HAL
open science

A syntax-semantics interface for Tree-Adjoining Grammars through Abstract Categorial Grammars

Sylvain Pogodalla

To cite this version:

Sylvain Pogodalla. A syntax-semantics interface for Tree-Adjoining Grammars through Abstract Categorial Grammars. Journal of Language Modelling, 2017, 5 (3), pp.527-605. 10.15398/jlm.v5i3.193 . hal-01242154v4

HAL Id: hal-01242154

https://inria.hal.science/hal-01242154v4

Submitted on 12 Dec 2017 (v4), last revised 5 Apr 2018 (v5)

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.

A syntax-semantics interface for Tree-Adjoining Grammars through Abstract Categorial Grammar

Sylvain Pogodalla
INRIA, Villers-lès-Nancy, France
Université de Lorraine, LORIA, Vandœuvre-lès-Nancy, France
CNRS, LORIA, Vandœuvre-lès-Nancy, France

ABSTRACT

We present a model of the syntax-semantics interface for Tree-Adjoining Grammars (TAGs). It is based on the encoding of TAGs within the framework of Abstract Categorial Grammars (ACGs). This encoding relies on a functional interpretation of the substitution and adjunction operations of TAGs. In ACGs, the abstract terms representing derivation trees are full-fledged objects of the grammar. These terms are mapped onto logical formulas representing the semantic interpretation of natural expressions that TAGs can analyze. Because of the reversibility properties of ACGs, this provides a way to parse and generate with the same TAG encoded grammar. We propose several analyses, including long distance dependencies, quantification, control and raising verbs, and subordinate clauses. We also show how this encoding easily extends to other phenomena such as idioms or scope ambiguities. All the lexical data for theses modellings are provided and can be run with the ACG toolkit, a software package dedicated to the development of ACGs that can use these grammars both for parsing and generation.

Keywords: Tree-adjoining grammars, syntax-semantics interface, abstract categorial grammars

1 MOTIVATIONS

1.1 *Tree-Adjoining Grammar and semantic representation*

The Tree-Adjoining Grammar (TAG) formalism (Joshi *et al.* 1975; Joshi and Schabes 1997) is a formalism dedicated to the modelling of natural languages. As the name indicates, the primary objects it

considers are trees rather than strings, as context-free grammars, for instance, do. As such, the object language a TAG generates is a tree language, the language of the *derived trees*. These trees result from the application of two operations, *substitution* and *adjunction*, to a set of generators: the *elementary* trees. The substitution operation consists in replacing the leaf of a tree by another tree, while the adjunction operation consists in inserting a tree into another one by replacing an internal node with a whole tree. A sequence of such operations and the elementary trees they operate on can be recorded as a *derivation* tree. Reading the leaves of the derived tree, or computing the *yield*, produces the associated generated string language.

The class of the generated string languages strictly includes the one generated by context-free grammars. This property, together with other ones such as the crucial polynomial parsing property, plays an important role in the characterization of the expressive power that natural language modelling requires. Joshi (1985) proposed to call the class of languages (resp. grammars) necessary for describing natural languages the class of *mildly context sensitive languages* or mCSL (resp. *mildly context sensitive grammars* or mCSG). These formal and computational properties have been extensively studied[1] and provide TAG with appealing features for natural language processing. In addition to its formal properties, TAG has also been studied both from a perspective of fine-grained modellings of a wide range of linguistic phenomena, and from a perspective of large coverage. Large scale grammars have been developed for several languages, including English (XTAG Research Group 2001) or French (Abeillé 2002; Crabbé 2005; de La Clergerie 2005). In addition to these hand-crafted grammars, automatic extraction of TAGs have also been proposed (Xia *et al.* 2000; Xia 2001; Chen *et al.* 2006).

Another key feature that makes TAG relevant to natural language modelling lies in the capability of its elementary trees to locally specify (syntactic and semantic) dependencies between parts that can occur arbitrarily far from each other at the surface level at the end of a derivation. This property to locally state, within the elementary trees, dependency constraints is also known as the *extended domain of local-*

[1] See for instance (Vijay-Shanker and Joshi 1985; Vijay-Shanker 1987; Weir 1988; Kuhlmann and Möhl 2007; Kanazawa 2008b,a; Kallmeyer 2010).

ity (Joshi 1994). Thanks to the adjunction operation, a dependency described locally in an elementary tree can end as a long-distance dependency in the resulting derived tree. The relevant structure to store the relations between the elementary trees that are used in a derivation is then the derivation tree. This makes the latter structure appropriate to derive semantic representations for TAGs.

It was however noticed that derivation trees do not directly express the semantic dependencies and that they seem to lack some structural information (Vijay-Shanker 1992; Candito and Kahane 1998). To overcome this problem, several approaches have been proposed. Some rely on extensions of the TAG formalism (Rambow *et al.* 1995, 2001), some others revisit the derivation tree definition (Schabes and Shieber 1994; Shieber 1994; Kallmeyer 2002; Joshi *et al.* 2003; Kallmeyer and Joshi 2003) in order to allow for recovering all the semantic dependency relations. Then, solutions to the problem strictly relying on derivation trees have been proposed. They make use of unification (Kallmeyer and Romero 2004, 2008), functional tree interpretation (Pogodalla 2004a, 2009), or synchronous grammars (Nesson and Shieber 2006; Nesson 2009) and tree transduction (Shieber 2006; Kallmeyer and Kuhlmann 2012; Shieber 2014).

1.2 *TAG and Abstract Categorial Grammars: our approach*

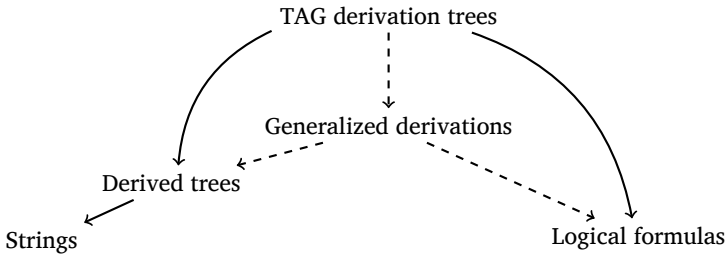
In this article, we elaborate on (Pogodalla 2004a, 2009) in order to propose a syntax-semantics interface and a semantic construction process for TAGs. We base our analysis on the framework of Abstract Categorial Grammars (ACGs, de Groote 2001). ACGs derive from type-theoretic grammars in the tradition of Lambek (1958), Curry (1961), and Montague (1973). They can be considered as a framework in which several grammatical formalisms may be encoded (de Groote and Pogodalla 2004), in particular TAGs (de Groote 2002). The definition of an ACG is based on a small set of mathematical primitives from type-theory, λ-calculus, and linear logic. These primitives combine via simple composition rules, offering ACGs a good flexibility. In particular, ACGs generate languages of linear λ-terms, which generalize both string and tree languages.

But ACGs are not restricted to languages of λ-terms encoding strings or trees. They can express logic-based semantic representation languages. And moving from one kind to another kind of language is

realized by composing ACGs. We take advantage of the different composition modes to control the admissible derivation structures on the one hand, and to model the syntax-semantics interface on the other hand.

The core contribution of this article is to show that ACGs offer a suitable model of the syntax-semantics interface for TAG. By construction, this model is fully compositional and satisfies the homomorphic requirement between parse structures (terms representing derivation trees) and semantic terms. It relies on an encoding of TAGs into ACGs. For a given TAG G, with this encoding, we can construct and relate several ACGs that generate the same string language, derived tree language, and derivation tree language as G. By ACG composition, this encoding is the same as the one proposed by de Groote (2002) (that only addresses the syntactic encoding of TAG into ACG, not the syntax-semantics interface), which ensures the correctness of the (syntactic) encoding. This encoding corresponds to the path with solid lines from *TAG derivation trees* to *Strings* in Figure 1. But we introduce an intermediate level, of generalized derivations, on which we base our syntax-semantics interface (the dashed lines in Figure 1). Doing so, we separate the level required for transfering the syntactic structures into semantics, and vice-versa, from the level that controls those structures so that only the ones that TAG considers to be admissible (i.e., TAG derivations) are kept. We show that this level allows us to account for the semantics of long-distance dependencies, quantification, separate modification without multiple adjunction, control verbs, raising verbs, etc. Moreover, this is done in a principled way, following the standard homomorphism between the syntactic and semantic categories of Montague (1973).

Figure 1: Overall architecture for the syntax-semantics interface



Contrary to Pogodalla (2004a) (and Kallmeyer and Romero (2004,

2008)), and similarly to synchronous TAG analyses (Nesson and Shieber 2006; Nesson 2009), the semantic modelling we propose does not rely on an intermediate underspecification language. We show instead that it is not required in order to model long distance dependencies, raising verbs, or quantification. We also introduce and precisely describe the syntax-semantics modelling for adjectives (without multiple adjunctions), control verbs, and subordinate clauses. We also discuss the encoding of features. While these modellings can essentially be rephrased in synchronous TAG (and vice-versa.[2] The solid lines of Figure 1 also correspond to the synchronous TAG architecture for the syntax-semantics interface), it is not the case for some other ones, and we show how the approach easily extends, without requiring the design of new parsing algorithms, to other phenomena such as idioms[3] or subordinate clauses for which we propose a novel modelling. Other TAG extensions such as the cosubstitution operation proposed by Barker (2010) to model scope ambiguities also easily falls within the scope of our approach and can be given a type-raising account. It exemplifies how to model the non-functional nature of the form-meaning relation.

Finally, except for the type-raising account of quantification, the ACG model for the syntax-semantics interface of TAG that we propose belongs to the class of second-order ACGs. This class has the property that whatever the language we parse (strings, trees, or any kind of terms, such as first-order or higher-order logical formulas), parsing is polynomial. This parsing is implemented in the ACG toolkit.[4] Consequently, there is a parser that can actually recover the TAG derivation structure (if any) of some string, or of some derived tree, and interpret it as a logical formula, or that can actually recover the TAG derivation structure (if any) of some logical formula and interpret it as a derived tree or a string. The ACG framework is *inherently reversible* (Dymetman 1994), and parsing and generation of second-order ACGs are

[2] Synchronous TAG analyses often hinge on Multi-Component TAG (MCTAG, Weir (1988)), which is beyond the scope of this article. But we do not consider this to be an essential difference, since this can be integrated as well into the ACG approach (Pogodalla 2009). We discuss the differences between the approaches in Section 9.

[3] This encoding is due to Kobele (2012).

[4] The toolkit is available at `http://acg.loria.fr`.

performed in polynomial time, including for the modellings that go beyond TAG (except the type-raising account of quantification) without having to design new parsers. Note, however, that we do not yet address the problem of logical-form equivalence (Shieber 1993) which states that, even if two formulas are logically equivalent, it might be possible to recover a derivation structure for one but not for the other.

We also validated the modellings and the lexicons we provide in this article, both for parsing and generation, by implementing in the ACG toolkit all the examples of this article. This results in a toy grammar (corresponding to about forty elementary trees) covering the linguistic phenomena with the analyses presented in the article.[5] An extension to a real size TAG grammar for French is ongoing.

1.3 *Organisation of the article*

In Section 2, we show a functional interpretation of the substitution and adjunction operations. We review the definitions that are necessary to model strings and trees as λ-terms (Section 2.2) and we use them to model the elementary trees of TAG. Section 3 reviews the definitions that are specific to ACGs. We show their main composition models and present their formal properties.

In Section 4, we introduce the ACGs that are necessary for the encoding of the syntax-semantics interface: the ACG relating strings and derived trees in Section 4.1; the ACG relating derived trees and generalized derivation trees in Section 4.2 and Section 4.3. We show that these generalized derivations over-generate with respect to TAG derivation trees: the full TAG encoding is not yet completed, but we already have all the necessary bricks to implement the syntax-semantics interface.

Section 5 is devoted to the model of the syntax-semantics interface we propose. We first define the semantic representation language in Section 5.1. Then, in Section 5.2, we show how to interpret the generalized derivations as semantic terms and we provide several classical examples.

[5] The example files are available at `https://hal.inria.fr/hal-01242154/file/acg-examples.zip`. The script file illustrates the terms we use in this article and refers in comments to the relevant sections, equations, and term names

In Section 6 we complete the faithful TAG encoding by controlling the generalized derivations so that only TAG derivations are accepted. The correctness of the encoding is ensured by recovering, by ACG composition, de Groote's (2002) encoding. Then, again by ACG composition, we directly get a syntax-semantics interface for TAG. Because ACGs do not make use of features, we explain how we model the adjunction constraints induced by feature structures in TAG (Section 7).

In Section 8, we take advantage of the architecture we propose and give examples of modellings that this framework offers and that are beyond TAG. We finally discuss the relation of our approach to the syntax-semantics interface for TAG with other ones, in particular the ones using synchronous grammars or feature unification (Section 9).

2 BACKGROUND

2.1 *Adjunction and substitution*

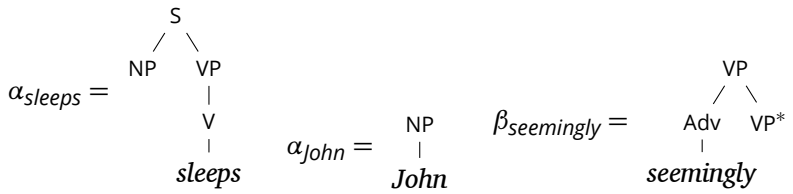
A TAG consists of a finite set of elementary trees whose nodes are labeled by terminal and non-terminal symbols. Nodes labeled with terminals can only be leaves. Elementary trees are divided into *initial* and *auxiliary* trees. Figure 2 exemplifies such trees. Substituting α_{John} in α_{sleeps} consists in replacing a leaf of α_{sleeps} labeled with a non-terminal symbol NP with the tree α_{John} whose root node is labeled by NP as well.[6] Figure 3(a) shows an example of such a substitution (at Gorn address 1) and of its result. The corresponding derivation tree recording the substitution is represented in Figure 3(b), where the Gorn address labels the edge between the two nodes, each of them being labeled by the name of the trees. Only initial trees, that possibly underwent some substitution or adjunction operations, can substitute into a leaf.

