An ACG Analysis of the G-TAG Generation Process*

Laurence Danlos
Université Paris Diderot (Paris 7)
ALPAGE, INRIA Paris–Rocquencourt
Institut Universitaire de France, Paris, France
laurence.danlos@inria.fr

Aleksandre Maskharashvili and **Sylvain Pogodalla**
INRIA Villers-lès-Nancy, France
Université de Lorraine,
CNRS, LORIA, UMR 7503
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, 2004; Pogodalla, 2009)[1] based on Abstract Categorial Grammars (ACG) (de Groote, 2001). The latter is *intrinsically reversible*: the grammars and the algorithms are the same for parsing and for generation.

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,

*This work has been supported by the French agency Agence Nationale de la Recherche (ANR-12-CORD-0004).

[1] Synchronous 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 vacumed in order to be rewarded by Mary. Then he took a nap).

$$\begin{aligned}\texttt{SUCCESSION}(\texttt{GOAL}(&\texttt{VACUUMING}(\texttt{Jean}),\\ &\texttt{REWARDING}(\texttt{Marie},\texttt{Jean})),\\ &\qquad\texttt{NAPPING}(\texttt{Jean}))\end{aligned} \tag{1}$$

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: $(+\mathrm{T}, +\mathrm{S})$ indicates it is a text (two ore more sentences); $(+\mathrm{S})$ indicates it is either a single sentence or a text; $(-\mathrm{T}, +\mathrm{S})$ 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 ensuite$(+\mathrm{T},+\mathrm{S})$ belonging to the lexical database associated with the concept 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 GOAL relation that can be realized either by pour (in order to) or by pour que (so that), as exemplified by the rightmost g-derivation trees of Figure 1. Both have features $(-\mathrm{T},+\mathrm{S})$ for the inputs (i.e. arguments) and return a tree labeled at the root by $(-\mathrm{T},+\mathrm{S})$.

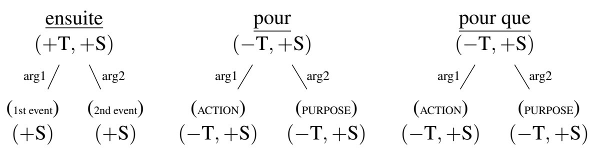


Figure 1: G-derivation trees samples

Despite pour and pour que bearing the same features, the syntactic trees corresponding to pour and pour que are quite different. For pour que S substitution nodes can be substituted by two tensed sentences, while 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 (+reduc-subj) feature as shown in Fig. 3. Again, we may assume that G-TAG selects pour,

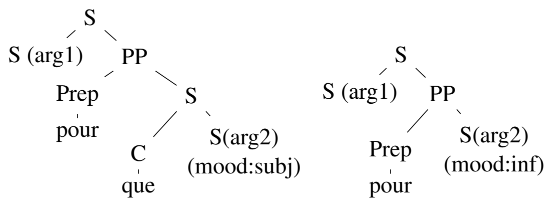


Figure 2: Elementary trees of *pour que* (so that) and *pour* (in order to)

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 Jean. From a semantic point of view, the agent of VACUUMING has to be the beneficiary of REWARDING (the rewardee). VACUUMING can only be lexicalized as passer-l’aspirateur (run-the-vacuum-cleaner), while there are several lexicalization options for the 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. s Assuming a choice of récompenser[+passive],[5] 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 VACUUMING and NAPPING. Therefore it can be either lexicalized in both of the cases as Jean, or Jean and the pronoun 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.)

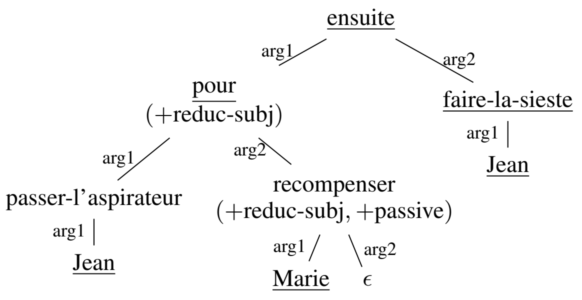


Figure 3: Fully instantiated g-derivation tree

4 ACG Definition

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

los, 2000).

[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.

