representation makes use of notions as second order relation, first order relation and thing. Second order relations have two arguments which are relations (either first or second order ones) and typically correspond to discourse relations.

1Synchronous approaches (Nesson and Shieber, 2006) are similar in many respects, as shown in (Storoshenk and Frank, 2012).
whereas first order relations have things as their arguments. While (Danlos, 2000) uses reified formulas of a logical conceptual representation language as G-TAG inputs, it can also be represented as a higher-order logical formula (Meunier, 1997) or as a SDRT-like formula (Danlos et al., 2001). We follow here this presentation. Equation (1) exemplifies an input that could be realized as Jean a passé l’aspirateur pour être récompensé par Marie. Puis il a fait une sieste (John has vacuumed in order to be rewarded by Mary. Then he took a nap).

\[
\text{SUCCESSION}(\text{GOAL}(\text{VACUUMING}(\text{Jean})), \\
\quad \text{REWARDING}(\text{Marie, Jean}), \\
\quad \text{NAPPING}(\text{Jean})) \quad (1)
\]

**G-TAG Lexical Database** A lexical entry of G-TAG corresponds to a lemma. For each lexical entry (i.e. lemma) there is a set of TAG elementary trees which corresponds to it. Among the TAG elementary trees that correspond to a given lexical entry, there is the canonical representative, and all the other representatives are represented by adding features to the canonical representative. For example, if the lexical entry is to love, then the canonical representative will be the active form of the verb to love. Then the passive alternation is represented by adding a feature [+passive] to to love. Moreover, all the lexical entries attached to a concept (such as SUCCESSION) belong to a same lexical base. So for a concept, there can be a lexical entry describing verbal realizations of the concept. These realizations can correspond to the active or to the passive forms, etc. There can also be a lexical entry which corresponds to nominal realizations, etc.

**G-Derivation Trees** A TAG derivation tree can be seen as a record of the substitutions and adjunction occurring during a TAG analysis. The same is true for g-derivation tree. However, while TAG derivation trees are considered as a by-product, with inflected anchors, G-TAG derivation trees are first class structures that are combined in order to reflect the conceptual input. To abstract from the surface form and from the derived tree they can relate to, they don’t correspond to inflected forms but bear features that are used in a post-processing step. Complex g-derivation trees are built by going through the dynamic selection process of a lexical item from the set of appropriate candidates for a given concept. So contrary to TAG derivation trees, they are not fully instantiated trees: their arguments are represented by variables whose lexicalization are not carried out yet.

**G-Derived Trees** A g-derivation tree defines a unique g-derived tree corresponding to it. This correspondance is maintained all along the realization process and a post-processing module outputs the surface representation (text) from the g-derived tree. In addition to inflecting forms using the feature values it can make some rewriting to propose different versions of the initial text. In this particular sense, g-derived tree corresponds to possibly multiple text outputs generated by the post-processing module.

3 The G-TAG Generation Process

Let us assume the input of Equation 1. The G-TAG process starts by lexicalizing relations that have the widest scope in the conceptual representation: typically second order relations, then first order relations, and things.\(^2\) Back to the example, we first lexicalize the second order relation SUCCESSION. Several items are associated with this relation: après (after), avant (before), ensuite (afterwards), auparavant (beforehand), puis (then), etc. Each of them has two arguments, however, some of them produce texts comprising two or more sentences, like ensuite(afterwards); some of them can produce either two sentence texts or one sentence text, while others produce only one sentence. For instance, Jean a passé l’aspirateur. Ensuite, il a fait une sieste (John has vacuumed. Afterwards, he took a nap) is a two sentences text while John a fait une sieste après avoir passé l’aspirateur (John took a nap after having vacuumed) is a one sentence text. For this reason, items describing the arguments or the result of second order relations have features expressing the following constraints: (+T, +S) indicates it is a text (two ore more sentences); (+S) indicates it is either a single sentence or a text; (−T, +S) indicates it is a sentence (not a text). Every second order relation has three features: one for output, and two for inputs.\(^3\)

\(^2\)Correctness of the process is ensured because the grammars don’t contain auxiliary trees that would reverse the predication order. (Danlos, 2000) argues such cases don’t occur in technical texts, the first target of G-TAG. We don’t elaborate on this point since the ACG approach we propose remove this constraint for free.