The adjunction of $\beta_{seemingly}$ into α_{sleeps} consists in inserting the tree $\beta_{seemingly}$ at the VP node of α_{sleeps}: the subtree of α_{sleeps} at its VP node is first removed then substituted to the VP foot node of $\beta_{seemingly}$, and the whole resulting tree is then plugged again at the VP node of α_{sleeps}, as Figure 4(a) shows. The associated derivation tree of Figure 4(b) records the adjunction with a dotted edge. Only auxiliary

[6] Substitution sites are often marked by decorating the label with a ↓ symbol.

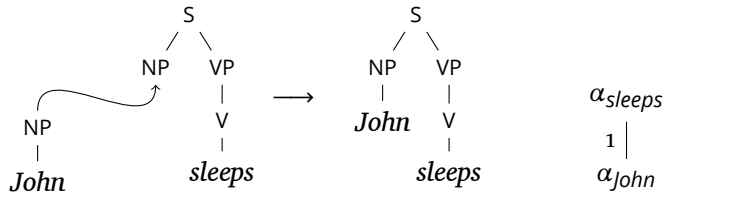
trees, that possibly underwent some substitution or adjunction operations, can adjoined into another tree.

Figure 2: TAG elementary trees



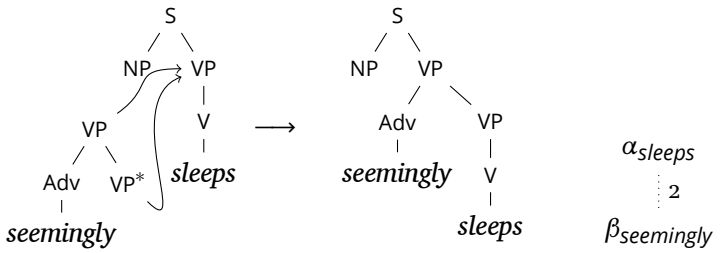
(a) Initial tree with one substitution node (b) Initial tree with no substitution node (c) Auxiliary tree

Figure 3: Substitution operation



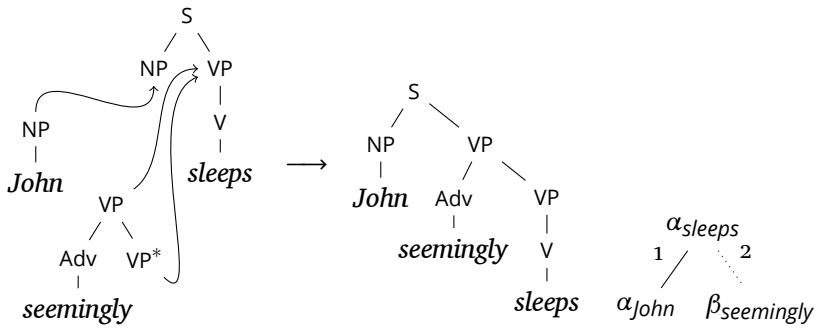
(a) Substitution (b) Derivation tree

Figure 4: Adjunction operation



(a) Adjunction (b) Derivation tree

Figure 5 shows a TAG analysis of *John seemingly sleeps* with both operations and the associated derivation tree.



(a) Adjunction and substitution (b) Derivation tree

Figure 5: TAG analysis of *John seemingly sleeps*

2.2 *TAG elementary trees as functions*

We now present the two operations of adjunction and substitution using a functional interpretation of the elementary trees. We use the standard notations of the typed λ-calculus and we formally present the syntax of λ-terms and their types.

Definition 1 (Types)**.** Let A be a set of atomic types. The set $\mathscr{T}(A)$ of *implicative types* built upon A is defined with the following grammar:

$$\mathscr{T}(A) ::= A \mid \mathscr{T}(A) \multimap \mathscr{T}(A) \mid \mathscr{T}(A) \rightarrow \mathscr{T}(A)$$

The set of *linear implicative types* built upon A is defined with the following grammar:

$$\mathscr{T}^{o}(A) ::= A \mid \mathscr{T}^{o}(A) \multimap \mathscr{T}^{o}(A)$$

Definition 2 (Higher-Order signatures)**.** A *higher-order signature* Σ is a triple $\Sigma = \langle A, C, \tau \rangle$ where:

- A is a finite set of atomic types;
- C is a finite set of constants;
- $\tau : C \rightarrow \mathscr{T}(A)$ is a function assigning types to constants.

A higher-order signature $\Sigma = \langle A, C, \tau \rangle$ is *linear* if the codomain of τ is $\mathscr{T}^{o}(A)$.

Definition 3 (λ-Terms)**.** Let X be an countably infinite set of λ-variables. The set $\Lambda(\Sigma)$ of λ-terms built upon a higher-order signature $\Sigma = \langle A, C, \tau \rangle$ is inductively defined as follows:

- if $c \in C$ then $c \in \Lambda(\Sigma)$;
- if $x \in X$ then $x \in \Lambda(\Sigma)$;
- if $x \in X$ and $t \in \Lambda(\Sigma)$ and x occurs free in t exactly once, then $\lambda^{o} x.t \in \Lambda(\Sigma)$;
- if $x \in X$ and $t \in \Lambda(\Sigma)$, then $\lambda x.t \in \Lambda(\Sigma)$;
- if $t, u \in \Lambda(\Sigma)$ then $(t\,u) \in \Lambda(\Sigma)$.

Note there is a linear λ-abstraction (denoted by λ^{o}) and a (usual) intuitionistic λ-abstraction (denoted by λ). A variable that is bound by λ^{o} occurs exactly once in the body of the abstraction, whereas it can occur zero, one, or any number of times when it is bound by λ. This distinction is important when discussing the complexity of parsing with ACGs.

We also use the usual notions of α, β, and η conversions (Barendregt 1984). We also use the left associativity of the (linear and non-linear) application (so $(t\ u)\ v$ is written $t\ u\ v$), the right associativity of the (linear and non-linear) abstraction over several variables (so $\lambda x.(\lambda y.(\lambda z.t)) = \lambda x\ y\ z.t = \lambda x\ y.\lambda z.t$, etc., the same for λ^{o}), and the right associativity of the implication (so $\alpha \to (\beta \to \gamma) = \alpha \to \beta \to \gamma$, the same for $\multimap$).

Definition 4 (Typing judgment)**.** Given a higher-order signature Σ, the typing rules are given with an inference system whose judgments are of the form: $\Gamma; \Delta \vdash_{\Sigma} t : \alpha$ where:

- Γ is a finite *set* of non-linear variable typing declarations of the form $x : \beta$ where x is a variable and β is a type;
- Δ is a finite *multi-set* of linear variable typing declarations of the form $x : \beta$ where x is a variable and β is a type. In order to distinguish the elements of the typing declaration, we always use variables with different names.

Both Γ and Δ may be empty. If both of them are empty, we usually write $t : \alpha$ (t is of type α) instead of $\vdash_{\Sigma} t : \alpha$. Moreover, we drop the Σ subscript when the context permits. Table 1 gives the typing rules: constant introduction, variable introduction (linear and non-linear), linear abstraction and linear application, (non-linear) abstraction and (non-linear) application, in this order.

Table 1: Typing rules for deriving typing judgments

$$\frac{}{\Gamma;\vdash_\Sigma c : \tau(c)}\ \text{(const.)}$$

$$\frac{}{\Gamma; x : \alpha \vdash_\Sigma x : \alpha}\ \text{(lin. var.)} \qquad \frac{}{\Gamma, x : \alpha;\vdash_\Sigma x : \alpha}\ \text{(var.)}$$

$$\frac{\Gamma;\Delta, x : \alpha \vdash_\Sigma t : \beta}{\Gamma;\Delta \vdash_\Sigma \lambda^{o} x.t : \alpha \multimap \beta}\ \text{(lin. abs.)} \qquad \frac{\Gamma;\Delta_1 \vdash_\Sigma t : \alpha \multimap \beta \quad \Gamma;\Delta_2 \vdash_\Sigma u : \alpha}{\Gamma;\Delta_1, \Delta_2 \vdash_\Sigma (t\,u) : \beta}\ \text{(lin. app.)}$$

$$\frac{\Gamma, x : \alpha;\Delta \vdash_\Sigma t : \beta}{\Gamma;\Delta \vdash_\Sigma \lambda x.t : \alpha \to \beta}\ \text{(abs.)} \qquad \frac{\Gamma;\Delta \vdash_\Sigma t : \alpha \to \beta \qquad \Gamma;\vdash_\Sigma u : \alpha}{\Gamma;\Delta \vdash_\Sigma (t\,u) : \beta}\ \text{(app.)}$$

The fact that Γ is a set and Δ is a multi-set corresponds to implicitly allowing for the structural rules of contraction and weakening for the non-linear context, and disallowing them on the linear context.[7]

Remark 1. From a logical point of view, the theorems that can be proved using only the non-linear or the linar context are different. For instance, if $c : \alpha \to \alpha \to \beta$ and $d : \alpha \multimap \alpha \multimap \beta$ are constants of Σ, $x : \alpha;\vdash c\ x\ x : \beta$ is derivable as the following derivation shows:

$$\frac{\dfrac{\dfrac{}{x : \alpha;\vdash_\Sigma c : \alpha \to \alpha \to \beta}\ \text{(const.)} \qquad \dfrac{}{x : \alpha;\vdash_\Sigma x : \alpha}\ \text{(var.)}}{x : \alpha;\vdash_\Sigma (c\,x) : \alpha \to \beta}\ \text{(app.)} \qquad \dfrac{}{x : \alpha;\vdash_\Sigma x : \alpha}\ \text{(var.)}}{x : \alpha;\vdash_\Sigma ((c\,x)\,x) : \beta}\ \text{(app.)}$$

whereas $;x : \alpha \vdash d\ x\ x : \beta$ is not (and $;x : \alpha, y : \alpha \vdash d\ x\ y : \beta$ is).

Remark 2. The linear context of the second premise in the non-linear application rule ($\Gamma;\vdash_\Sigma u : \alpha$) is empty. This is required in order to avoid duplicating or erasing linear variables by non-linear application to a linear variable. Otherwise we could have derivations such as:

$$\frac{\begin{array}{c}\vdots\\ ;\vdash_\Sigma \lambda x.c\ x\ x : \alpha \to \beta\end{array} \qquad ;y : \alpha \vdash_\Sigma y : \alpha}{;y : \alpha \vdash_\Sigma (\lambda x.c\ x\ x)\ y : \beta}\ \text{(app.)}$$

Then we have that $y : \alpha$ belongs to the linear context, but $(\lambda x.c\ x\ x)\ y$ reduces to $c\ y\ y$ where y is duplicated.

[7] Contraction corresponds to allowing the duplication of hypotheses, and weakening corresponds to allowing the deletion of useless hypothesis.

Definition 5 (Linear and almost linear λ-terms)**.** A term without any λ where each λ^{o} binds exactly one variable, and no subterm contains more than one free occurrence of the same variable is a *linear* λ-term, otherwise it is *non-linear*.

A term where each λ^{o} binds exactly one variable, where each λ binds at least one variable, and no subterm contains more than one free occurrence of the same variable, except if the variable has an atomic type, is an *almost linear* λ-term.

The notion of linearity and almost linearity are important with respect to the tractability and the computational complexity of the parsing algorithms, because they allow for characterising the set of almost linear λ-terms that are β-equivalent to some given almost linear term (Kanazawa 2017, p. 1120).

Definition 6 (Order)**.** The order $\mathsf{ord}(\tau)$ of a type $\tau \in \mathscr{T}(A)$ is inductively defined as:

- $\mathsf{ord}(a) = 1$ if $a \in A$
- $\mathsf{ord}(\alpha \multimap \beta) = \mathsf{ord}(\alpha \to \beta) = \max(1 + \mathsf{ord}(\alpha), \mathsf{ord}(\beta))$ otherwise

By extension, the order of a term is the order of its type.

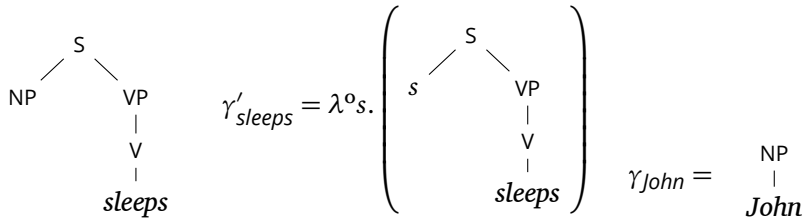
Remark 3. Second-order terms (i.e., terms whose order is 2) play an important role in our TAG encoding, and more generally for the expressive power of ACGs. A second-order term has type $a_1 \multimap a_2 \ldots \multimap a_n \multimap a$ where a, $a_1, \ldots, a_n$ are atomic types. If a signature Σ contains only first-order and second-order constants, an easy induction shows that ground terms (i.e., terms with no free variable) of atomic types in $\Lambda(\Sigma)$ do not contain any variable (bound or free) at all. In particular, they cannot have terms of the form $\lambda^{o}x.u$ or $\lambda x.u$ as sub-terms.

We now assume the single atomic type T of trees and constants of this type (in Section 2.3 we make explicit how to systematically encode trees into λ-terms).

2.2.1 Substitution as function application

The ability for the tree of Figure 6(a) to accept a substitution at its NP node allows it to be considered as a function that takes a tree as argument and replaces the NP node by this argument. Hence we can represent it as the function γ'_{sleeps} shown in Figure 6(b) with γ'_{sleeps} :

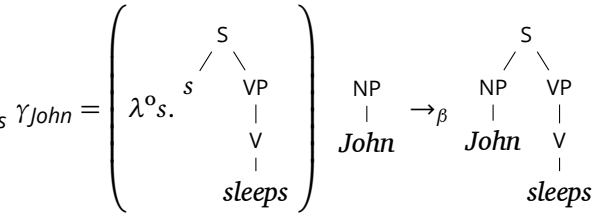
$T \multimap T$. A tree where no substitution can occur can be represented as $\gamma_{John} : T$ (see Figure 6(c)).