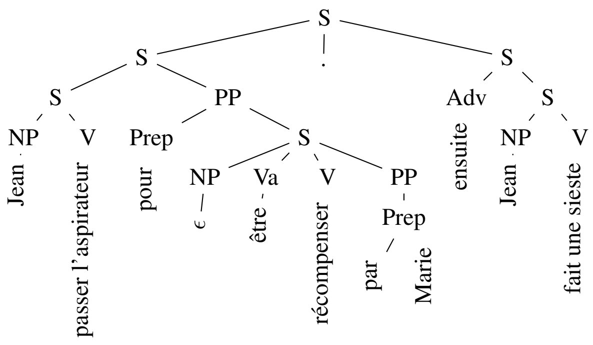


Figure 4: Non-inflected derived tree

retical framework that is able to encode several grammatical formalisms (de Groote 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* $\Sigma = \langle A, C, \tau\rangle$*, where:*

- *A is a finite set of atomic types (also noted A_Σ),*
- *C is a finite set of constants (also noted C_σ),*
- *and τ is a mapping from C to $\mathscr{T}_A$ the set of types built on A: $\mathscr{T}_A ::= A|\mathscr{T}_A \rightarrow \mathscr{T}_A$ (also noted $\mathscr{T}_\Sigma$).*

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

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

1. *Σ and Ξ are two higher-order linear signatures, which are called the* abstract vocabulary *and the* object vocabulary*, respectively;*
2. *$\mathscr{L} : \Sigma \longrightarrow \Xi$ is a lexicon from the abstract vocabulary to the object vocabulary. It is a homomorphism that maps types and terms built on Σ to types and terms built on Ξ as follows:*
   - *if $\alpha \rightarrow \beta \in \mathscr{T}_\Sigma$ then $\mathscr{L}(\alpha \rightarrow \beta) = \mathscr{L}(\alpha) \rightarrow \mathscr{L}(\beta)$*
   - *if $x \in \Lambda(\Sigma)$ (resp. $\lambda x.t \in \Lambda(\Sigma)$ and $t\,u \in \Lambda(\Sigma)$) then $\mathscr{L}(x) = x$ (resp. $\mathscr{L}(\lambda x.t) = \lambda x.\mathscr{L}(t)$ and $\mathscr{L}(t\,u) = \mathscr{L}(t)\,\mathscr{L}(u)$)*

   It is then enough to define $\mathscr{L}$ on the atomic types and on the constants of Σ to define it on all types and terms, provided that for any constant $c : \alpha$ of Σ we have $\mathscr{L}(c) : \mathscr{L}(\alpha)$. We note $t :=_\mathscr{G} u$ if $\mathscr{L}(t) = u$ and omit the $\mathscr{G}$ subscript if obvious from the context.
3. *$s \in \mathscr{T}_\Sigma$ is a type of the abstract vocabulary, which is called the* distinguished type *of the grammar.*

Table 1 provides an ACG example $\mathscr{G}_{\textit{d-ed trees}}$ where the abstract typed constants of $\Sigma_{der\theta}$ encode the combinatorial properties of the associated (through the lexicon $\mathscr{L}_{\textit{d-ed trees}}$) elementary trees.

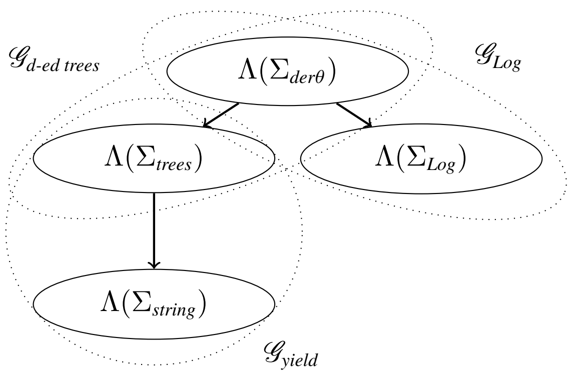
Definition. *The* abstract language *of an ACG* $\mathscr{G} = \langle \Sigma, \Xi, \mathscr{L}, s\rangle$ *is* $\mathcal{A}(\mathscr{G}) = \{t \in \Lambda(\Sigma) \mid t :_\Sigma s\}$

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

For instance, the term $C_{\textit{reward}}\ I_{\mathsf{S}}\ I_{\mathsf{v}}\ C_{\textit{Mary}}\ C_{\textit{Jean}} : \mathsf{S} \in \mathscr{G}_{\textit{d-ed trees}}$, and its image, the derived tree for *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. $\mathscr{G}_{\textit{d-ed trees}}$ and $\mathscr{G}_{\textit{Log}}$ 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. $\mathscr{G}_{\textit{yield}}$ and $\mathscr{G}_{\textit{d-ed trees}}$ in Fig. 5 illustrate such a composition.



$\mathscr{G}_{\textit{d-ed trees}}$
$\mathscr{G}_{\textit{Log}}$
$\Lambda(\Sigma_{der\theta})$
$\Lambda(\Sigma_{trees})$
$\Lambda(\Sigma_{Log})$
$\Lambda(\Sigma_{string})$
$\mathscr{G}_{\textit{yield}}$

Figure 5: ACG architecture for TAG

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.[6] The result relies on a reduction of the parsing problem to Datalog 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 $\mathscr{G}_{\textit{d-ed trees}} = \langle \Sigma_{\textit{der}\theta}, \Sigma_{\textit{trees}}, \mathscr{L}_{\textit{d-ed trees}}, \mathsf{S}\rangle$ and $\mathscr{G}_{\textit{yield}} = \langle \Sigma_{\textit{trees}}, \Sigma_{\textit{string}}, \mathscr{L}_{\textit{yield}}, \tau\rangle$.