\(^3\)In G-TAG, any discourse connective has exactly two arguments. A discussion about this point is provided in (Dan-
Let us assume that the G-TAG g-derivation tree \textit{ensuite} (+T, +S) belonging to the lexical database associated with the concept \textit{succession} is first chosen, resulting in a text rather than a sentence (illustrated by the leftmost g-derivation tree of Figure 1). The process then tries to realize its two arguments. The first one involves the \textit{goal} relation that can be realized either by \textit{pour} (in order to) or by \textit{pour que} (so that), as exemplified by the rightmost g-derivation trees of Figure 1. Both have features \((-T, +S)\) for the inputs (i.e. arguments) and return a tree labeled at the root by \((-T, +S)\).

\begin{verbatim}
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>\textit{ensuite}</td>
<td>\textit{pour}</td>
<td>\textit{pour que}</td>
</tr>
<tr>
<td>(+T, +S)</td>
<td>(-T, +S)</td>
<td>(-T, +S)</td>
</tr>
</tbody>
</table>
\end{verbatim}

Figure 1: G-derivation trees samples

Despite \textit{pour} and \textit{pour que} bearing the same features, the syntactic trees corresponding to \textit{pour} and \textit{pour que} are quite different. For \textit{pour que} \textit{S} substitution nodes can be substituted by two tensed sentences, while \textit{pour} takes a finite sentence and a “sentence” in the infinitive form without any nominal subject. Figure 2 shows the associated elementary trees. Selecting one or the other during the generation process restricts the possible realizations for the arguments. This is enforced by a feature associated to the elementary tree, namely the \((+\text{reduc-subj})\) feature as shown in Fig. 3. Again, we may assume that G-TAG selects \textit{pour}, which will enforce, because of the associated elementary trees, that the subject of the first and the second arguments are the same. Afterwards, we need to lexicalize these two arguments with a common subject \textit{Jean}. From a semantic point of view, the agent of \textit{vacuuming} has to be the beneficiary of \textit{rewarding}: récompenser (to reward), donner-récompense (to give-reward), and recevoir-récompense (to receive-reward). Let us notice that donner-récompense does not meet the constraint on a shared subject as it cannot have the rewardee as its subject\(^4\). The remaining options are: recevoir-récompense, whose canonical representation has the rewardee as subject; and récompense whose passive construction has the rewardee as subject. Assuming a choice of récompenser\(^{+}\text{passive}\), the lexicalizations of the arguments of the first order relations remain. As Marie occurs only once and in subject position, it can only be lexicalized as Marie. On the other hand, Jean three times: one will be the implicit subject of the subordinate, then as argument of \textit{vacuuming} and \textit{napping}. Therefore it can be either lexicalized in both of the cases as Jean, or Jean and the pronoun \textit{il} (he). In G-TAG, there are some post-processing rules that take care of the generation of referring expressions, but not in a really principled way so we do not demonstrate them here. We assume a lexicalization by Jean in both cases. Figure 3 shows the g-derivation tree associated with the input of Equation 1 and Fig. 4 show the unique resulting (non-flected) derived tree. The post-processing modules then outputs: Jean a passé l’aspirateur pour être récompensé par Marie. Ensuite, il a fait une sieste. (John vacuumed in order to be rewarded by Mary. Afterwards, he took a nap.)

\begin{verbatim}
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>\textit{ensuite}</td>
<td>\textit{pour}</td>
<td>\textit{faire-la-sieste}</td>
</tr>
<tr>
<td>\textit{Jean}</td>
<td>\textit{Jean}</td>
<td>\textit{Marie}</td>
</tr>
<tr>
<td>(+\text{reduc-subj})</td>
<td>(+\text{reduc-subj}, +\text{passive})</td>
<td></td>
</tr>
</tbody>
</table>
\end{verbatim}

Figure 3: Fully instantiated g-derivation tree

4 ACG Definition

Abstract Categorial Grammars (ACGs) (de Groote, 2001) are a type theo-

\(^4\)It lacks passivation in French and there is no form equivalent to: John was given a reward by Mary.

\(^5\)Of course, all these branching points offer several realizations of the same entry. But for explanatory purposes, we describe only one at each step.
retorical framework that is able to encode several grammatical formalisms (de Groot and Pogodalla, 2004). An ACG defines two languages: the abstract one and the object one. The abstract level describe the admissible parse structures and a lexicon maps these structures to the ones we observe at the object level (strings for surface forms, logical formulas for semantic forms). In all cases, the considered languages are sets of λ-terms that generalize string and tree languages.