(a) TAG initial tree (b) Function from trees to trees (c) Term of type T

Figure 6: Functional interpretation of the substitution operation

Applying the function γ'_{sleeps} to the simple tree γ_{John} of Figure 6(c) and performing β-reduction gives the expected result as (1) shows.



(1)

Note that despite the derived tree for *John sleeps* has a root labelled by S and the elementary tree for *John* has a root labelled by NP, they are represented by the terms $\gamma'_{sleeps}\ \gamma_{John}$ and γ_{Joh} which both have type T. The issue of recording the distinction is addressed in Section 4.3.

2.2.2 Adjunction as function application

In order to deal with the adjunction operation, we first observe what happens to the auxiliary tree $\beta_{seemingly}$ in Figure 4 (p. 8): a subtree of the tree it is adjoined to (the VP rooted subtree of α_{sleeps}) is substituted at the VP* foot node of $\beta_{seemingly}$. This means that the auxiliary tree also behaves as a function from trees to trees and can be represented as in Figure 7(a) on the following page with $\gamma'_{seemingly} : T \multimap T$. Then, a tree with an adjunction site can be represented by a term such as $\gamma''_{sleeps} : (T \multimap T) \multimap T$ in Figure 7(b). Note the higher-order type of γ''_{sleeps}.

Figure 7: Functional interpretation of the adjunction operation

$$\gamma'_{seemingly} = \lambda^{o} v. \left(\begin{array}{c} \text{VP} \\ \text{Adv} \quad v \\ \textit{seemingly} \end{array} \right)$$

(a) Function from trees to trees

$$\gamma'_{sleeps} = \lambda^{o} a. \left(\begin{array}{c} \text{S} \\ \text{NP} \quad \\ a \left(\begin{array}{c} \text{VP} \\ \text{V} \\ \textit{sleeps} \end{array} \right) \end{array} \right)$$

(b) Elementary tree ready to accept an adjunction

In order to model the adjunction, we then apply γ''_{sleeps} to $\gamma'_{seemingly}$ and perform β-reductions as (2) shows.

$$\gamma''_{sleeps}\ \gamma'_{seemingly} = \left(\lambda^{o} a. \begin{array}{c} \text{S} \\ \text{NP} \quad \\ a \left(\begin{array}{c} \text{VP} \\ \text{V} \\ \textit{sleeps} \end{array} \right) \end{array} \right) \left(\lambda^{o} v. \begin{array}{c} \text{VP} \\ \text{Adv} \quad v \\ \textit{seemingly} \end{array} \right)$$

$$\rightarrow_{\beta} \begin{array}{c} \text{S} \\ \text{NP} \quad \\ \left(\lambda^{o} v. \begin{array}{c} \text{VP} \\ \text{Adv} \quad v \\ \textit{seemingly} \end{array} \right) \left(\begin{array}{c} \text{VP} \\ \text{V} \\ \textit{sleeps} \end{array} \right) \end{array} \tag{2}$$

$$\rightarrow_{\beta} \begin{array}{c} \text{S} \\ \text{NP} \quad \text{VP} \\ \text{Adv} \quad \text{VP} \\ \textit{seemingly} \quad \text{V} \\ \textit{sleeps} \end{array}$$

We now (almost) are in position to define the function standing for the elementary tree representing the intransitive verb *sleeps* in its

canonical form as in Figure 8 with $\gamma'''_{sleeps} : (T \multimap T) \multimap T \multimap T$. Such a term can be used to represent the TAG analysis of *John seemingly sleeps* shown in Figure 5 with the β-reduction of $\gamma'''_{sleeps}\ \gamma'_{seemingly}\ \gamma_{John}$ shown in (3).

$$\gamma'''_{sleeps} = \lambda^{o} a\ s.$$

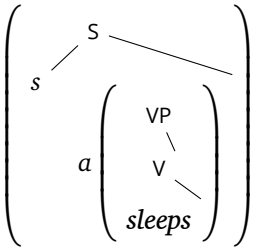
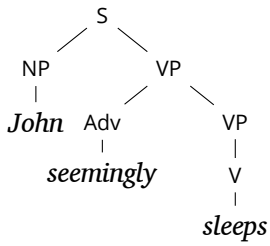


Figure 8: Elementary tree representation available to substitution and adjunction operations

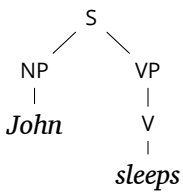
$$\gamma'''_{sleeps}\ \gamma'_{seemingly}\ \gamma_{John} \rightarrow_{\beta}$$



(3)

Remark 4 (No adjunction)**.** Typing γ'''_{sleeps} with $(T \multimap T) \multimap T \multimap T$ makes it require an adjunction (the first $(T \multimap T)$ argument) to return a plain tree term of type T. But of course, we also want to use this term in case no adjunction in a TAG analysis would occur, as in *John sleeps*. We make use of a fake adjunction, applying γ'''_{sleeps} to the identity function $I = \lambda^{o} x.x : T \multimap T$.[8] Then (4) holds.

$$\gamma'''_{sleeps}\ I\ \gamma_{John} \rightarrow_{\beta}$$



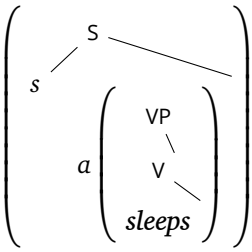
(4)

[8] This idea is present in the seminal work with ACGs (de Groote 2002; Pogodalla 2004a), but also in the synchronous approach (Shieber 2004, 2006) and in (Shieber 2014) as *vestigial auxiliary tree*.

Finally, we have to also model the possible adjunction on the S node of α_{sleeps}. So the corresponding term γ_{sleeps} has type $(T \multimap T) \multimap (T \multimap T) \multimap T \multimap T$ where the first argument stands for the auxiliary tree to be adjoined at the S node, the second argument stands for the auxiliary tree to be adjoined at the VP node, and the third argument stands for the tree to be substituted at the NP node as Figure 9 shows.[9]

Figure 9: Encoding of α_{sleeps} available to substitution and adjunctions both at the VP and at the S nodes

$\gamma_{sleeps} = \lambda^o S\ a\ s.S$



Remark 5 (Multiple adjunction)**.** Following Vijay-Shanker (1987), the typing we provide prevents two adjunctions from occurring at the same node in the same elementary tree. We discuss this difference with the multiple-adjunction approach of Schabes and Shieber (1994) in Section 5. Accordingly, an auxiliary tree should typically also allow for adjunction at its root. So instead of using $\gamma'_{seemingly} : T \multimap T$, we use the terms defined in Figure 10 in order to analyze sentences such as *John usually seemingly sleeps* as in Figure 11 with the term $\gamma_{sleeps}\ I\ (\gamma_{seemingly}\ (\gamma_{usually}\ I))\ \gamma_{John}$.[10]

2.3 *Trees and strings as λ-terms*

So far, we did not make explicit how to represent strings and trees as λ-terms. In particular, we did not tell how strings can combine and how the parent-child relation can be represented. While this is quite standard, and because we use this encoding to implement the example

[9] We could also allow adjunctions to the V node in a similar way. But we do not use examples of such adjunctions, and for sake of conciseness, we keep the type as small as required by the examples.

[10] Although $\lambda^o v.\gamma'_{usually}\ (\gamma'_{seemingly}\ v) = \gamma_{seemingly}\ (\gamma_{usually}\ I)$, introducing $\gamma_{seemingly}$ and $\gamma_{usually}$ with this more complex types is important because, as we will see in Section 6, at the most abstract level, we want terms without any free or bound variable to represent derivations (see Remark 3).

$$\gamma_{seemingly} = \lambda^o a\ v.a \left(\begin{array}{c} \text{VP} \\ \text{Adv} \quad v \\ \textit{seemingly} \end{array} \right) : (T \multimap T) \multimap T \multimap T$$

$$\gamma_{usually} = \lambda^o a\ v.a \left(\begin{array}{c} \text{VP} \\ \text{Adv} \quad v \\ \textit{usually} \end{array} \right) : (T \multimap T) \multimap T \multimap T$$

Figure 10: Auxiliary tree representation available to adjunction operations

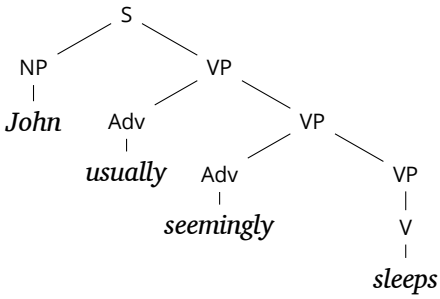


Figure 11: A TAG analysis of *John usually seemingly sleeps*

grammars using the ACG toolkit, this section describes how it can be done.

2.3.1 Encoding strings

We encode strings over an alphabet C using the following higher-order signature $\Sigma^C_{strings} = \langle A_\sigma, C, \tau_\sigma \rangle$ where:

- $A_\sigma = \{o\}$ contains a unique atomic type o;
- τ_σ is the constant function that maps any constant to the type σ, the string type, defined as $(o \multimap o)$. Note it is not an atomic type.

We use the notation $\triangleq$ to introduce terms or types that are defined using the atomic types and the constants of a signature, but are not atomic types nor constants of this signature. So $\sigma \triangleq (o \multimap o)$. We also define two other terms:

- $+ \triangleq \lambda^o f\ g.\lambda^o z.f(g\ z)$ (the function composition, used with an infix notation) to represent the *concatenation*;
- $\epsilon \triangleq \lambda^o x.x$ (the identity function) to represent the empty string.

It is easy to check that $+$ is associative and that ϵ is a neutral element for $+$.

Remark 6. If a and b are two strings, $a+b = \lambda^{o}z.a\ (b\ z)$. In this article, we usually do not unfold the definition of $+$ and we use the notation $x_1 + \ldots + x_n$ to represent the string $\lambda^{o}z.x_1\ (\ldots(x_n\ z)\ldots)$.

2.3.2 Encoding trees

Trees were originally defined in TAG using a mapping from positions (or Gorn addresses) to labels, elements of a vocabulary (Joshi *et al.* 1975). Hence, the same node label could be used to represent nodes of different arity. For instance, in Figure 2(a) (p. 8), the VP node has arity 1 whereas the VP node of Figure 2(c) has arity 2.

We prefer to represent trees as terms over a ranked alphabet as in (Shieber 2004; Comon *et al.* 2007) in order to make the encoding of trees as λ-terms easier. So we use the notation X_n with n the arity of the symbol X_n. It allows us to distinguish two nodes with n and m ($n \neq m$) children that would be mapped to the same label X by using the different symbols X_n and X_m. As terminal symbols can only occur as leaves, they always have arity 0, so we do not use any subscript for them.

In order to encode the trees defined over a ranked alphabet $\mathscr{F}_a = (\mathscr{F}, \mathrm{arity})$, where arity is a mapping from $\mathscr{F}$ to $\mathbb{N}$, we use the following higher-order signature $\Sigma_{\mathit{trees}}^{\mathscr{F}_a} = \langle A_T, \mathscr{F}, \tau_T^{\mathscr{F}_a} \rangle$ where:

- $A_T = \{T\}$ contains a unique atomic type T, the type of trees;
- $\tau_T^{\mathscr{F}_a}$ is a function that maps any constant X such that $\mathrm{arity}(X) = n$ to the type $\underbrace{T \multimap \cdots \multimap T}_{n \text{ times}} \multimap T$. If $\mathrm{arity}(X) = 0$, then $\tau_T^{\mathscr{F}_a}(X) = T$.

For instance, the TAG elementary trees δ_{anchor}[11] of our running examples can be modeled as the functions (or terms) γ_{anchor} built on the signature Σ_{trees} as Table 2 (p. 19) shows.[12] Then (5) shows that $\gamma_5 = \gamma_{\mathit{sleeps}}\ I\ (\gamma_{\mathit{seemingly}}\ I)\ \gamma_{\mathit{John}}$ corresponds (modulo β) to the tree of

[11] We use the notation δ_{anchor} to refer either to the initial tree α_{anchor} or to the auxiliary tree β_{anchor}.

[12] Note that *sleeps* and *seemingly* are used as constants of arity 0 and have type T. We also introduce an auxiliary tree that can adjoin to the S node.

Table 2: Encoding of the TAG elementary trees with Σ_{trees}

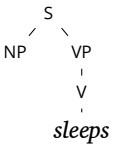
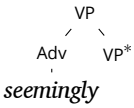
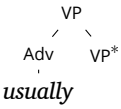
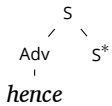
	Terms of $\Lambda(\Sigma_{trees})$	Corresponding TAG tree
γ_{John}	$= \mathsf{NP}_1\ \mathit{John}$ $: T$	
γ_{sleeps}	$= \lambda^{o} S\ a\ s.S\ (\mathsf{S}_2\ s\ (a\ (\mathsf{VP}_1\ (\mathsf{V}_1\ \mathit{sleeps}))))$ $: (T \multimap T) \multimap (T \multimap T) \multimap T \multimap T$	
$\gamma_{seemingly}$	$= \lambda^{o} a\ v.a\ (\mathsf{VP}_2\ (\mathsf{Adv}_1\ \mathit{seemingly})\ v)$ $: (T \multimap T) \multimap T \multimap T$	
$\gamma_{usually}$	$= \lambda^{o} a\ v.a\ (\mathsf{VP}_2\ (\mathsf{Adv}_1\ \mathit{usually})\ v)$ $: (T \multimap T) \multimap T \multimap T$	
γ_{hence}	$= \lambda^{o} a\ s.a\ (\mathsf{S}_2\ (\mathsf{Adv}_1\ \mathit{hence})\ s)$ $: (T \multimap T) \multimap T \multimap T$	
I	$= \lambda^{o} x.x : T \multimap T$	

Figure 5 (p. 9).

$$\begin{aligned}\gamma_5 &= \gamma_{sleeps}\ I\ (\gamma_{seemingly}\ I)\ \gamma_{John} \\ &\to_\beta \mathsf{S}_2\ (\mathsf{NP}_1\ \mathit{John})\ (\mathsf{VP}_2\ (\mathsf{Adv}_1\ \mathit{seemingly})\ (\mathsf{VP}_1\ (\mathsf{V}_1\ \mathit{sleeps})))\end{aligned} \tag{5}$$

We now want to relate the tree that is represented by the term $\gamma_{sleeps}\ I\ (\gamma_{seemingly}\ (\gamma_{usually}\ I))\ \gamma_{John} : T$ to the string *John usually seemingly sleeps* that is represented by the term $\mathit{John} + \mathit{usually} + \mathit{seemingly} + \mathit{sleeps} : \sigma$. We do this in Section 4, using an *interpretation* of the former as defined by an ACG.