We exemplify the encoding[7] 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 adjoins at the v node.

The three higher-order signatures are:

$\Sigma_{\textit{der}\theta}$: Its atomic types include S, v, np, S_A, v_A...where the X types stand for the categories X of the nodes where a substitution can occur while the X_A types stand for the categories X of the nodes where an adjunction can occur. For each elementary tree $\gamma_{\textsf{lex. entry}}$ it contains a constant $C_{\textsf{lex. entry}}$ whose type is based on the adjunction and substitution sites as Table 1 shows. It additionally contains constants $I_X : X_A$ that are meant to provide a fake auxiliary tree on adjunction sites where no adjunction actually takes place in a TAG derivation.

$\Sigma_{\textit{trees}}$: Its unique atomic type is τ 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

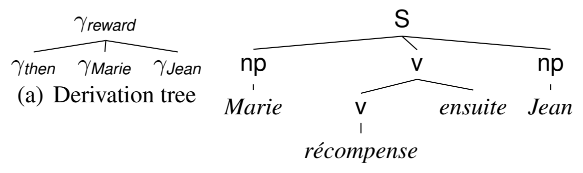
$$X_n : \overbrace{\tau \multimap \cdots \multimap \tau}^{n \text{ times}} \multimap \tau$$

$\Sigma_{\textit{string}}$: Its unique atomic type is σ the type of strings. The constants are the terminal symbols of the TAG (with type σ), the concatenation $+ : \sigma \multimap \sigma \multimap \sigma$ and the empty string $\varepsilon : \sigma$.

Table 1 illustrates $\mathscr{L}_{\textit{d-ed trees}}$.[9] $\mathscr{L}_{\textit{yield}}$ is defined as follows:

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

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



(a) Derivation tree (b) Derived tree

Figure 6: *Marie récompense ensuite Jean*

$$\gamma_5 = C_{\textsf{reward}}\ I_{\mathsf{S}}\ (C^{\mathsf{V}}_{\textsf{then}}\ I_{\mathsf{S}})\ C_{\textsf{Marie}}\ C_{\textsf{Jean}}$$

$$\begin{aligned}&\mathscr{L}_{\textit{d-ed trees}}(\gamma_5)\\ &\quad= \mathsf{S}_3\ (\mathsf{np}_1\ \textit{Marie})\\ &\quad(\mathsf{v}_2\ (\mathsf{v}_1\ \textit{récompense})\ \textit{ensuite})\ (\mathsf{np}_1\ \textit{Jean})\\ &\mathscr{L}_{\textit{yield}}(\mathscr{L}_{\textit{d-ed trees}}(\gamma_5)) = \textit{Marie} + \textit{récompense}\\ &\qquad\qquad + \textit{ensuite} + \textit{Jean}\end{aligned}$$

5.2 G-TAG as ACG

In order to model G-TAG in ACG, first we need to design the abstract signature $\Sigma_{\textit{g-der}\theta}$ 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 $D^{SS}_{\textsf{then}}$: $\mathsf{S} \multimap \mathsf{S} \multimap \mathsf{T}$, which means that $D^{SS}_{\textsf{then}}$ has takes 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).

[9] With $\mathscr{L}_{\textit{d-ed trees}}(X_A) = \tau \multimap \tau$ and for any other type X, $\mathscr{L}_{\textit{d-ed trees}}(X_A) = \tau$.

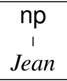
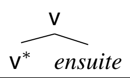
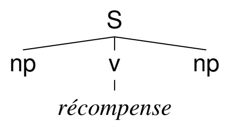
Abstract constants of $\Sigma_{der\theta}$	Their images by $\mathscr{L}_{\textit{d-ed trees}}$	The corresponding TAG trees
$C_{\mathit{Jean}} : \mathsf{np}$	$c_{\mathit{Jean}} \begin{array}{l} : \tau \\ = \mathsf{np}_1\ \mathit{Jean} \end{array}$	$\gamma_{\mathit{Jean}} =$ 
$C^{\mathsf{v}}_{\mathit{then}} : \mathsf{v}_A \multimap \mathsf{v}_A$	$c^{\mathsf{v}}_{\mathit{then}} \begin{array}{l} : (\tau \multimap \tau) \multimap (\tau \multimap \tau) \\ = \lambda^o vx.v\ (\mathsf{v}_2\ x \mathit{ensuite}) \end{array}$	$\gamma_{\mathit{then}} =$ 
$C_{\mathit{reward}} : \begin{array}{c} \mathsf{S}_A \multimap \mathsf{v}_A \multimap \mathsf{np} \\ \multimap \mathsf{np} \multimap \mathsf{S} \end{array}$	$c_{\mathit{reward}} \begin{array}{l} : \begin{array}{l} (\tau \multimap \tau) \multimap (\tau \multimap \tau) \multimap \tau \\ \multimap \tau \multimap \tau \end{array} \\ = \lambda^o avso.a\ (\mathsf{S}_3\ s\ (v\ (\mathsf{v}_1\ \textit{récompense}))\ o) \end{array}$	$\gamma_{\mathit{reward}} =$ 
$I_X : X_A$	$\lambda x.x : \tau \multimap \tau$	