**Definition.** A higher-order linear signature (also called a vocabulary) is defined to be a triple \( \Sigma = (A, C, \tau) \), where:
- \( A \) is a finite set of atomic types (also noted \( A_\Sigma \)),
- \( C \) is a finite set of constants (also noted \( C_\sigma \)),
- and \( \tau \) is a mapping from \( C \) to \( \mathbb{F}_A \) the set of types built on \( A \): \( \mathbb{F}_A := A | \mathbb{F}_A \rightarrow \mathbb{F}_A \) (also noted \( \mathbb{F}_\Sigma \)).

Given a higher-order linear signature \( \Sigma, \Lambda(\Sigma) \) is the set of \( \lambda \)-terms built on \( \Sigma \), and for \( t \in \Lambda(\Sigma) \) and \( \alpha \in \mathbb{F}_\Sigma \) such that \( t \) has type \( \alpha \), we note \( t : \Sigma \alpha \) (the \( \Sigma \) subscript is omitted when obvious from the context).

**Definition.** An abstract categorial grammar is a quadruple \( \mathcal{G} = (\Sigma, \Xi, \mathcal{L}, s) \) where:
1. \( \Sigma \) and \( \Xi \) are two higher-order linear signatures, which are called the abstract vocabulary and the object vocabulary, respectively;
2. \( \mathcal{L} : \Sigma \rightarrow \Xi \) is a lexicon from the abstract vocabulary to the object vocabulary. It is a homomorphism that maps types and terms built on \( \Sigma \) to types and terms built on \( \Xi \) as follows:
   - if \( \alpha \rightarrow \beta \in \mathbb{F}_\Sigma \) then \( \mathcal{L}(\alpha \rightarrow \beta) = \mathcal{L}(\alpha) \rightarrow \mathcal{L}(\beta) \)
   - if \( x \in \Lambda(\Sigma) \) (resp. \( \lambda x.t \in \Lambda(\Sigma) \) and \( t \in \Lambda(\Sigma) \)) then \( \mathcal{L}(x) = x \) (resp. \( \mathcal{L}(\lambda x.t) = \lambda x.\mathcal{L}(t) \) and \( \mathcal{L}(t u) = \mathcal{L}(t) \mathcal{L}(u) \))

It is then enough to define \( \mathcal{L} \) on the atomic types and on the constants of \( \Sigma \) to define it on all types and terms, provided that for any constant \( c : \alpha \) of \( \Sigma \) we have \( \mathcal{L}(c) : \mathcal{L}(\alpha) \).

We note \( t :=_g u \) if \( \mathcal{L}(t) = u \) and omit the \( \mathcal{G} \) subscript if obvious from the context.

3. \( s \in \mathbb{F}_\Sigma \) is a type of the abstract vocabulary, which is called the distinguished type of the grammar.

Table 1 provides an ACG example \( \mathcal{G}_diced_\text{trees} \) where the abstract typed constants of \( \Sigma_{\text{diced}} \) encode the combinatorial properties of the associated (through the lexicon \( \mathcal{L}_{diced_\text{trees}} \)) elementary trees.

**Definition.** The abstract language of an ACG \( \mathcal{G} = (\Sigma, \Xi, \mathcal{L}, s) \) is \( A(\mathcal{G}) = \{ t \in \Lambda(\Sigma) \mid t : \Sigma s \} \)

The object language of the grammar \( O(\mathcal{G}) = \{ t \in \Lambda(\Xi) \mid \exists u \in A(\mathcal{G}), t = \mathcal{L}(u) \} \)

For instance, the term \( C_{\text{reward}} I_S I_v C_{\text{Mary}} C_{\text{Jean}} : S \in \mathcal{G}_diced_\text{trees} \), and its image, the derived tree for \( \text{Marie récompense Jean} \) (Mary rewards John).