3 ABSTRACT CATEGORIAL GRAMMARS

Grammars can be considered as a device to relate concrete objects to hidden underlying structures. For instance, context-free grammars relate strings to syntactic trees, and TAGs relate derived trees to deriva-

tion trees. However, in both cases, the underlying structure is not a first-class citizen of the formalism.

ACGs take another perspective and provide the user a direct way to define the parse structures of the grammar, the *abstract language*. Such structures are later on interpreted by a morphism, the *lexicon*, to get the concrete *object language*. The process of recovering an abstract structure from an object term is called *ACG parsing* and consists in inverting the lexicon. In this perspective, derivation trees of TAGs are represented as terms of an abstract language, while derived trees (and yields) are represented by terms of some other object languages. It is an object language of trees in the first case and an object language of strings in the second case. We also use a logical language as object language to express the semantic representations.

For sake of self-containedness, we first review the definitions of de Groote (2001).

Definition 7 (Lexicon)**.** Let $\Sigma_1 = \langle A_1, C_1, \tau_1 \rangle$ and $\Sigma_2 = \langle A_2, C_2, \tau_2 \rangle$ be two higher-order signatures, Σ_1 being linear. A *lexicon* $\mathscr{L} = \langle F, G \rangle$ from Σ_1 to Σ_2 is such that:

- $F : A_1 \to \mathscr{T}(A_2)$. We also note $F : \mathscr{T}(A_1) \to \mathscr{T}(A_2)$ its homomorphic extension, that is the function $\hat{F}$ that extends F such that $\hat{F}(\alpha \multimap \beta) = \hat{F}(\alpha) \multimap \hat{F}(\beta)$ and $\hat{F}(\alpha \to \beta) = \hat{F}(\alpha) \to \hat{F}(\beta)$;
- $G : C_1 \to \Lambda(\Sigma_2)$. We also note $G : \Lambda(\Sigma_1) \to \Lambda(\Sigma_2)$ its homomorphic extension, that is the function $\hat{G}$ that extends G such that $\hat{G}(t\ u) = \hat{G}(t)\ \hat{G}(u)$, $\hat{G}(x) = x$, $\hat{G}(\lambda x.t) = \lambda x.\hat{G}(t)$, and $\hat{G}(\lambda^{o} x.t) = \lambda^{o} x.\hat{G}(t)$;
- F and G are such that for all $c \in C_1$, $\vdash_{\Sigma_2} G(c) : F(\tau_1(c))$ is provable.

We also use $\mathscr{L}$ instead of F or G.

The lexicon is the interpreting device of ACGs.

Definition 8 (Abstract Categorial Grammar and vocabulary)**.** An *abstract categorial grammar* is a quadruple $\mathscr{G} = \langle \Sigma_1, \Sigma_2, \mathscr{L}, \mathsf{S} \rangle$ where:

- $\Sigma_1 = \langle A_1, C_1, \tau_1 \rangle$ and $\Sigma_2 = \langle A_2, C_2, \tau_2 \rangle$ are two higher-order signatures. Σ_1 (resp. Σ_2) is called the *abstract vocabulary* (resp. the *object vocabulary*) and $\Lambda(\Sigma_1)$ (resp. $\Lambda(\Sigma_2)$) is the set of *abstract terms* (resp. the set of *object terms*).
- $\mathscr{L} : \Sigma_1 \to \Sigma_2$ is a lexicon and Σ_1 is a linear signature.

- $\mathsf{s} \in \mathscr{T}(A_1)$ is the *distinguished type* of the grammar.

Given an ACG $\mathscr{G}_{name} = \langle \Sigma_1, \Sigma_2, \mathscr{L}_{name}, \mathsf{s} \rangle$, instead of using $\mathscr{L}_{name}(\alpha) = \beta$ (resp. $\mathscr{L}_{name}(t) = u$) in order to express that the interpretation of the type α is the type β (resp. the interpretation of the term t is the term u), we use the following notational variants: $\mathscr{G}_{name}(\alpha) = \beta$ and $\alpha :=_{name} \beta$ (resp. $\mathscr{G}_{name}(t) = u$ and $t :=_{name} u$). The subscript may be omitted if clear from the context.

Definition 9 (Abstract and object languages)**.** Given an ACG $\mathscr{G}$, the *abstract language* is defined by

$$\mathscr{A}(\mathscr{G}) = \{t \in \Lambda(\Sigma_1) \mid \vdash_{\Sigma_1} t : \mathsf{s} \text{ is derivable}\}$$

The *object language* is defined by

$$\mathscr{O}(\mathscr{G}) = \{u \in \Lambda(\Sigma_2) \mid \exists t \in \mathscr{A}(\mathscr{G}) \text{ such that } u = \mathscr{L}(t)\}$$

In this article, we consider object languages such as strings or logical formulas, and abstract languages such as derivation trees. Some languages, such as the language of derived trees, will be considered sometimes as object languages, sometimes as abstract languages.

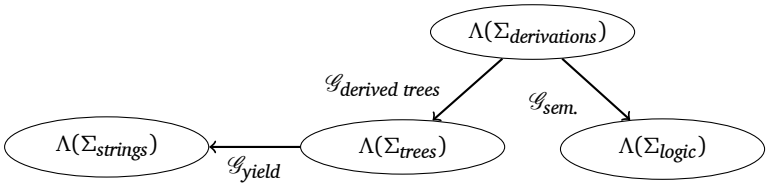
Parsing with an ACG $\mathscr{G}$ any term u that is built over the object vocabulary of $\mathscr{G}$ amounts to finding the abstract terms $t \in \mathscr{A}(\mathscr{G})$ such that $u = \mathscr{G}(t)$. In other words, ACG parsing is morphism inversion.

3.1 *ACG composition*

The lexicon defines the way structures are interpreted. It plays a crucial role in our proposal in two different ways. First, two interpretations may share the same abstract vocabulary, hence mapping a single structure into two different ones. For instance, the structure representing the derivations may be mapped both into a surface form and a semantic form. This composition is illustrated by $\mathscr{G}_{derived\ trees}$ and $\mathscr{G}_{sem.}$ sharing the $\Sigma_{derivations}$ vocabulary in Figure 12. It corresponds to the core model of the syntax-semantics interface as proposed by ACGs (de Groote 2001, Section 2.3), but also to the one proposed by synchronous TAG. It allows us to relate the derived trees and the semantic expressions that have the same derivation structures. We use this in Section 5 as our model of the syntax-semantics interface for TAG.

Second, the result of a first interpretation can itself be interpreted by a second lexicon when the object vocabulary of the first interpretation is the abstract vocabulary of the second one. This composition, illustrated by the $\mathcal{G}_{yield} \circ \mathcal{G}_{derived\ trees}$ composition in Figure 12, provides modularity. It also allows one to control admissible intermediate structures. For instance, the abstract language of $\mathcal{G}_{yield}$ may contain too many structures. If the object language of $\mathcal{G}_{derived\ trees}$ is a strict subset of this abstract language, then the object language of $\mathcal{G}_{yield} \circ \mathcal{G}_{derived\ trees}$ is a subset of the object language of $\mathcal{G}_{yield}$. We take advantage of this property in Section 4.2 to enforce the matching between node labels in substitution and adjunction operations, and to further restrict the set of derivations to only TAG derivations in Section 4.3. This ACG composition corresponds to the typical ACG way to control admissible parse structures.

Figure 12: ACG composition for control and syntax-semantic interface



3.2 *Formal properties of ACGs*

In this section, we review the main properties of ACGs and mention the relevant references. Two parameters are useful to define a hierarchy of ACGs: the *order* and the *complexity* of an ACG.

Definition 10 (Order and complexity of an ACG; ACG hierarchy)**.** The *order* of an ACG is the maximum of the orders of its abstract constants. The *complexity* of an ACG is the maximum of the orders of the realizations of its atomic types.

We call *second-order ACGs* the set of ACGs whose order is at most 2.

$\mathrm{ACG}_{(n,m)}$ denotes the set of ACGs whose order is at most n and whose complexity is at most m.

For instance, in Figure 12, $\mathcal{G}_{yield}$ is a second-order ACG (because all the constants of Σ_{trees} are of type $T \multimap \cdots \multimap T$ with T atomic, hence are at most second-order). On the other hand, $\Sigma_{derivations}$ is third-order

Table 3: The hierarchy of second-order ACGs

	String language	Tree language
$\mathrm{ACG}_{(1,n)}$	finite	finite
$\mathrm{ACG}_{(2,1)}$	regular	regular
$\mathrm{ACG}_{(2,2)}$	context-free	linear context-free
$\mathrm{ACG}_{(2,3)}$	non-duplicating macro well-nested multiple context-free	$\subset$ 1-visit attribute grammar
$\mathrm{ACG}_{(2,4)}$	mildly context-sensitive (multiple context-free)	tree-generating hyperedge replacement gram.
$\mathrm{ACG}_{(2,4+n)}$	$\mathrm{ACG}_{(2,4)}$	$\mathrm{ACG}_{(2,4)}$

(it contains terms such as $c_{seemingly} : (\mathsf{VP} \multimap \mathsf{VP}) \multimap \mathsf{VP} \multimap \mathsf{VP}$, where NP and VP are atomic, that are third-order. See Section 4). Hence $\mathscr{G}_{derived\ trees}$ is third-order as well.

The class of second-order ACGs is of particular interest because of its polynomial parsing property (Salvati 2005). When considering strings as the object language, the generated languages coincide with multiple context-free languages (Salvati 2006). When considering trees, the generated languages coincide with the tree languages generated by hyperedge replacement grammars (Kanazawa 2009). A further refinement on the ACG hierarchy provides a fine-grained correspondence with regular (string or tree) languages, context-free string and linear context-free tree languages, or well-nested multiple context-free languages (string), in particular tree-adjoining languages. Table 3 (p. 23) sums up some of the formal properties of second-order ACGs (de Groote and Pogodalla 2004; Salvati 2006; Kanazawa and Salvati 2007; Kanazawa 2009).

For second-order ACGs, parsing algorithms and optimization techniques are grounded on well established fields such as type-theory and Datalog. Kanazawa (2007) showed how parsing of second-order ACGs reduces to Datalog querying, offering a general method for getting efficient tabular parsing algorithms (Kanazawa 2017). This parsing method applies whatever the object language: representing strings, trees, but also any kind of (almost linear) λ-terms. When the object language consists of logical formulas, the latter can then be parsed as well, and the resulting parse structures can further be interpreted (e.g., as strings) to implement surface realization (Kanazawa 2007). This also allows for deriving algorithms with specific properties

such as prefix-correctness in a general way.[13]

The computational properties of lexicon inversion for ACGs have been studied for different classes of λ-terms.[14] It is worth noting that as far as second-order ACGs are concerned, whatever the form of the semantic λ-term, lexicon inversion is decidable (Salvati 2010) even with replication and vacuous abstraction of variables, though with a high computational complexity. From a theoretical point of view, this corresponds to removing any kind of semantic monotonicity requirement for generation (Shieber 1988; Shieber *et al.* 1989) in a very general setting.[15]

The examples we present in this article use only almost linear semantic terms. This allows us to run them in the ACG toolkit. The latter implements the method of parsing by reduction to Datalog, and allows us to parse strings, trees, and logical formulas using the grammars we propose. Large scale tests of the software are however ongoing work, and a quantitative evaluation is beyond the scope of this article.

Parsing with ACGs whose order is strictly greater than 2 is equivalent (Salvati 2005) to the decidability of the Multiplicative Exponential fragment of Linear Logic (MELL, Girard (1987)).[16] de Groote (2015) shows a reduction of ACG parsing of higher-order ACGs to linear logic programming. It is of course trivially (by linearity) decidable for ACGs where the interpretation of abstract constants always introduces a constant of the object language. But even in this case, third-order ACGs can generate languages that are NP-complete (Yoshinaka and Kanazawa 2005; Salvati 2006). For higher-order ACGs, the ACG toolkit implements abstract term interpretation, but no parsing.

[13] For a n^6 prefix-correct Earley recognizer for TAGs, see (Kanazawa 2008b).

[14] See for instance (Kanazawa 2017; Bourreau and Salvati 2011; Bourreau 2012) for the linear, almost linear, and almost affine cases.

[15] In its strongest form, this requirement corresponds to having lexicalized semantic recipes (i.e., where at least one constant appears and no deletion is allowed). Linear and almost linear pure terms (i.e., where no constant occur) are already dealt with in the Datalog reduction. Allowing deletion leads to more challenging issues. It is used, for instance, for the modelling of ellipsis (Kobele 2007; Bourreau 2012, 2013) or for providing intensional semantics to intension-insensitive words (de Groote and Kanazawa 2013).

[16] It has recently been proved to be decidable (Bimbó 2015).

4 RELATING GENERALIZED DERIVATIONS, TAG DERIVED TREES, AND STRINGS WITH ABSTRACT CATEGORIAL GRAMMARS

From now on, we assume a TAG $\mathscr{G} = (\mathscr{I}, \mathscr{A})$ where $\mathscr{I}$ is the set of initial trees and $\mathscr{A}$ the set of auxiliary trees. The labels of the trees in $\mathscr{I} \cup \mathscr{A}$ range over the alphabet V^0, and $C \subset V^0$ is the terminal alphabet. V is the set of symbols of V^0 disambiguated by subscripting them with their arity (except for terminal symbols of arity 0), and $\mathscr{V}$ is the associated ranked alphabet.