Table 1: A TAG as an ACG: $\mathscr{L}_{\textit{d-ed trees}}$ and $\mathscr{L}_{\textit{log.sem}}$ lexicons

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^{TT}_{then}: $\mathsf{T} \multimap \mathsf{T} \multimap \mathsf{T}$. For the same kind of reason, we introduce following constants: D^{ST}_{then}: $\mathsf{S} \multimap \mathsf{T} \multimap \mathsf{T}$, D^{TS}_{then} and $\mathsf{T} \multimap \mathsf{S} \multimap \mathsf{T}$. Other relations, like *auparavant* is modeled in the same way as *ensuite* in $\Sigma_{\textit{g-der}\theta}$.

Apart from *ensuite* and *auparavant*, there are connectives as *avant* (before) and *après* (after) that need to be modeled differently from *ensuite*. Indeed, while *ensuite* results in a text, placing side by side a text and a sentence separated with a period, *avant* and *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 *avant* and *après* in the $\Sigma_{\textit{g-der}\theta}$ signature, one should use a type which schematically looks as $\ldots \multimap \mathsf{S}$. On the other hand, one needs to give the exact type to them. Despite that in TAG and G-TAG *avant* and *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: *Jean a fait une sieste après avoir passé l'aspirateur* (John took a nap after having vacuumed), here the subject of *avoir passé l'aspirateur* (having vacuumed) is *Jean*, which comes from the sentence Jean a fait une sieste (John took a nap). So, *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 point, we use following type: $\mathsf{Sws} \multimap \mathsf{Sh} \multimap \mathsf{np} \multimap \mathsf{S}$. Indeed, the Sws and the Sh types correspond to the type of sentences missing a subject. Furthermore, we need to model *pour* and *pour que*, which were introduced in order to lexicalize the GOAL relation in G-TAG. First, let us have a look at *pour que*. It can take as its arguments two complete (from a syntax point of view) sentences and results in a sentence as in: *Il travaille pour que vous puissiez manger*. So, $D_{\mathit{pour\ que}}$, which is an entry corresponding to *pour que*, can be assigned a $\mathsf{S} \multimap \mathsf{S} \multimap \mathsf{S}$ type. The syntactic difference between *pour que* and *pour* was highlighted in Section 3: *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 *avant* and *après*, *pour* has to be modeled as an entry that has type $\mathsf{Sws} \multimap \mathsf{Sinf} \multimap \mathsf{np} \multimap \mathsf{S}$, where 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, $\overline{\text{récompenser}}$ 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 $\overline{\text{récompenser}}$. So, we need to have two distinct entries $D^{\mathit{passive}}_{\mathit{recompense}}$ and $D^{\mathit{active}}_{\mathit{recompense}}$, and both of them have type $\mathsf{S}_A \multimap \mathsf{v}_A \multimap \mathsf{np} \multimap \mathsf{np} \multimap \mathsf{S}$. 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: *Jean a passé l'aspirateur. Il a* $\boxed{\textit{ensuite}}$ *fait une sieste* (John vacuumed. He $\boxed{\textit{then}}$ 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 $D^{\mathsf{v}}_{\mathit{then}} : \mathsf{S} \multimap (\mathsf{v}_A \multimap \mathsf{S}) \multimap \mathsf{T}$ in $\Sigma_{\textit{g-der}\theta}$ which is third order and that is, as such, beyond the TAC into ACG encoding (that only requires second-order types).[10] This also offers a

[10]Currently, there is no theoretical complexity result for parsing such ACG fragments. However, in this particu-