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. This allows for combining ACGs in different ways:
- by having a same abstract vocabulary shared by several ACGs: this can be used to make two object terms (for instance a string and a logical formula) share the same underlying structure. \( \mathcal{G}_diced_\text{trees} \) and \( \mathcal{G}_\log \) in Fig. 5 illustrate such a composition.
- by making the abstract vocabulary of one ACG the object vocabulary of another ACG, allowing for the control of the admissible structures of the former by the latter. \( \mathcal{G}_\text{field} \) and \( \mathcal{G}_diced_\text{trees} \) in Fig. 5 illustrate such a composition.
Crucial to our analysis is that ACG parsing of a term \( u \) amounts to finding an abstract term \( t \) such that \( t := u \), no matter whether \( u \) represents a string, a tree, or a logical formula. This can be done in polynomial time for ACGs whose abstract constant types are at most of order 2; second order ACGs as (Kanazawa, 2007) shows. The result relies on a reduction of the parsing problem to Data-log querying where the term to be parsed is stored in a database. Interestingly, this database can represent a set of terms (Kanazawa, 2011, Section 4.2) and the query reduces to checking whether at least one of them can be retrieved. This allows the query associated with a term representing a logical formula to extend to all the terms that are equivalent modulo the associativity and the commutativity of the conjunction.

5 ACG Encoding

5.1 TAG as ACG

Because ACG considers both the abstract language and the object language, the encoding of TAG into ACG makes (abstract) terms representing derivation trees primary. The encoding uses two ACGs \( \mathcal{G}_{d-\text{ed\-trees}} = (\Sigma_{\text{der}}, \Sigma_{\text{trees}}, \mathcal{L}_{d-\text{ed\-trees}}, S) \) and \( \mathcal{G}_{\text{yield}} = (\Sigma_{\text{trees}}, \Sigma_{\text{strings}}, \mathcal{L}_{\text{yield}}, \tau) \).

We exemplify the encoding\(^8\) of a TAG analyzing (2) in Fig. 6.\(^8\)

(2) Marie récompense ensuite Jean

Mary rewards then John

This sentence is usually analyzed in TAG with a derivation tree where the adverb adjoints at the \( \nu \) node.

The three higher-order signatures are:

\( \Sigma_{\text{der}} \): Its unique atomic type is \( \tau \) the type of trees. Then, for any \( X \) of arity \( n \) belonging to the ranked alphabet describing the elementary trees of the TAG, we have a constant \( X_n : \tau \rightarrow \cdots \rightarrow \tau \rightarrow \tau \)

\( \Sigma_{\text{trees}} \): Its unique atomic type is \( \sigma \) the type of trees. The constants are the terminal symbols of the TAG (with type \( \sigma \)), the empty string \( \varepsilon : \sigma \).

Table 1 illustrates \( \mathcal{L}_{d-\text{ed\-trees}} \). \( \mathcal{L}_{\text{yield}} \) is defined as follows:

- \( \mathcal{L}_{\text{yield}}(\tau) = \sigma \);
- for \( n > 0 \), \( \mathcal{L}_{\text{yield}}(X_n) = \lambda x_1 \cdots x_n \cdot x_1 + \cdots + x_n \);
- for \( n = 0 \), \( X_0 : \tau \) represents a terminal symbol and \( \mathcal{L}_{\text{yield}}(X_0) = X \).

Then, the derivation tree, the derived tree, and the yield of Fig. 6 are represented by:

![Derivation tree](image1)

![Derived tree](image2)

**Figure 6: Marie récompense ensuite Jean**

\( \gamma_5 = C_{\text{reward}} \ S \ C_\text{then} \ S \ C_\text{Marie} \ C_\text{Jean} \)

\( \mathcal{L}_{d-\text{ed\-trees}}(\gamma_5) = S_3 \ (np_1 \ Marie) \)

\( \mathcal{L}_{\text{yield}}(\mathcal{L}_{d-\text{ed\-trees}}(\gamma_5)) = Marie + \text{récompense} + \text{ensuite} + Jean \)

5.2 G-TAG as ACG

In order to model G-TAG in ACG, first we need to design the abstract signature \( \Sigma_{\text{g-der}} \) in which we can have entries for G-TAG. This entries will reflect the ideology that G-TAG is based on. For instance, in G-TAG discourse level words like ensuite can take as its arguments texts and sentences and produces text. In order to model this, we introduce types \( S \) and \( T \). Then, we can define \( D_{\text{then}} : S \rightarrow S \rightarrow T \), which means that \( D_{\text{then}}^{SS} \) has two arguments of type \( S \) and returns a result of type \( T \). As in G-TAG, ensuite can take two

\(^6\)It actually extends this result to almost linear object terms where variables with atomic type can be duplicated, as it commonly happens at the semantic level.

\(^7\)This corresponds to the systematic encoding of (Pogodalla, 2009) of TAG and its semantics into ACG.