4.1 *Derived trees and strings*

In the constructions of Section 2.3, we introduced two higher-order signatures: $\Sigma_{strings} = \Sigma^{C}_{strings}$ and $\Sigma_{trees} = \Sigma^{\mathscr{V}}_{trees}$. We can now relate terms built on them using an ACG $\mathscr{G}_{yield} = \langle \Sigma_{trees}, \Sigma_{strings}, \mathscr{L}_{yield}, T \rangle$ by specifying $\mathscr{L}_{yield}$ as follows:

- $\mathscr{L}_{yield}(T) = \sigma$ (a tree is interpreted as a string);
- for $X_n \in V \setminus C$, $\mathscr{L}_{yield}(X_n) = \lambda^{o} x_1 \ \ldots \ x_n.x_1 + \cdots + x_n$ (a tree labeled by a non-terminal symbol is interpreted by the function that concatenate the interpretation of its children);
- for $a \in C$, $\mathscr{L}_{yield}(a) = a$ (a terminal symbol is interpreted by the same symbol as a string).

For instance, (8) (p. 26) shows that the yield of the tree represented by $\gamma_5 = \gamma_{sleeps}\ I\ (\gamma_{seemingly}\ I)\ \gamma_{John}$ (p. 19) actually is *John* + *seemingly*+*sleeps* (which can be rephrased as $\gamma_{sleeps}\ I\ (\gamma_{seemingly}\ I)\ \gamma_{John} :=_{yield}$ *John* + *seemingly* + *sleeps*). Indeed, we have (6).

$$
\begin{aligned}
\mathscr{G}_{yield}(\gamma_{John}) &= \mathscr{G}_{yield}(\mathsf{NP}_1\ John) \\
&= \mathscr{G}_{yield}(\mathsf{NP}_1)\ \mathscr{G}_{yield}(John) \\
&\qquad\qquad \text{because } \mathscr{G}_{yield} \text{ is a morphism} \\
&= (\lambda^{o} x.x)\ John \\
&\qquad\qquad \text{by definition of } \mathscr{G}_{yield} \text{ on constants} \\
&\rightarrow_{\beta} John
\end{aligned}
\tag{6}
$$

And with $\gamma_7 = \mathsf{VP}_2\ (\mathsf{Adv}_1\ \textit{seemingly})\ (\mathsf{VP}_1\ (\mathsf{V}_1\ \textit{sleeps}))$, we have (7), hence (8).

$$
\begin{aligned}
\mathscr{G}_{yield}(\gamma_7) &= \mathscr{G}_{yield}(\mathsf{VP}_2\ (\mathsf{Adv}_1\ \textit{seemingly})\ (\mathsf{VP}_1\ (\mathsf{V}_1\ \textit{sleeps})))\\
&= \mathscr{G}_{yield}(\mathsf{VP}_2)\ (\mathscr{G}_{yield}(\mathsf{Adv}_1\ \textit{seemingly}))\ (\mathscr{G}_{yield}(\mathsf{VP}_1\ (\mathsf{V}_1\ \textit{sleeps})))\\
&\qquad\qquad \text{because } \mathscr{G}_{yield} \text{ is a morphism}\\
&= (\lambda^{\mathrm{o}} x_1\ x_2.x_1 + x_2)\\
&\qquad (\mathscr{G}_{yield}(\mathsf{Adv}_1\ \textit{seemingly}))\ (\mathscr{G}_{yield}(\mathsf{VP}_1\ (\mathsf{V}_1\ \textit{sleeps})))\\
&\qquad\qquad \text{by definition of } \mathscr{G}_{yield} \text{ on } \mathsf{VP}_2\\
&\to_\beta \mathscr{G}_{yield}(\mathsf{Adv}_1\ \textit{seemingly}) + \mathscr{G}_{yield}(\mathsf{VP}_1\ (\mathsf{V}_1\ \textit{sleeps}))\\
&= (\mathscr{G}_{yield}(\mathsf{Adv}_1)\ \mathscr{G}_{yield}(\textit{seemingly})) + (\mathscr{G}_{yield}(\mathsf{VP}_1)\ (\mathscr{G}_{yield}(\mathsf{V}_1\ \textit{sleeps})))\\
&\qquad\qquad \text{because } \mathscr{G}_{yield} \text{ is a morphism}\\
&= ((\lambda^{\mathrm{o}} x.x)\ \mathscr{G}_{yield}(\textit{seemingly})) + ((\lambda^{\mathrm{o}} x.x)\ (\mathscr{G}_{yield}(\mathsf{V}_1\ \textit{sleeps})))\\
&\qquad\qquad \text{by definition of } \mathscr{G}_{yield} \text{ on } \mathsf{Adv}_1 \text{ and } \mathsf{VP}_1\\
&= \mathscr{G}_{yield}(\textit{seemingly}) + \mathscr{G}_{yield}(\mathsf{V}_1\ \textit{sleeps})\\
&= \textit{seemingly} + (\mathscr{G}_{yield}(\mathsf{V}_1)\ (\mathscr{G}_{yield}(\textit{sleeps})))\\
&= \textit{seemingly} + ((\lambda^{\mathrm{o}} x.x)\ \textit{sleeps})\\
&= \textit{seemingly} + \textit{sleeps}
\end{aligned}
\tag{7}
$$

$$
\begin{aligned}
\mathscr{G}_{yield}(\gamma_5) &= \mathscr{G}_{yield}(\mathsf{S}_2\ (\mathsf{NP}_1\ \textit{John})\ (\mathsf{VP}_2\ (\mathsf{Adv}_1\ \textit{seemingly})\ (\mathsf{VP}_1\ (\mathsf{V}_1\ \textit{sleeps}))))\\
&\qquad\qquad \text{by (5), p. 19}\\
&= \mathscr{G}_{yield}(\mathsf{S}_2)\ (\mathscr{G}_{yield}(\mathsf{NP}_1\ \textit{John}))\ (\mathscr{G}_{yield}(\gamma_7))\\
&= (\lambda^{\mathrm{o}} x_1\ x_2.x_1 + x_2)\ \textit{John}\ (\textit{seemingly} + \textit{sleeps})\\
&\qquad\qquad \text{by (6) and (7)}\\
&\to_\beta \textit{John} + \textit{seemingly} + \textit{sleeps}
\end{aligned}
\tag{8}
$$

4.2 *Derivation trees and derived trees*

In this section, we illustrate how to introduce more control on the accepted structures. Note indeed that according to the definition of $\mathscr{G}_{yield}$, whatever is a closed term of type T belongs to its abstract language. For instance, $\gamma_{13} = \gamma_{\textit{seemingly}}\ I\ \gamma_{\textit{John}}$ is a well-typed term of type

T corresponding to the tree of Figure 13 as (9) shows. Consequently, its interpretation *seemingly* + *John* belongs to the object language.

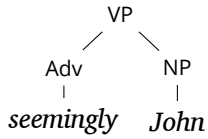


Figure 13: Tree in the abstract language of $\mathscr{G}_{yield}$

$$\begin{aligned}\gamma_{13} &= \gamma_{seemingly} \; I \; \gamma_{John} \\ &\rightarrow_{\beta} \mathsf{VP}_2 \; (\mathsf{Adv}_1 \; \textit{seemingly}) \; (\mathsf{NP}_1 \; \textit{John}) \\ &:=_{yield} \textit{seemingly} + \textit{John}\end{aligned} \tag{9}$$

In order to avoid such terms belonging to the language we are interested in, we provide another ACG, $\mathscr{G}_{derived\ trees}$, such that its object language is a strict subset of $\mathscr{A}(\mathscr{G}_{yield})$ (see Figure 12 p. 22). Consequently, the object language of $\mathscr{G}_{yield} \circ \mathscr{G}_{derived\ trees}$ is a subset (strict in this case, as expected) of $\mathscr{O}(\mathscr{G}_{yield})$. $\mathscr{G}_{derived\ trees}$ is defined as $\mathscr{G}_{derived\ trees} = \langle \Sigma_{derivations}, \Sigma_{trees}, \mathscr{L}_{derived\ trees}, \mathsf{S} \rangle$.

4.3 *Generalized derivation trees*

4.3.1 A vocabulary for derivations: the $\Sigma_{derivations}$ signature

Adjoining $\gamma_{seemingly}$ on γ_{John} is possible in $\Lambda(\Sigma_{trees})$ because the type T does not take the node labels into account. Hence, there is, for instance, no distinction between trees rooted by VP and trees rooted by NP. We introduce this distinction in a new signature $\Sigma_{derivations} = \langle A_{derivations}, C_{derivations}, \tau_{derivations} \rangle$. $A_{derivations} = V^0$ is the set of non-terminal symbols of the TAG grammar $\mathscr{G}$. Then, for any $\delta_{anchor} \in \mathscr{I} \cup \mathscr{A}$ an elementary tree of $\mathscr{G}$, we define c_{anchor} a constant of type $(X^1 \multimap X^1) \multimap \cdots \multimap (X^n \multimap X^n) \multimap Y^1 \multimap \cdots \multimap Y^m \multimap \alpha$ where:

- the X^i are the labels of the n internal nodes of δ_{anchor} labeled with a non-terminal where an adjunction is possible (by convention we use the breadth-first traversal); [17]

[17] Instead of the types $(X^i \multimap X^i)$, we may have types $X^i_{i_1} \multimap X^i_{i_2}$ to denote a difference between the top and the bottom feature of the node of label X^i. This is in particular used to account for selecting adjoining constraints as described in Feature-based TAG (FTAG, Vijay-Shanker and Joshi (1988, 1991)). See note 18.

- the Y^i are the labels of the m leaves labeled with non-terminals, *not counting the foot node if δ_{anchor} is an auxiliary tree*, of δ_{anchor} (by convention, we use the left-right order);
- let Z be the label of the root node of δ_{anchor}. $\alpha = Z$ if $\delta_{anchor} \in \mathscr{I}$ is an initial tree, and $\alpha = Z' \multimap Z$ with Z' the label of the foot node if $\delta_{anchor} \in \mathscr{A}$ is an auxiliary tree.[18] In the latter case, we call $Z' \multimap Z$ the *modifier type* of the constant modelling the auxiliary tree.

We get for instance the constants typed as in (10)[19] from the elementary trees of Figure 2 (p. 8).

$$\begin{aligned} c_{sleeps} &: (\mathsf{S} \multimap \mathsf{S}) \multimap (\mathsf{VP} \multimap \mathsf{VP}) \multimap \mathsf{NP} \multimap \mathsf{S} \\ c_{John} &: \mathsf{NP} \\ c_{seemingly} &: (\mathsf{VP} \multimap \mathsf{VP}) \multimap \mathsf{VP} \multimap \mathsf{VP} \end{aligned} \tag{10}$$

For each non-terminal X of the TAG grammar where an adjunction can occur, we also define $I_X : X \multimap X$ as in (11). These constants play a similar role as I at the Σ_{trees} level: they are used when a TAG derivation does not involve any adjunction on sites where it would be possible to have some.

$$\begin{aligned} I_{\mathsf{S}} &: \mathsf{S} \multimap \mathsf{S} \\ I_{\mathsf{VP}} &: \mathsf{VP} \multimap \mathsf{VP} \end{aligned} \tag{11}$$

Then the set of typed constants of $\Sigma_{derivations}$ is $C_{derivations} = \{c_{anchor} | \ \delta_{anchor} \in \mathscr{I} \cup \mathscr{A}\} \cup \{I_X | \ X \in V^0\}$ and $\tau_{derivations}$ is the associated typing function defined as above. The typing provided by $\Sigma_{derivations}$ now disallows the application of $c_{seemingly}\ I_{\mathsf{VP}} : \mathsf{VP} \multimap \mathsf{VP}$ to $c_{John} : \mathsf{NP}$.

We now need to relate the terms of $\Lambda(\Sigma_{derivations})$ to the terms of $\Lambda(\Sigma_{trees})$ by a suitable interpretation.

4.3.2 Interpretation of derivations as derived trees: the $\mathscr{G}_{derived\ trees}$ ACG

In order to define $\mathscr{G}_{derived\ trees} = \langle \Sigma_{derivations}, \Sigma_{trees}, \mathscr{L}_{derived\ trees}, \mathsf{S} \rangle$ we are left with defining $\mathscr{L}_{derived\ trees}$. All the atomic types (S, VP, etc.) are

[18] In standard TAG, we typically have $Z = Z'$. However, we shall see examples in Sections 5.3.2 and 7 where such a distinction is relevant.

[19] We assume that no adjunction is allowed on the V node nor on the Adv node.

Table 4: Interpretation of $\Sigma_{derivations}$ constants by $\mathscr{G}_{derived\ trees}$

c_{John}	$: \mathsf{NP}$	
	$:=_{derived\ trees}\ \gamma_{John}$	$= \mathsf{NP}_1\ \mathit{John} : T$
c_{sleeps}	$: (\mathsf{S} \multimap \mathsf{S}) \multimap (\mathsf{VP} \multimap \mathsf{VP}) \multimap \mathsf{NP} \multimap \mathsf{S}$	
	$:=_{derived\ trees}\ \gamma_{sleeps}$	$= \lambda^{o} S\ a\ s.S\ (\mathsf{S}_2\ s\ (a\ (\mathsf{VP}_1\ (\mathsf{V}_1\ \mathit{sleeps}))))$
		$: (T \multimap T) \multimap (T \multimap T) \multimap T \multimap T$
$c_{seemingly}$	$: (\mathsf{VP} \multimap \mathsf{VP}) \multimap \mathsf{VP} \multimap \mathsf{VP}$	
	$:=_{derived\ trees}\ \gamma_{seemingly}$	$= \lambda^{o} a\ v.a\ (\mathsf{VP}_2\ (\mathsf{Adv}_1\ \mathit{seemingly})\ v)$
		$: (T \multimap T) \multimap T \multimap T$
$c_{usually}$	$: (\mathsf{VP} \multimap \mathsf{VP}) \multimap \mathsf{VP} \multimap \mathsf{VP}$	
	$:=_{derived\ trees}\ \gamma_{usually}$	$= \lambda^{o} a\ v.a\ (\mathsf{VP}_2\ (\mathsf{Adv}_1\ \mathit{usually})\ v)$
		$: (T \multimap T) \multimap T \multimap T$
c_{hence}	$: (\mathsf{S} \multimap \mathsf{S}) \multimap \mathsf{S} \multimap \mathsf{S}$	
	$:=_{derived\ trees}\ \gamma_{hence}$	$= \lambda^{o} a\ s.a\ (\mathsf{S}_2\ (\mathsf{Adv}_1\ \mathit{hence})\ s)$
		$: (T \multimap T) \multimap T \multimap T$
I_{S}	$: \mathsf{S} \multimap \mathsf{S}$	
	$:=_{derived\ trees}\ I$	$= \lambda^{o} x.x : T \multimap T$
I_{VP}	$: \mathsf{VP} \multimap \mathsf{VP}$	
	$:=_{derived\ trees}\ I$	$= \lambda^{o} x.x : T \multimap T$

interpreted as trees (i.e., with the T type). And for a TAG elementary tree δ_{anchor}, the constant c_{anchor} is interpreted by γ_{anchor} (defined in Section 2.3.2). This leads us to the interpretations of Table 4.