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 $\Sigma_{g\text{-}der\theta}$, which correspond to g-derivation trees, into the TAG derivation tree language defined on $\Sigma_{der\theta}$ using the lexicon $\mathscr{L}_{der\text{-}der}$ of Table 2. It is interesting to see how to interpret D^{V}_{then}: $\mathsf{S} \multimap (\mathsf{v}_A \multimap \mathsf{S}) \multimap \mathsf{T}$ into $\Sigma_{der\theta}$. For this reason, we introduce in $\Sigma_{der\theta}$ the following constant: $\circledS_2 : \mathsf{S} \multimap \mathsf{S} \multimap \mathsf{S}$ that allows for combining two sentences with a period. Now, it is possible to translate D^{V}_{then} into $\Sigma_{der\theta}$ as follows: $\mathscr{L}_{der\text{-}der}(D^{\mathsf{V}}_{then}) = \lambda^{o} S_1\, S_2.\circledS_2\ S_1 (S_2\, C^{\mathsf{V}}_{then})$. It means that D^{V}_{then} is interpreted as performing both the operation of combining two sentences with a period and the adjunction of *ensuite* on the v node of the second sentence.

$$
\begin{aligned}
\mathscr{L}_{der\text{-}der}(\mathsf{S}) &= \mathscr{L}_{der\text{-}der}(\mathsf{T}) &&= \mathscr{L}_{der\text{-}der}(\mathsf{Sws})\\
&= \mathscr{L}_{der\text{-}der}(\mathsf{Sinf}) &&= \mathscr{L}_{der\text{-}der}(\mathsf{Sh})\\
&= \mathsf{S}\\
\mathscr{L}_{der\text{-}der}(\mathsf{S}_A) &= \mathsf{S}_A\\
\mathscr{L}_{der\text{-}der}(\mathsf{v}_A) &= \mathsf{v}_A\\
\mathscr{L}_{der\text{-}der}(\mathsf{np}) &= \mathsf{np}\\
\mathscr{L}_{der\text{-}der}(I_{\mathsf{S}}) &= I_{\mathsf{S}}\\
\mathscr{L}_{der\text{-}der}(I_{\mathsf{v}}) &= I_{\mathsf{v}}
\end{aligned}
$$

Table 2: The $\mathscr{L}_{der\text{-}der}$ lexicon

G-Derived Trees as Interpretation of G-Derivation Trees As soon as g-derivation trees as term built on $\Sigma_{g\text{-}der\theta}$ are interpreted as term built on $\Sigma_{der\theta}$, we can map them to derived trees. Thus, by composing the two lexicons $\mathscr{L}_{der\text{-}der}$ and $\mathscr{L}_{d\text{-}ed\ trees}$ 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 Σ_{Log} and a lexicon $\mathscr{L}_{Log}$ from $\Sigma_{der\theta}$ to Σ_{Log}. The entries in Σ_{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 Σ_{conrep} of conceptual representation that is similar to the one of (Pogodalla, 2009). Σ_{conrep} defines two atomic types e and t and constants such as: **j**, **m** . . . of type e, the constant **REWARD** of type $e \multimap e \multimap t$, the constant **CLAIM** of type $e \multimap t \multimap t$ and the constant **SEEM** of type $t \multimap t$. Moreover, we have constants **SUCC**, **GOAL** of type $t \multimap t \multimap t$.

We are able to translate $\Sigma_{g\text{-}der\theta}$ into Σ_{conrep} with the help of the lexicon $\mathscr{L}_{der\text{-}con}$. The lexicon $\mathscr{L}_{der\text{-}con}$ 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 $\Sigma_{der\theta}$.

$$
\begin{aligned}
\mathscr{L}_{der\text{-}con}(\mathsf{S}) &= \mathscr{L}_{der\text{-}con}(\mathsf{T}) = t\\
\mathscr{L}_{der\text{-}con}(\mathsf{v}_A) &= (e \to t) \multimap (e \to t)\\
\mathscr{L}_{der\text{-}con}(\mathsf{S}_A) &= t \multimap t\\
\mathscr{L}_{der\text{-}con}(\mathsf{np}) &= (e \to t) \multimap t\\
\mathscr{L}_{der\text{-}con}(D_{jean}) &= \lambda^{o} P.P(\mathbf{j})\\
\mathscr{L}_{der\text{-}con}(D^{ST}_{then}) &= \mathscr{L}_{der\text{-}con}(D^{SS}_{then})\\
&= \mathscr{L}_{der\text{-}con}(D^{ST}_{then})\\
&= \mathscr{L}_{der\text{-}con}(D^{TS}_{then})\\
&= \mathscr{L}_{der\text{-}con}(D^{TT}_{then})\\
&= \lambda s_2 s_1.\mathbf{SUCC}\ s_2\ s_1\\
\mathscr{L}_{der\text{-}con}(D^{ST}_{bef.}) &= \mathscr{L}_{der\text{-}con}(D^{SS}_{bef.})\\
&= \mathscr{L}_{der\text{-}con}(D^{ST}_{bef.})\\
&= \mathscr{L}_{der\text{-}con}(D^{TS}_{bef.})\\
&= \mathscr{L}_{der\text{-}con}(D^{TT}_{bef.})\\
&= \lambda^{o}\ s_1 s_2.\ \mathbf{SUCC}\ s_2\ s_1\\
\mathscr{L}_{der\text{-}con}(D_{rewards}) &= \lambda^{o} s\ a\ O\ S.s(S(a(\lambda^{o} x.O(\lambda^{o} y.\\
&\qquad (\mathbf{REWARD}\ x\ y))))
\end{aligned}
$$