\(^8\)We follow the grammar of (Abeillé, 2002).
texts as arguments and return text as well, we need to do have another entry for modeling this fact. This makes us to introduce another constant $D_{\text{then}}$: $T \rightarrow T \rightarrow T$. For the same kind of reason, we introduce following constants: $D_{\text{sp}}^T$, $S \rightarrow T \rightarrow T$, $D_{\text{sp}}^T$ and $T \rightarrow S \rightarrow T$. Other relations, like $\text{aparavant}$ is modeled in the same way as $\text{ensuite}$ in $\Sigma_{g-der}$. 

Apart from $\text{ensuite}$ and $\text{aparavant}$, there are connectives as $\text{avant}$ (before) and $\text{après}$ (after) that need to be modeled differently from $\text{ensuite}$. Indeed, while $\text{ensuite}$ results in a text, placing side by side a text and a sentence separated with a period, $\text{avant}$ and $\text{après}$ in French combine in a single sentence a (full) clause and an infinitive clause with an implicit subject: the one of the first clause. It is clear that in order to type $\text{avant}$ and $\text{après}$ in the $\Sigma_{g-der}$ signature, one should use a type which schematically looks as $\ldots \rightarrow \tau$. On the other hand, one needs to give the exact type to them. Despite that in TAG and G-TAG $\text{avant}$ and $\text{après}$ take two sentential arguments (labelled by $S$), the second argument bears a feature indicating it lacks the subject and that the latter has to be shared with the first sentence. For instance: $\text{Jean a fait une sieste après avoir passé l\'aspirateur}$ (John took a nap after having vacuumed), here the subject of $\text{avoir passé l\'aspirateur}$ (having vacuumed) is $\text{Jean}$, which comes from the sentence $\text{Jean a fait une sieste}$ (John took a nap). So, $\text{Jean a fait une sieste}$ (John took a nap) can be seen as a sentence whose subject is shared by another sentence as well. In order to model this fact, we use following type: $\text{Sws} \rightarrow \text{Sh} \rightarrow \text{np} \rightarrow \text{S}$. Indeed, the $\text{Sws}$ and the $\text{Sh}$ types correspond to the type of sentences missing a subject. Furthermore, we need to model $\text{pour}$ and $\text{pour que}$, which were introduced in order to lexicalize the $\text{goal}$ relation in G-TAG. First, let us have a look at $\text{pour que}$. It can take as its arguments two complete (from a syntax point of view) sentences and results in a sentence as in: $\text{Il travaille pour que vous puissiez manger}$. So, $D_{\text{pour que}}$, which is an entry corresponding to $\text{pour que}$, can be assigned a $S \rightarrow S \rightarrow S$ type. The syntactic difference between $\text{pour que}$ and $\text{pour}$ was highlighted in Section 3: $\text{pour}$ takes as arguments a complete sentence and an infinitive form of a sentence missing a subject whose subject comes from the first argument. Thus, in this case, similarly to case of $\text{avant}$ and $\text{après}$, $\text{pour}$ has to be modeled as an entry that has type $\text{Sws} \rightarrow \text{Sinf} \rightarrow \text{np} \rightarrow \text{S}$, where $\text{Sinf}$ stands for the type of an infinitive form of a clause missing a subject. We also need to deal with encoding different forms of a verb. For instance, récompenser has an active and a passive form. In G-TAG derivation, both of them can be encountered. In order to model this fact, two different entries are introduced: one for the passive form and one for the active form, which is the canonical construction for récompenser. So, we need to have two distinct entries $D_{\text{passive}}$ and $D_{\text{active}}$, and both of them have type $\text{S}_A \rightarrow \text{v}_A \rightarrow \text{np} \rightarrow \text{np} \rightarrow \text{S}$. Moreover, (Danlos, 2000) poses the problem that G-TAG cannot handle a text where the adverb adjoin at the $v$ node rather than on the $S$ node as in: $\text{Jean a passé l\'aspirateur. Il a [\text{ensuite}] fait une sieste}$ (John vacuumed. He [then] took a nap.) According to (Danlos, 2000) modelling such text production requires a formalism more powerful than TAG. In the ACG framework, this observations translates into defining an entry $D_{\text{then}}$: $S \rightarrow (\text{v}_A \rightarrow S) \rightarrow T$ in $\Sigma_{g-der}$ which is third order and that is, as such, beyond the TAC into ACG encoding (that only requires second-order types).\footnote{Currently, there is no theoretical complexity result for parsing such ACG fragments. However, in this particular, }
general mechanism for providing constants encoding adverbial connectives with two arguments as in discourse grammars such as D-STAG (Danlos, 2011), but contrary to D-LTAG where one of the arguments is anaphorically given from the preceding discourse (Webber, 2004).