In Section 4.2, we noticed that $\gamma_{13} = \gamma_{seemingly}\ I\ \gamma_{John} : T \in \mathscr{A}(\mathscr{G}_{yield})$ (see Equation 9 on page 27). By definition of the object language of an ACG, its interpretation $\mathscr{G}_{yield}(\gamma_{13}) = \mathit{seemingly} + \mathit{John}$ is such that $\mathscr{G}_{yield}(\gamma_{13}) \in \mathscr{O}(\mathscr{G}_{yield})$.

However, $\gamma_{13} \notin \mathscr{O}(\mathscr{G}_{derived\ trees})$. Indeed, there is no c_5 such that $\mathscr{G}_{derived\ trees}(c_5) = \gamma_{13}$. A simple argument using the linearity of the interpretation shows that only $c_{seemingly}$ (once and only once), c_{John} (once and only once), and I_X can be used. But c_{John} can not combine with any of the other terms (none of them use the type NP). Consequently, $\mathit{seemingly} + \mathit{John} \notin \mathscr{O}(\mathscr{G}_{yield} \circ \mathscr{G}_{derived\ trees})$, as is expected from the TAG grammar.

4.3.3 $\mathscr{G}_{derived\ trees}$ abstract terms and generalized derivation trees

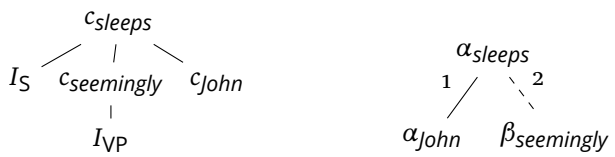
It is interesting to note that abstract terms of $\mathscr{G}_{derived\ trees}$ describe the way the encoding of trees in Σ_{trees} can combine. We can see this combination in terms such as $\gamma_5 = \gamma_{sleeps}\ I\ (\gamma_{seemingly}\ I)\ \gamma_{John}$, but it is in some sense an artifact of the definition we gave: γ_5 β-reduces to a tree that does not show this structure anymore. However, a term

such as $c_5 = c_{sleeps}\ I_{\mathsf{S}}\ (c_{seemingly}\ I_{\mathsf{VP}})\ c_{John}$ does not further β-reduce. Because we considered substitution as function application on arguments of atomic types and adjunction as function application on arguments of second-order types, c_5 keeps track of the adjunction of I_{S} on c_{sleeps}, of the adjunction of I_{VP} on $c_{seemingly}$, of the adjunction of the latter result on c_{sleeps}, and of the substitution of c_{John}. And the relation $\mathscr{G}_{derived\ trees}(c_5) = \gamma_5$ expresses the relation between the record of these operations and the resulting derived tree.

We can represent c_5 as a tree (see Figure 14(a)): each node corresponds to a constant, possibly applied to the terms represented by the children of the node. It makes explicit how similar to TAG derivation trees they are (Figure 14(b)). There is a one to one correspondence despite the following superficial differences:

- in the abstract term representation, the fake adjunctions (of I_X) are represented;
- instead of specifying the role of the arguments with the Gorn address, we set a specific order for the arguments.

Figure 14: Derivation representations



(a) Abstract term of $\mathscr{G}_{derived\ trees}$ (b) TAG Derivation tree

All the objects of a TAG grammar now have an ACG counterpart:

- terms of the abstract language of $\mathscr{G}_{derived\ trees}$ correspond to the TAG derivation trees;[20]
- terms of $\Lambda(\Sigma_{trees})$ that are in the object language of $\mathscr{G}_{derived\ trees}$ correspond to the TAG derived trees;
- terms of $\Lambda(\Sigma_{strings})$ that are in the object language of $\mathscr{G}_{yield} \circ \mathscr{G}_{derived\ trees}$ correspond to the TAG generated language.

[20] With some reservations that Section 6 clears up, though. See Remark 8 on the facing page.

(12) and (13) illustrate these correspondences for the abstract term $c_5 = c_{sleeps}\ I_{\mathsf{S}}\ (c_{seemingly}\ I_{\mathsf{VP}})\ c_{John}$ representing the derivation for the analysis of *John seemingly sleeps*.

$$\begin{aligned}&\mathscr{G}_{derived\ trees}(c_{sleeps}\ I_{\mathsf{S}}\ (c_{seemingly}\ I_{\mathsf{VP}})\ c_{John})\\ &\quad = \gamma_{sleeps}\ I\ (\gamma_{seemingly}\ I)\ \gamma_{John}\\ &\quad = \mathsf{S}_2\ (\mathsf{NP}_1\ \textit{John})\ (\mathsf{VP}_2\ (\mathsf{Adv}_1\ \textit{seemingly})\ (\mathsf{VP}_1\ (\mathsf{V}_1\ \textit{sleeps})))\\ &\qquad \text{by (5), p. 19}\end{aligned} \tag{12}$$

$$\begin{aligned}&\mathscr{G}_{yield} \circ \mathscr{G}_{derived\ trees}(c_{sleeps}\ I_{\mathsf{S}}\ (c_{seemingly}\ I_{\mathsf{VP}})\ c_{John})\\ &\quad = \textit{John} + (\textit{seemingly} + \textit{sleeps}) \quad \text{by (12) and (8) (p. 26)}\end{aligned} \tag{13}$$

Remark 7 ($\mathscr{G}_{derived\ trees}$ terms and description of trees)**.** Let us have a look at the $(X \multimap X)$ type of the argument of an abstract constant of $\mathscr{G}_{derived\ trees}$ and at its interpretation. In c_{sleeps} for instance, the argument with type $(\mathsf{VP} \multimap \mathsf{VP})$ is interpreted by the a variable of γ_{sleeps} (see Table 4 on page 29). The position of a in the term $S\ (\mathsf{S}_2\ s\ (a\ (\mathsf{VP}_1\ (\mathsf{V}_1\ \textit{sleeps}))))$ makes it explicit that the result of a applied to $(\mathsf{VP}_1\ (\mathsf{V}_1\ \textit{sleeps}))$, hence the latter term itself, is dominated by the second child of S_2 (the variable s being the first one). So, in some sense, the type $(\mathsf{VP} \multimap \mathsf{VP})$ of a corresponds to the *dominance constraint* between the node where a occurs (here the second child of S_2) and the root node of its argument (here $\mathsf{VP}_1\ (\mathsf{V}_1\ \textit{sleeps})$), as in the tree descriptions of Vijay-Shanker (1992): the root node of the argument of a is always dominated by the node where a occurs. In particular, replacing a by $\lambda^o x.x$ corresponds to having these two nodes identified.

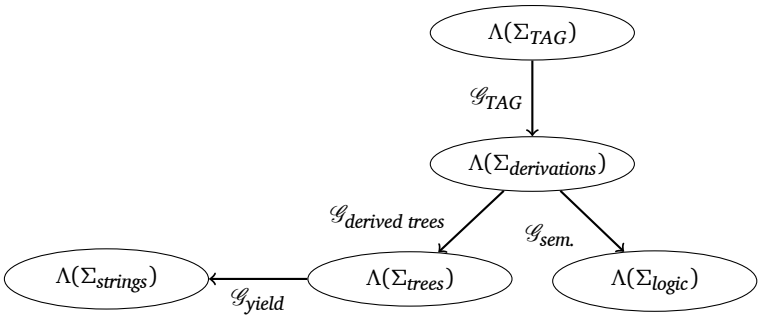
Remark 8 (Generalized derivations and TAG derivation trees)**.** It should be noted that $\mathscr{G}_{derived\ trees}$ and $\mathscr{G}_{yield} \circ \mathscr{G}_{derived\ trees}$ are *not* second-order ACGs. It means that the polynomial parsing results do not directly apply. But we know that TAG parsing is polynomial. So what is happening here?

The answer is that while $\mathscr{G}_{derived\ trees}$ constrains the string language more than $\mathscr{G}_{yield}$ does, it does not constrain it enough to generate only the corresponding TAG language. There is indeed no difference between the type of the encoding of an elementary tree of root S with a substitution node S and the encoding of an auxiliary tree with root and foot nodes labeled by S: it is $\mathsf{S} \multimap \mathsf{S}$ in both cases.

The solution, that we develop in Section 6, is to further control $\mathscr{A}(\mathscr{G}_{derived\ trees})$ with another ACG $\mathscr{G}_{TAG}$ such that $\mathscr{O}(\mathscr{G}_{TAG}) \subset$

$\mathscr{A}(\mathscr{G}_{derived\ trees})$ just as $\mathscr{G}_{derived\ trees}$ allows us to control $\mathscr{A}(\mathscr{G}_{yield})$. The general architecture is then the one that Figure 15 describes. But while this additional control is necessary to have a faithful encoding of TAG, it is not necessary to provide the semantic interpretation of the derivation trees (and may somewhat obfuscate it). That is why we first present the syntax-semantic interface we propose and delay the final encoding of TAG (that corresponds to the one of de Groote 2002) to Section 6.

Figure 15: ACG composition for control and syntax-semantics interface



5 SEMANTIC CONSTRUCTION

In the previous section, we defined a signature $\Sigma_{derivations}$ to represent derivation structures as terms of $\Lambda(\Sigma_{derivations})$. We now use this signature as pivot to transfer these structures into semantic representations. From a practical point of view, as mentioned in Section 3.1, it amounts to defining an ACG $\mathscr{G}_{sem.} = \langle \Sigma_{derivations}, \Sigma_{logic}, \mathscr{L}_{sem.}, \mathsf{S} \rangle$ and composing it with $\mathscr{G}_{derived\ trees}$ thanks to the shared abstract vocabulary $\Sigma_{derivations}$. The object vocabulary Σ_{logic} of this ACG is the vocabulary for defining the semantic representations. In this article, we use higher-order logic (and, more often than not, simply first-order logic). Other languages, such as description languages to express underspeficied representations (Bos 1995; Egg *et al.* 2001), modal logic languages, etc. are possible as well. But we want to focus on *how* to build semantic representations rather than on the semantic modelling of some linguistic phenomenon itself.

Table 5: The vocabulary Σ_{logic}

Logical constants			
$\wedge$	$: t \multimap t \multimap t$	$\vee$	$: t \multimap t \multimap t$
$\Rightarrow$	$: t \multimap t \multimap t$	$\neg$	$: t \multimap t$
$\exists$	$: (e \to t) \multimap t$	$\forall$	$: (e \to t) \multimap t$
Non-logical constants			
john	$: e$	**love, chase**	$: e \multimap e \multimap t$
sleep	$: e \multimap t$	**seemingly, usually, hence**	$: t \multimap t$
seem	$: e \multimap (e \multimap t) \multimap t$	**claim, think**	$: e \multimap t \multimap t$
WHO	$: (e \multimap t) \multimap t$	**big, black, dog, cat**	$: e \multimap t$

5.1 *A vocabulary for semantic representations:* Σ_{logic}

We first define the object vocabulary $\Sigma_{logic} = \langle A_{logic}, C_{logic}, \tau_{logic} \rangle$ as in Table 5 with $A_{logic} = \{e, t\}$ the atomic types for *entities* and *truth values* respectively. As usual, we write the λ-term $\exists(\lambda x.P)$ as $\exists x.P$. The same, *mutatis mutandis*, holds for $\forall$. Note that in this signature, we also use the non-linear implication, as a lot of semantic formulas (e.g., adjectives and quantifiers) use non linearity of entities. But we stay within the fragment of almost linear terms as only terms of atomic type are duplicated (see Definition 5 on page 11).[21]

5.2 *Generalized derivation-based interpretation*

The first step in defining $\mathscr{G}_{sem.}$, to interpret the abstract vocabulary $\Sigma_{derivations}$ into types and terms built on the object vocabulary Σ_{logic}, is to define the interpretation of the atomic types (S, VP...). We simply follow the standard interpretation of these syntactic types into the semantic types as proposed in (Montague 1973). This results in the interpretation described in Table 6(a). The interpretation of the constants follows, as in Table 6(b).[22] We do not repeat here the type of the constants c_{anchor} of $\Sigma_{derivations}$, nor the constraint that the image of this type has to be the type of $\mathscr{G}_{sem.}(c_{anchor})$ (e.g., the type of c_{John} is NP, hence $\mathscr{G}_{sem.}(c_{John}) : \mathscr{G}_{sem.}(\mathsf{NP}) = (e \to t) \multimap t$). But the reader can check that this proviso holds.

[21] For sake of simplicity, we use simplified extensional types e and t. A more accurate semantic relevant would require, for instance, intensional types.

[22] The types now also use the intuitionistic implication $\to$. This is required when variables that are abstracted over appear more than once in the semantic recipes. This is in particular the case for entities in quantified formulas, or in the semantics of intersective adjectives (see next Section).