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

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

$$
\begin{aligned}
&\mathscr{L}_{der\text{-}con}(D_{pour}) =\\
&\quad \lambda^{o} s_1.\lambda^{o} s_2.\lambda^{o} N.N(\lambda x.(\mathbf{GOAL}(s_1(\lambda P.P\ x))\\
&\qquad\qquad (s_2(\lambda P.P\ x))))
\end{aligned}
$$

$$
\begin{aligned}
&\mathscr{L}_{der\text{-}con}(D_{apres}) =\\
&\quad \lambda^{o} s_1.\lambda^{o} s_2.\lambda^{o} N.N(\lambda x.(\mathbf{SUCC}(s_1(\lambda P.P\ x))\\
&\qquad\qquad (s_2(\lambda P.P\ x))))
\end{aligned}
$$

lar case, we could use a second-order—and polynomial—encoding of multi-component TAG into ACG.

$$\mathscr{L}_{\textit{der-con}}(D_{\textsf{avant}}) = \lambda^{o} s_1.\lambda^{o} s_2.\lambda^{o} N.N(\lambda x.(\mathbf{SUCC}(s_2(\lambda P.P\ x))\ (s_1(\lambda P.P\ x))))$$

5.4 The G-TAG Process as a Morphism Composition

We exemplify the whole process using the term $T_0 = \mathbf{SUCC}(\mathbf{VAC}(\mathbf{jean}), \mathbf{REWARD}(\mathbf{marie}, \mathbf{jean}))$ of type t.[11] The terms representing the g-derivation trees that generate this conceptual representation are the antecedents of T_o by $\mathscr{L}^{-1}_{\textit{der-con}}$: $\mathscr{L}^{-1}_{\textit{der-con}}(T_0) = \{t_1, \ldots, t_8\}$ 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 $\Sigma_{\textit{der}\theta}$ via the lexicon $\mathscr{L}_{\textit{der-der}}$. They can be can in turn be interpreted as syntactic derived trees via the lexicon $\mathscr{L}_{\textit{d-ed trees}}$, and the latter can be interpreted as strings using the lexicon $\mathscr{L}_{\textit{yield}}$. So from T_0 we can have eight surface forms: $\mathscr{L}_{\textit{yield}}(\mathscr{L}_{\textit{d-ed trees}}(\mathscr{L}_{\textit{der-der}}(t_i))), i \in [1, 8]$. Let us show this process on the example of t_5[12]. It illustrates the generation of the example (3).[13]

(3) *Jean* *a passé l'aspirateur.* *Marie*
John vacuumed. Mary
a récompensé *ensuite* *Jean.*
rewarded afterwards John.

$$\mathscr{L}_{\textit{der-der}}(t_5) = \circledS_2\ (C_{\textsf{vac}} I_{\mathsf{S}} I_{\mathsf{v}} C_{\textsf{jean}})\ (C_{\textsf{reward}} I_{\mathsf{S}} C^{\mathsf{v}}_{\textsf{then}} C_{\textsf{marie}} C_{\textsf{jean}})$$

$$\mathscr{L}_{\textit{d-ed trees}}(\mathscr{L}_{\textit{der-der}}(t_5) = \mathsf{S}_3\ (\mathsf{S}_2\ (\mathsf{np}_1\ \textit{Jean})(\mathsf{v}_1\ \textit{a passé l'aspirateur}))\ .\ (\mathsf{S}_3(\mathsf{np}_1\ \textit{Marie})(\mathsf{v}_2\ (\mathsf{v}_1\ \textit{a récompensé})\ \textit{ensuite})(\mathsf{np}_1\ \textit{Jean}))$$

And the surface forms is given by composing the interpretations:

$$\mathscr{L}_{\textit{yield}}(\mathscr{L}_{\textit{d-ed trees}}(\mathscr{L}_{\textit{der-der}}(t_5)) = \textit{Jean} + \textit{a passé} + \textit{l'aspirateur} + . + \textit{Marie} + \textit{a recompensé} + \textit{ensuite} + \textit{Jean}$$

[11]The associated conceptual input is a simplified version of the conceptual input of Equation 1 without the GOAL concept and a replacement of the NAP one by the REWARDING one.

[12]t_5 is such that $\mathscr{L}_{\textit{der-der}}(t_5) = \gamma_5$ and the term γ_5 was used as example at Section 5.1.