**G-Derivation Trees to Derivation Trees** We translate terms of $\Sigma_{g\text{-der}}$, which correspond to g-derivation trees, into the TAG derivation tree language defined on $\Sigma_{der}$ using the lexicon $L_{der\text{-der}}$ of Table 2. It is interesting to see how to interpret:

$$L_{der\text{-der}}(S) = L_{der\text{-der}}(T) = L_{der\text{-der}}(Sws) = L_{der\text{-der}}(Sinf) = L_{der\text{-der}}(Sh) = S$$

$$L_{der\text{-der}}(S_A) = S_A$$

$$L_{der\text{-der}}(V_A) = v_A$$

$$L_{der\text{-der}}(np) = np$$

$$L_{der\text{-der}}(I_S) = I_S$$

$$L_{der\text{-der}}(I_v) = I_v$$

Table 2: The $L_{der\text{-der}}$ lexicon

pret $D^S_{then}: S \rightarrow (v_A \rightarrow S) \rightarrow T$ into $\Sigma_{der\theta}$. For this reason, we introduce in $\Sigma_{der\theta}$ the following constant: $\mathbb{S}_2 : S \rightarrow S \rightarrow S$ that allows for combining two sentences with a period. Now, it is possible to translate $D^S_{then}$ into $\Sigma_{der\theta}$ as follows:

$$L_{der\text{-der}}(D^S_{then}) = \lambda S_1 S_2. \mathbb{S}_2 (S_2 C^S_{then}).$$

It means that $D^S_{then}$ is interpreted as performing both the operation of combining two sentences with a period and the conjunction of *ensuite* on the $v$ node of the second sentence.

**G-Derived Trees as Interpretation of G-Derivation Trees** As soon as g-derivation trees as term built on $\Sigma_{g\text{-der}}$ are interpreted as term built on $\Sigma_{der\theta}$, we can map them to derived trees. Thus, by composing the two lexicons $L_{der\text{-der}}$ and $L_{d\text{-ed\ trees}}$ we can get directly from G-TAG into derived trees.

5.3 From G-TAG to Montague Style Semantics Using ACGs

(Pogodalla, 2009) defines a signature $\Sigma_{Log}$ and a lexicon $L_{Log}$ from $\Sigma_{der\theta}$ to $\Sigma_{Log}$. The entries in $\Sigma_{Log}$ have Montague like semantics. The lexicon translates a derivation tree into a corresponding formula. We will use the same kind of semantic language for conceptual representations. In other words, our language will produce the formulas that are used in the conceptual representation of G-TAG, while we will stick to the Montague style translations from syntax to semantics.

So, we define a signature $\Sigma_{concep}$ of conceptual representation that is similar to the one of (Pogodalla, 2009). $\Sigma_{concep}$ defines two atomic types $e$ and $t$ and constants such as: $j, m \ldots$ of type $e$, the constant *REWARD* of type $e \rightarrow e \rightarrow t$, the constant *CLAIM* of type $e \rightarrow t \rightarrow t$ and the constant *SEEM* of type $t \rightarrow t$. Moreover, we have constants *SUCCEP*, *GOAL* of type $t \rightarrow t \rightarrow t$.

We are able to translate $\Sigma_{g\text{-der\theta}}$ into $\Sigma_{concep}$ with the help of the lexicon $L_{der\text{-con}}$. The lexicon $L_{der\text{-con}}$ is extension of the lexicon defined in (Pogodalla, 2009), because we are adding to the domain (i.e. abstract language) the constants that are not in the $\Sigma_{der\theta}$.

$$L_{der\text{-con}}(S) = L_{der\text{-con}}(T) = t$$

$$L_{der\text{-con}}(V_A) = (e \rightarrow t)$$

$$L_{der\text{-con}}(S_A) = (e \rightarrow t)$$

$$L_{der\text{-con}}(np) = (e \rightarrow t)$$

$$L_{der\text{-con}}(D^S_{then}) = \lambda E. P. P(j)$$

$$L_{der\text{-con}}(D^S_{bat}) = \lambda E. S. S(a(E.a x. O(E.y).)$$

(REWARD x y)))

Note that the interpretation of *np* is $[np] = (e \rightarrow t)$, using a non-linear implication (but almost linear). Typically, the sharing of the subject by the two clauses related by *pour ou avant de* induces non linearity.