Table 6: Interpretation by $\mathscr{G}_{sem.}$ of the $\Sigma_{derivations}$ vocabulary

$$\mathsf{S} :=_{sem.} t \qquad \mathsf{NP} :=_{sem.} (e \to t) \multimap t \qquad \mathsf{N} :=_{sem.} e \to t$$
$$\mathsf{VP} :=_{sem.} e \to t \qquad \mathsf{WH} :=_{sem.} (e \to t) \multimap t$$

(a) Interpretation of the atomic types

$$\begin{array}{ll}
c_{John} & :=_{sem.} \lambda^{o} P.P\ \mathbf{john} \\
c_{sleeps} & :=_{sem.} \lambda^{o} adv_{\mathsf{S}}\ adv_{\mathsf{VP}}\ subj.adv_{\mathsf{S}}\ (subj\ (adv_{\mathsf{VP}}\ (\lambda x.\mathbf{sleep}\ x))) \\
c_{seemingly} & :=_{sem.} \lambda^{o} adv_{mod}\ pred.adv_{mod}\ (\lambda x.\mathbf{seemingly}\ (pred\ x)) \\
c_{usually} & :=_{sem.} \lambda^{o} adv_{mod}\ pred.adv_{mod}\ (\lambda x.\mathbf{usually}\ (pred\ x)) \\
c_{hence} & :=_{sem.} \lambda^{o} adv_{mod}\ pred.adv_{mod}\ (\mathbf{hence}\ pred) \\
I_{\mathsf{S}} & :=_{sem.} \lambda^{o} x.x \\
I_{\mathsf{VP}} & :=_{sem.} \lambda^{o} x.x
\end{array}$$

(b) Interpretation of the constants

We let the reader check that for our favourite example, $c_5 :=_{sem.}$ **seemingly** (**sleep john**) as (14) shows.

$$\begin{aligned} c_5 &= c_{sleeps}\ I_{\mathsf{S}}\ (c_{seemingly}\ I_{\mathsf{VP}})\ c_{John} \\ &:=_{sem.} \mathbf{seemingly}\ (\mathbf{sleep\ john}) \end{aligned} \tag{14}$$

5.3 *From derivation dependencies to semantic dependencies*

We now turn to accounting for the mismatch between the dependencies as expressed in the derivation trees and in the logical semantic representations.

5.3.1 Long-Distance dependencies

The first mismatch we consider, in order to make explicit what exactly this mismatch refers to, relates to the classical examples (15–17).

(15) Paul claims John loves Mary

(16) Mary Paul claims John seems to love

(17) Who does Peter think Paul claims John seems to love

The TAG analysis relies on the elementary trees of Figure 16 and results in the derived tree and derivation tree of Figure 17 (next page) for (15). The mismatch appears in the contrast between the derivation tree where α_{loves} scopes over β_{claims} whereas the opposite scoping is to be expected from a semantic point of view. A similar effect occurs with (16) as the derivation tree, shown Figure 18(b) on page 36, makes

$\alpha_{to\ love}$ scope over both β_{claims} and β_{seems}, while semantically both should scope over the **love** predicate. Moreover, the derivation tree does not specify any scoping relation between the two auxiliary trees, whereas we expect **claim** to semantically scope over **seem**.

Finally, (17) and the derivation tree of Figure 19(b) (p. 37) illustrate how an element such as a *wh*-word can scope over a whole sentence and all its predicates while providing a semantic argument to the semantically "lowest" predicate (**love**).

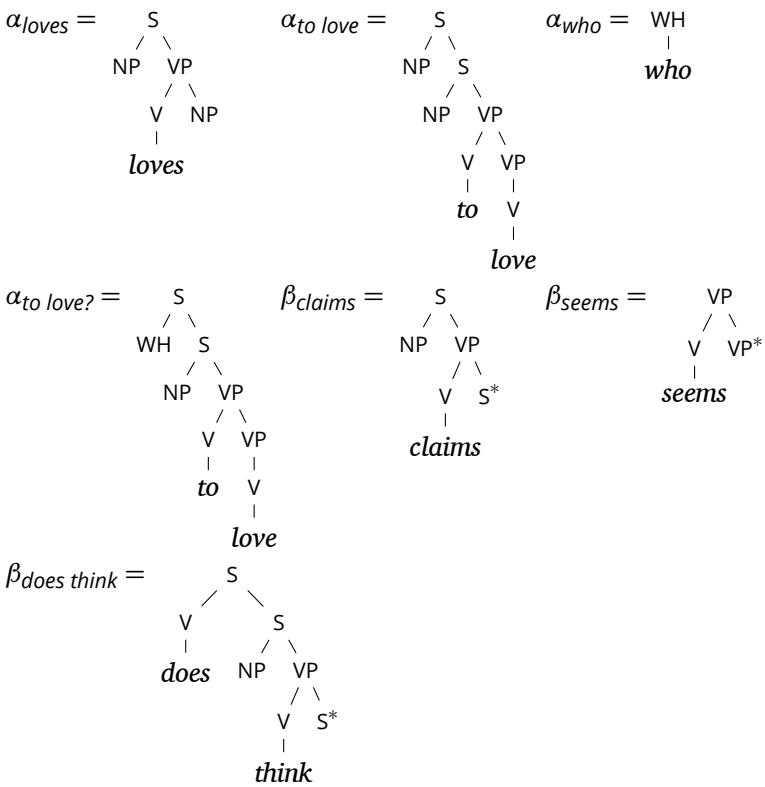
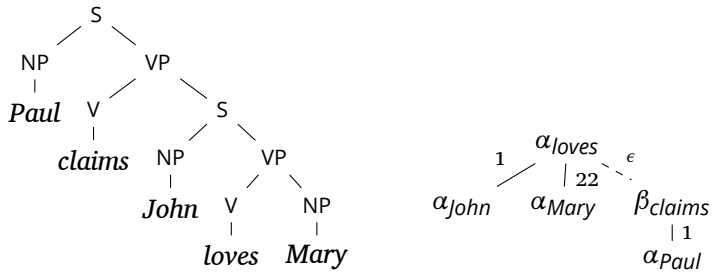


Figure 16: TAG elementary trees for long distance dependencies

To semantically account for these phenomena, we first extend $\Sigma_{derivations}$ and $\mathscr{G}_{derived\ trees}$ to represent the trees of Figure 16. Table 7 (p. 38) shows the new constants and their interpretations.[23] The terms c_{17}, c_{18}, and c_{19} in (18) represent the derivation trees of Figure 17(b),

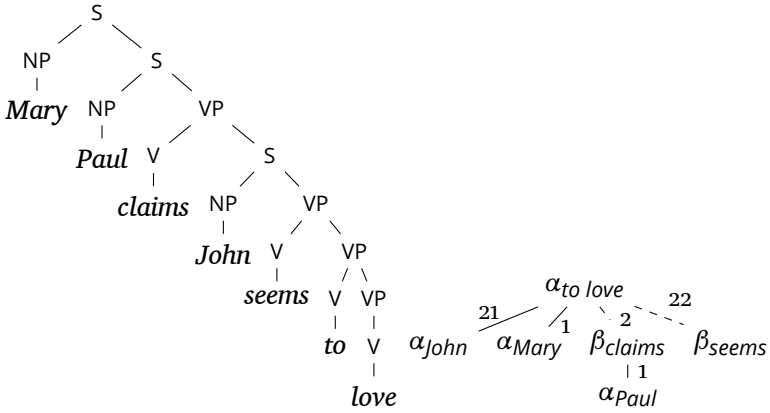
[23] Despite $\alpha_{to\ love}$ has two S nodes, its typing and its interpretation show that S adjunction is only allowed at the root node.

Figure 17: TAG analysis of *Paul claims John loves Mary*



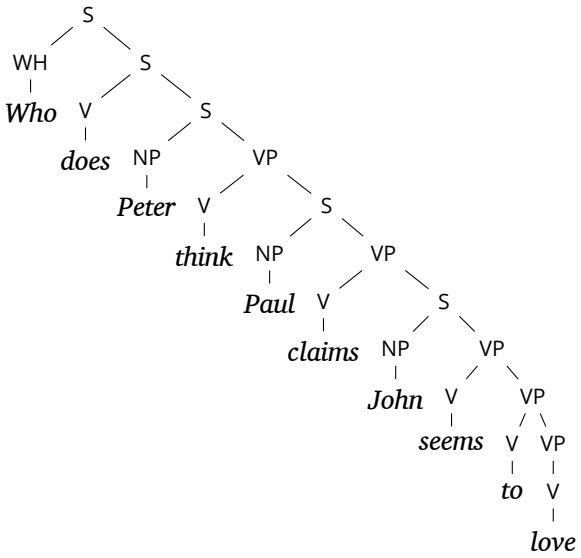
(a) Derived tree (b) Derivation tree

Figure 18: TAG analysis of *Mary Paul claims John seems to love*

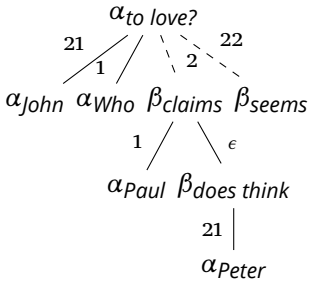


(a) Derived tree (b) Derivation tree

Figure 19:
TAG analysis of
Who does Peter think Paul claims John seems to love



(a) Derived tree



(b) Derivation tree

Table 7: Interpretation by $\mathscr{G}_{\textit{derived trees}}$

c_{who}	$: \mathsf{NP}$
	$:=_{\textit{derived trees}} \gamma_{who}$
γ_{who}	$\triangleq \mathsf{NP}_1\ \textit{who} : T$
c_{loves}	$: (\mathsf{S} \multimap \mathsf{S}) \multimap (\mathsf{VP} \multimap \mathsf{VP}) \multimap \mathsf{NP} \multimap \mathsf{NP} \multimap \mathsf{S}$
	$:=_{\textit{derived trees}} \gamma_{loves}$
γ_{loves}	$\triangleq \lambda^{\mathrm{o}} S\ a\ s\ o.S\ (\mathsf{S}_2\ s\ (a\ (\mathsf{VP}_2\ (\mathsf{V}_1\ \textit{loves})\ o)))$
	$: (T \multimap T) \multimap (T \multimap T) \multimap T \multimap T \multimap T$
$c_{to\ love}$	$: (\mathsf{S} \multimap \mathsf{S}) \multimap (\mathsf{VP} \multimap \mathsf{VP}) \multimap \mathsf{NP} \multimap \mathsf{NP} \multimap \mathsf{S}$
	$:=_{\textit{derived trees}} \gamma_{to\ love}$
$\gamma_{to\ love}$	$\triangleq \lambda^{\mathrm{o}} S\ a\ o\ s.\mathsf{S}_2\ o\ S\ ((\mathsf{S}_2\ s\ (a\ (\mathsf{VP}_2\ (\mathsf{V}_1\ \textit{to})\ (\mathsf{VP}_1\ (\mathsf{V}_1\ \textit{love}))))))$
	$: (T \multimap T) \multimap (T \multimap T) \multimap T \multimap T \multimap T$
$c_{to\ love?}$	$: (\mathsf{S} \multimap \mathsf{S}) \multimap (\mathsf{VP} \multimap \mathsf{VP}) \multimap \mathsf{NP} \multimap \mathsf{NP} \multimap \mathsf{S}$
	$:=_{\textit{derived trees}} \gamma_{to\ love?}$
$\gamma_{to\ love?}$	$\triangleq \lambda^{\mathrm{o}} S\ a\ w\ s.S\ (\mathsf{S}_2\ w\ (\mathsf{S}_2\ s\ (a\ (\mathsf{VP}_2\ (\mathsf{V}_1\ \textit{to})\ (\mathsf{VP}_1\ (\mathsf{V}_1\ \textit{love}))))))$
	$: (T \multimap T) \multimap (T \multimap T) \multimap T \multimap T \multimap T$
c_{claims}	$: (\mathsf{S} \multimap \mathsf{S}) \multimap (\mathsf{VP} \multimap \mathsf{VP}) \multimap \mathsf{NP} \multimap (\mathsf{S} \multimap \mathsf{S})$
	$:=_{\textit{derived trees}} \gamma_{claims}$
γ_{claims}	$\triangleq \lambda^{\mathrm{o}} S\ a\ s.\lambda^{\mathrm{o}} c.S\ (\mathsf{S}_2\ s\ (a\ (\mathsf{VP}_2\ (\mathsf{V}_1\ \textit{claims})\ c)))$
	$: (T \multimap T) \multimap (T \multimap T) \multimap T \multimap (T \multimap T)$
c_{seems}	$: (\mathsf{VP} \multimap \mathsf{VP}) \multimap (\mathsf{VP} \multimap \mathsf{VP})$
	$:=_{\textit{derived trees}} \gamma_{seems}$
γ_{seems}	$\triangleq \lambda^{\mathrm{o}} a\ v.a\ (\mathsf{VP}_2\ (\mathsf{V}_1\ \textit{seems})\ v)$
	$: (T \multimap T) \multimap (T \multimap T)$
$c_{does\ think}$	$: (\mathsf{S} \multimap \mathsf{S}) \multimap (\mathsf{VP} \multimap \mathsf{VP}) \multimap \mathsf{NP} \multimap (\mathsf{S} \multimap \mathsf{S})$
	$:=_{\textit{derived trees}} \gamma_{does\ think}$
$\gamma_{does\ think}$	$\triangleq \lambda^{\mathrm{o}} S\ a\ s.\lambda^{\mathrm{o}} c.\mathsf{S}_2\ (\mathsf{V}_1\ \textit{does})\ (S\ (\mathsf{S}_2\ s\ (a\ (\mathsf{VP}_2\ (\mathsf{V}_1\ \textit{think})\ c))))$
	$: (T \multimap T) \multimap (T \multimap T) \multimap T \multimap (T \multimap T)$

18(b), and 19(b) respectively. We leave it to the reader to check that the $\mathscr{G}_{\textit{derived trees}}$ interpretations of theses terms are the derived trees of the corresponding figures 17(a), 18(a), and 19(a) respectively.