[13]For sake of simplicity we assume the adverb adjoins on the whole auxiliary+verb phrase rather than only on the auxiliary as it would be in French.

$$\begin{aligned}
t_1 &= D^{SS}_{\textsf{then}}(D_{\textsf{vac}} I_{\mathsf{S}} I_{\mathsf{v}} D_{\textsf{jean}})(D_{\textsf{reward}} I_{\mathsf{S}} I_{\mathsf{v}} D_{\textsf{marie}} D_{\textsf{jean}})\\
t_2 &= D^{SS}_{\textsf{then}}(D_{\textsf{vac}} I_{\mathsf{S}} I_{\mathsf{v}} D_{\textsf{jean}})(D^{\text{passive}}_{\textsf{reward}} I_{\mathsf{S}} I_{\mathsf{v}} D_{\textsf{marie}} D_{\textsf{jean}})\\
t_3 &= D^{SS}_{\textsf{bef.}}(D_{\textsf{reward}} I_{\mathsf{S}} I_{\mathsf{v}} D_{\textsf{marie}} D_{\textsf{jean}})(D_{\textsf{vac}} I_{\mathsf{S}} I_{\mathsf{v}} D_{\textsf{jean}})\\
t_4 &= D^{SS}_{\textsf{bef.}}(D^{\text{passive}}_{\textsf{reward}} I_{\mathsf{S}} I_{\mathsf{v}} D_{\textsf{jean}} D_{\textsf{marie}})(D_{\textsf{vac}} I_{\mathsf{S}} I_{\mathsf{v}} D_{\textsf{jean}})\\
t_5 &= D^{\mathsf{v}}_{\textsf{then}}(D_{\textsf{vac}} I_{\mathsf{S}} I_{\mathsf{v}} D_{\textsf{jean}})(\lambda^{o} a.D_{\textsf{reward}}\ I_{\mathsf{S}}\ a\ D_{\textsf{marie}} D_{\textsf{jean}})\\
t_6 &= D^{\mathsf{v}}_{\textsf{then}}(D_{\textsf{vac}} I_{\mathsf{S}} I_{\mathsf{v}} D_{\textsf{jean}})(D^{\text{passive}}_{\textsf{reward}} I_{\mathsf{S}} I_{\mathsf{v}} D_{\textsf{jean}} D_{\textsf{marie}})\\
t_7 &= D_{\textsf{after}}(D^{\text{sws}}_{\textsf{vac}} I_{\mathsf{S}} I_{\mathsf{v}})(D_{\textsf{receive-rew.}} I_{\mathsf{S}} I_{\mathsf{v}} D_{\textsf{jean}}) D_{\textsf{marie}}\\
t_8 &= D_{\textsf{bef.}}(D^{\text{sws}}_{\textsf{vac}} I_{\mathsf{S}} I_{\mathsf{v}})(D_{\textsf{receive-rew.}} I_{\mathsf{S}} I_{\mathsf{v}} D_{\textsf{jean}}) D_{\textsf{marie}}
\end{aligned}$$

Figure 7: Antecedents of T_0 by $\mathscr{L}_{\textit{der-con}}$

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 Categorial 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

[Abeillé2002] Anne Abeillé. 2002. *Une grammaire électronique du français*. Sciences du langage. CNRS Éditions.

[Danlos et al.2001] Laurence Danlos, Bertrand Gaiffe, and Laurent Roussarie. 2001. Document sructuring à la SDRT. In Helmut Horacek, Nicolas Nicolov, and Leo Wanner, editors, *Proceedings of the ACL 2001 Eighth European Workshop on Natural Language Generation (EWNLG)*. http://aclweb.org/anthology/W/W01/W01-0803.pdf.

[Danlos et al.2011] Laurence Danlos, Frédéric Meunier, and Vanessa Combet. 2011. EasyText: an operational NLG system. In *ENLG 2011, 13th European Workshop on Natural Language Generation*, September. http://hal.inria.fr/inria-00614760/en/.

[Danlos1998] Laurence Danlos. 1998. G-TAG : Un formalisme lexicalisé pour la génération de textes inspiré de TAG. *Traitement Automatique des Langues*, 39(2). http://hal.inria.fr/inria-00098489.

[Danlos2000] Laurence Danlos. 2000. G-TAG: A lexicalized formalism for text generation inspired by tree adjoining grammar. In Anne Abeillé and Owen Rambow, editors, *Tree Adjoining Grammars: Formalisms, Linguistic Analysis, and Processing*, pages 343–370. CSLI Publications.

[Danlos2011] Laurence Danlos. 2011. D-STAG: a formalism for discourse analysis based on SDRT and using synchronous TAG. In Philippe de Groote, Markus Egg, and Laura Kallmeyer, editors, *14th conference on Formal Grammar - FG 2009*, volume 5591 of *LNCS/LNAI*, pages 64–84. Springer. http://dx.doi.org/10.1007/978-3-642-20169-1_5.