The *Sinf*, *Sh*, and *Sws* types all are interpreted as $[np] \rightarrow [S] = ((e \rightarrow t) \rightarrow t) \rightarrow t$ as they denote clauses lacking a subject. Then we translate the constants $D_{pour}$, $D_{apres}$, and $D_{avant}$ in the following way:

$$L_{der\text{-con}}(D_{pour}) = \lambda x. S_1. S_2. S_N. (\lambda x. (GOAL(S_1(\lambda P. P x)))$$

$$L_{der\text{-con}}(D_{apres}) = (s_2(\lambda P. P x)))$$

$$L_{der\text{-con}}(D_{avant}) = \lambda x. S_1. S_2. S_N. (\lambda x. (\text{SUCCEP}(s_1(\lambda P. P x)))$$

$$L_{der\text{-con}}(D_{avant}) = (s_2(\lambda P. P x)))$$
5.4 The G-TAG Process as a Morphism Composition

We exemplify the whole process using the term $T_0 = \text{Succ}(\text{Vac}(\text{Jean}), \text{Reward}(\text{Marie}, \text{Jean}))$ of type $t$. The terms representing the derivation trees that generate this conceptual representation are the antecedents of $T_0$ by $\mathcal{L}^{-1}_{\text{der-con}}$: $\mathcal{L}^{-1}_{\text{der-con}}(T_0) = \{t_1, \ldots, t_8\}$ that all are of type $T$. They are given in Figure 7. Each of these retrieved terms $t_1, \ldots, t_8$ are then mapped to terms representing TAG derivation trees, i.e. built on $\Sigma_{\text{derb}}$ via the lexicon $\mathcal{L}_{\text{der-ed}}$. They can be in turn be interpreted as syntactic derived trees via the lexicon $\mathcal{L}_{\text{yield}}$. So from $T_0$ we can have eight surface forms: $\mathcal{L}_{\text{yield}}(\mathcal{L}_{\text{der-ed}}(t_i)), i \in [1, 8]$. Let us show this process on the example of $t_5$. It illustrates the generation of the example (3).


$\mathcal{L}_{\text{der-con}}(D_{\text{Jean}}) = X_{s_1}X_{s_2}X_{N. N}(\lambda x.(\text{Succ}(s_2(\lambda P. P x)))) (s_1(\lambda P. P x)))$

$\mathcal{L}_{\text{der-con}}(D_{\text{marie}}(D_{\text{Jean}})) = D_{\text{reme}}(D_{\text{reward}}I_5I_4D_{\text{marie}}D_{\text{Jean}})$

$\mathcal{L}_{\text{der-con}}(D_{\text{marie}}(D_{\text{Jean}})) = D_{\text{reme}}(D_{\text{reward}}I_5I_4D_{\text{marie}}D_{\text{Jean}})$

6 Related Work

We can only quickly mention two related pieces of work. On the one hand, (Gardent and Perez-Beltrachini, 2010) also takes advantage of the formal properties underlying the tree language of derivation trees to propose a generation process using TAG grammars. On the other hand, (Nakatsu and White, 2010) also includes discourse relations in the grammar with Discourse Combinatory Categorical Grammar and a type-theoretical framework to provide a text (rather than sentence) generation process.

7 Conclusion

This paper shows how G-TAG can be encoded as ACG. It relies on the fact that both G-TAG and the encoding of TAG within ACG make the derivation tree a primary notion. Then we can benefit from the polynomial reversibility of the ACG framework. It also offers a generalization of the process to all kinds of adjunctions, including the predicative ones. It also offers a new insight on discourse grammars for the adverbial connective encoding (Danlos, 2011). Note that contrary to an important part of G-TAG that offers a way (based on a semantic and a linguistic analysis) to rank the different realizations of a conceptual representation, we do not deal here with such preferences. As syntactic ambiguity treatment is not usually part of the syntactic formalism, we prefer the “realization ambiguity” treatment not to be part of the generation formalism. Finally, a crucial perspective is to integrate a theory of generation of referring expressions relying on type-theoretical approaches to dynamics semantics (de Groote, 2006; de Groote and Lebedeva, 2010) that would ensure a large compatibility with the ACG framework.
References