[de Groote and Lebedeva2010] Philippe de Groote and Ekaterina Lebedeva. 2010. Presupposition accommodation as exception handling. In *Proceedings of the SIGDIAL 2010 Conference*, pages 71–74, Tokyo, Japan, September. Association for Computational Linguistics. http://www.aclweb.org/anthology/W/W10/W10-4313.

[de Groote and Pogodalla2004] Philippe de Groote and Sylvain Pogodalla. 2004. On the expressive power of Abstract Categorial Grammars: Representing context-free formalisms. *Journal of Logic, Language and Information*, 13(4):421–438. http://hal.inria.fr/inria-00112956.

[de Groote2001] Philippe de Groote. 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. http://aclweb.org/anthology/P/P01/P01-1033.pdf.

[de Groote2006] Philippe de Groote. 2006. Towards a montagovian account of dynamics. In Masayuki Gibson and Jonathan Howell, editors, *Proceedings of Semantics and Linguistic Theory (SALT) 16*. http://elanguage.net/journals/index.php/salt/article/view/16.1/1791.

[Gardent and Perez-Beltrachini2010] Claire Gardent and Laura Perez-Beltrachini. 2010. RTG based surface realisation for TAG. In *Proceedings of the 23rd International Conference on Computational Linguistics (COLING 2010)*, pages 367–375, Beijing, China, August. Coling 2010 Organizing Committee. http://www.aclweb.org/anthology/C10-1042.

[Joshi and Schabes1997] Aravind K. Joshi and Yves Schabes. 1997. Tree-adjoining grammars. In G. Rozenberg and A. Salomaa, editors, *Handbook of formal languages*, volume 3, chapter 2. Springer.

[Joshi et al.1975] Aravind K. Joshi, Leon S. Levy, and Masako Takahashi. 1975. Tree adjunct grammars. *Journal of Computer and System Sciences*, 10(1):136–163.

[Kallmeyer and Romero2004] Laura Kallmeyer and Maribel Romero. 2004. LTAG semantics with semantic unification. In *Proceedings of TAG+7*, pages 155–162.

[Kallmeyer and Romero2007] Laura Kallmeyer and Maribel Romero. 2007. Scope and situation binding for LTAG. *Research on Language and Computation*, 6(1):3–52. http://dx.doi.org/10.1007/s11168-008-9046-6.

[Kanazawa2007] Makoto Kanazawa. 2007. Parsing and generation as datalog queries. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics (ACL)*, pages 176–183. http://www.aclweb.org/anthology/P/P07/P07-1023.

[Kanazawa2011] Makoto Kanazawa, 2011. *Parsing and generation as Datalog query evaluation*. Under review. http://research.nii.ac.jp/~kanazawa/publications/pagadqe.pdf.

[Meunier1997] Frédéric Meunier. 1997. *Implantation du formalisme de génération G-TAG*. Ph.D. thesis, Université Paris 7 — Denis Diderot.

[Nakatsu and White2010] Crytal Nakatsu and Michael White. 2010. Generating with discourse combinatory categorial grammar. *Linguistic Issues in Language Technology*, 4(1). http://elanguage.net/journals/index.php/lilt/article/view/1277/871.

[Nesson and Shieber2006] Rebecca Nesson and Stuart M. Shieber. 2006. Simpler TAG semantics through synchronization. In *Proceedings of the 11th Conference on Formal Grammar*, Malaga, Spain, 29–30 July. CSLI Publications.

http://cslipublications.stanford.edu/FG/2006/nesson.pdf.

[Pogodalla2004] Sylvain Pogodalla. 2004. Computing Semantic Representation: Towards ACG Abstract Terms as Derivation Trees. In *Proceedings of TAG+7*, pages 64–71. http://hal.inria.fr/inria-00107768.

[Pogodalla2009] Sylvain Pogodalla. 2009. Advances in Abstract Categorial Grammars: Language Theory and Linguistic Modeling. ESSLLI 2009 Lecture Notes, Part II. http://hal.inria.fr/hal-00749297.

[Schabes and Shieber1994] Yves Schabes and Stuart M. Shieber. 1994. An alternative conception of tree-adjoining derivation. *Computational Linguistics*, 20(1):91–124. http://aclweb.org/anthology/J/J94/J94-1004.pdf.

[Storoshenk and Frank2012] Dennis Ryan Storoshenk and Robert Frank. 2012. Deriving syntax-semantics mappings: node linking, type shifting and scope ambiguity. In *Proceedings of TAG+11*, pages 10–18.

[Webber2004] Bonnie Webber. 2004. D-LTAG: Extending :exicalized TAG to discourse. *Cognitive Science*, 28:751–779. http://dx.doi.org/0.1207/s15516709cog2805_6.