$$
\begin{aligned}
c_{17} &= c_{loves}\ (c_{claims}\ I_{\mathsf{S}}\ I_{\mathsf{VP}}\ c_{Paul})\ I_{\mathsf{VP}}\ c_{John}\ c_{Mary} \\
c_{18} &= c_{to\ love}\ (c_{claims}\ I_{\mathsf{S}}\ I_{\mathsf{VP}}\ c_{Paul})\ (c_{seems}\ I_{\mathsf{VP}})\ c_{Mary}\ c_{John} \\
c_{19} &= c_{to\ love?}\ (c_{claims}\ (c_{does\ think}\ I_{\mathsf{S}}\ I_{\mathsf{VP}}\ c_{Peter})\ I_{\mathsf{VP}}\ c_{Paul}) \\
&\qquad\qquad (c_{seems}\ I_{\mathsf{VP}})\ c_{who}\ c_{John}
\end{aligned}
\tag{18}
$$

We now need to define the $\mathscr{G}_{sem.}$ interpretation that provides the expected semantic dependencies. Table 8 (p. 39) shows the lexical semantics fulfilling the requirements. The constant c_{loves} scopes over

$$
\begin{array}{ll}
c_{who} & :=_{sem.} \lambda^{o} P.\mathbf{WHO}\ P \\
c_{loves} & :=_{sem.} \lambda^{o} adv_{S}\ adv_{VP}\ subj\ obj.adv_{S}\ (subj\ (adv_{VP}\ (\lambda x.obj\ (\lambda y.\mathbf{love}\ x\ y)))) \\
c_{to\ love} & :=_{sem.} \lambda^{o} adv_{S}\ adv_{VP}\ obj\ subj.adv_{S}\ (subj\ (adv_{VP}\ (\lambda x.obj\ (\lambda y.\mathbf{love}\ x\ y)))) \\
c_{to\ love?} & :=_{sem.} \lambda^{o} adv_{S}\ adv_{VP}\ wh\ subj.wh\ (\lambda^{o} y.adv_{S}\ (subj\ (adv_{VP}\ (\lambda x.\mathbf{love}\ x\ y)))) \\
c_{claims} & :=_{sem.} \lambda^{o} adv_{S}\ adv_{VP}\ subj\ comp.adv_{S}\ (subj\ (adv_{VP}\ (\lambda x.\mathbf{claim}\ x\ comp))) \\
c_{seems} & :=_{sem.} \lambda^{o} mod\ pred.mod\ (\lambda x.\mathbf{seem}\ x\ pred) \\
c_{does\ think} & :=_{sem.} \lambda^{o} adv_{S}\ adv_{VP}\ subj\ comp.adv_{S}\ (subj\ (adv_{VP}\ (\lambda x.\mathbf{think}\ x\ comp)))
\end{array}
$$

Table 8: Interpretation by $\mathscr{G}_{sem.}$ of the $\Sigma_{derivations}$ vocabulary—long distance dependencies

c_{claims} in the term $c_{17} = c_{loves}\ (c_{claims}\ I_{S}\ I_{VP}\ c_{Paul})\ I_{VP}\ c_{John}\ c_{Mary}$, as does α_{loves} over α_{claims} in the derivation tree of Figure 17(b). However, looking at $\mathscr{G}_{sem.}(c_{loves})$ in Table 8, we observe that its first argument adv_{S} scopes over the **love** predicate. This argument actually corresponds to the meaning of the auxiliary tree adjoined at the S node of α_{loves}. When it is replaced by some actual value, for instance by the interpretation of $(c_{claims}\ I_{S}\ I_{VP}\ c_{Paul})$, the predicate in this actual value (here **claim**) then takes scope over **love**, achieving the desired effect. The same holds for the adv_{VP} argument.

However, in $\mathscr{G}_{sem.}(c_{to\ love?})$, the *wh* argument takes scope over the whole interpretation. This argument corresponds to the meaning of the constituent to be substituted at the WH node of $\alpha_{to\ love?}$ (see Figure 16), typically $\lambda^{o}P.\mathbf{WHO}\ P : (e \multimap t) \multimap t$, making **WHO** eventually take scope over all the other predicates.

Equation (19) on page 39 shows that the $\mathscr{G}_{sem.}$ interpretation builds the expected semantics with the required scope inversions. In terms of lexical semantics, the analysis and the account we propose are very close to the one proposed in synchronous TAG (Nesson 2009, p. 142).

$$
\begin{aligned}
c_{17} &= c_{loves}\ (c_{claims}\ I_{S}\ I_{VP}\ c_{Paul})\ I_{VP}\ c_{John}\ c_{Mary} \\
&:=_{sem.} \mathbf{claim\ paul\ (love\ john\ mary)} \\
c_{18} &= c_{to\ love}\ (c_{claims}\ I_{S}\ I_{VP}\ c_{Paul})\ (c_{seems}\ I_{VP})\ c_{Mary}\ c_{John} \\
&:=_{sem.} \mathbf{claim\ paul\ (seem\ john\ (}\lambda x.\mathbf{love}\ x\ \mathbf{mary))} \\
c_{19} &= c_{to\ love?}\ (c_{claims}\ (c_{does\ think}\ I_{S}\ I_{VP}\ c_{Peter})\ I_{VP}\ c_{Paul}) \\
&\qquad\qquad (c_{seems}\ I_{VP})\ c_{who}\ c_{John} \\
&:=_{sem.} \mathbf{WHO}\ (\lambda y.\mathbf{think\ peter\ (claim\ paul\ (seem\ john\ (}\lambda x.\mathbf{love}\ x\ y))))
\end{aligned}
\tag{19}
$$

Remark 9. The interpretation of c_{loves}, $c_{to\ love}$, and $c_{to\ love?}$ are very

close to each other. Building large scale grammars would require some factoring as can be done by lexical rules or meta-grammars (Candito 1996, 1999; Xia 2001; Xia *et al.* 2005; Crabbé *et al.* 2013). But in this article, we give the terms corresponding to the actual elementary trees that would be generated.

5.3.2 Quantification

We address in a similar way the mismatch between the scoping relation of verbal predicates over quantifiers in derivations trees and of quantifiers over verbal predicates in the logical semantic formulas. The trees of Figure 20 provide the TAG elementary trees for the TAG analysis for (20) (resp. for (21)) shown in Figure 21 (resp. in Figure 22).

(20) everyone loves someone

(21) every man loves some woman

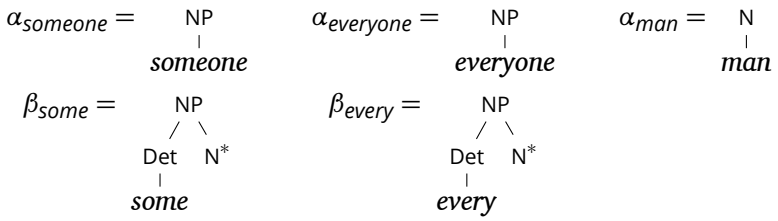
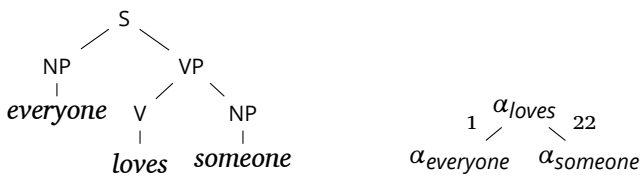


Figure 20: Determiners and quantifiers

Remark 10. We follow the standard TAG analyses for determiners (Abeillé 1990, 1993; XTAG Research Group 2001) where the latter adjoin on initial trees anchored by nouns, in order, in particular, to account for sequences of determiners (e.g., *all these ideas*) and mass nouns. While the auxiliary trees β_{some} and β_{every} of Figure 20 look unusual because the root node and the foot node do not have the same label, we can consider the label NP as a shorthand for the NP TAG category together with a positive (NP[+]) determiner feature. On the other hand, the N label is a shorthand for the NP TAG category together with a negative (NP[−]) determiner feature. Kasper *et al.* (1995) and others (Rogers 1999; Kahane *et al.* 2000) already noted that the differences between the features on the root node and on the foot node

could be reflected in allowing auxiliary trees to have different labels as root and foot nodes. While we discuss the modelling of features in TAG more generally in Section 7, the NP and N notations allow us to model the auxiliary trees of determiners with constants of the usual $\mathsf{N} \multimap \mathsf{NP}$ type (to be compared with a $\mathsf{NP}[-] \multimap \mathsf{NP}[+]$ type). While we could avoid introducing this distinction on the syntactic part of the TAG modelling, and have every node labelled with N, this distinction is semantically meaningful and records the different interpretation of N (as $e \to t$) and NP (as $(e \to t) \multimap t$, Table 6(a) on page 34).

The type of the constants modelling initial trees anchored by nouns has to be modified accordingly: it specifies that it requires an adjunction (an argument of type $(\mathsf{N} \multimap \mathsf{NP})$) before turning the noun into a noun phrase NP. So the type of constants (e.g., c_{man}) modelling initial trees anchored by nouns (e.g., α_{man}) is: $(\mathsf{N} \multimap \mathsf{NP}) \multimap \mathsf{NP}$. It corresponds to only keeping the constants that can indeed be used in actual derivations. For each noun, we could indeed introduce two constants with the following types: $(\mathsf{NP}[-] \multimap \mathsf{NP}[-]) \multimap \mathsf{NP}[-] = (\mathsf{N} \multimap \mathsf{N}) \multimap \mathsf{N}$ and $(\mathsf{NP}[-] \multimap \mathsf{NP}[+]) \multimap \mathsf{NP}[+] = (\mathsf{N} \multimap \mathsf{NP}) \multimap \mathsf{NP}$, but since there is no other constant that use $\mathsf{NP}[-] = \mathsf{N}$ as type for its arguments (i.e., substitution node),[24] we only keep the constant with the last type.



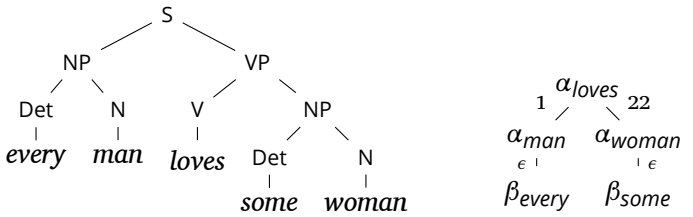
(a) Derived tree (b) Derivation tree

Figure 21: TAG analysis of *everyone loves someone*

The derivation trees of Figure 21 and 22 are again such that the elementary tree of the verb predicate dominates the other elementary trees, while the respective scopes of their semantic contributions are in the reverse order. To show how this apparent mismatch can be dealt with, we extend $\Sigma_{derivations}$ with the constants of Table 9. This table

[24] This of course depends on the grammar. In any case, if there were such a constant, it would not allow for performing first the adjunction of a determiner on the noun.

Figure 22: TAG analysis of *every man loves some woman*



(a) Derived tree (b) Derivation tree

Table 9: Interpretation of $\Sigma_{derivations}$ constants by $\mathscr{G}_{derived\ trees}$

Constants of $\Sigma_{derivations}$		Interpretation by $\mathscr{G}_{derived\ trees}$
c_{man}	$: (\mathsf{N} \multimap \mathsf{NP}) \multimap \mathsf{NP}$	$\lambda^{o} d.d\ (\mathsf{N}_1 man)$
$c_{someone}$	$: \mathsf{NP}$	$\mathsf{NP}_1\ someone$
$c_{everyone}$	$: \mathsf{NP}$	$\mathsf{NP}_1\ everyone$
c_{some}	$: \mathsf{N} \multimap \mathsf{NP}$	$\lambda^{o} n.\mathsf{NP}_2\ (\mathsf{Det}_1\ some)\ n$
c_{every}	$: \mathsf{N} \multimap \mathsf{NP}$	$\lambda^{o} n.\mathsf{NP}_2\ (\mathsf{Det}_1\ every)\ n$

also provides the interpretation of these constants by $\mathscr{G}_{derived\ trees}$ modelling the elementary trees of Figure 20. The terms of (22) belong to $\mathscr{A}(\mathscr{G}_{derived\ trees})$ and represent the derivation trees of Figure 21 and 22. Equation (23) shows they are interpreted as the derived trees of the same figures.

$$\begin{aligned} c_{21} &= c_{loves}\ I_{\mathsf{S}}\ I_{\mathsf{VP}}\ c_{everyone}\ c_{someone} \\ c_{22} &= c_{loves}\ I_{\mathsf{S}}\ I_{\mathsf{VP}}\ (c_{man}\ c_{every})\ (c_{woman}\ c_{some}) \end{aligned} \tag{22}$$

$$\begin{aligned} c_{21} &:=_{derived\ trees} \mathsf{S}_2\ (\mathsf{NP}_1\ everyone)\ (\mathsf{VP}_2\ (\mathsf{V}_1\ loves)\ (\mathsf{NP}_1\ someone)) \\ c_{22} &:=_{derived\ trees} \mathsf{S}_2\ (\mathsf{NP}_2\ (\mathsf{Det}_1\ every)\ (\mathsf{N}_1\ man)) \\ &\qquad (\mathsf{VP}_2\ (\mathsf{V}_1\ loves)\ (\mathsf{NP}_2\ (\mathsf{Det}_1\ some)\ (\mathsf{N}_1\ woman))) \end{aligned} \tag{23}$$

Then we extend $\mathscr{G}_{sem.}$ with the interpretations of these new constants of $\Sigma_{derivations}$ as terms of $\Lambda(\Sigma_{logic})$ (Table 10). The semantic interpretations of the terms c_{21} and c_{22} are then as expected, as (24) shows.

$$\begin{aligned} c_{21} &:=_{sem.} \forall x.(\mathbf{human}\ x) \Rightarrow (\exists y.(\mathbf{human}\ y) \wedge (\mathbf{love}\ x\ y)) \\ c_{22} &:=_{sem.} \forall x.(\mathbf{man}\ x) \Rightarrow (\exists y.(\mathbf{woman}\ y) \wedge (\mathbf{love}\ x\ y)) \end{aligned} \tag{24}$$

This shows how to use the derivation tree as a pivot towards the semantic representation of an expression. The (lexical) semantic interpretation of the terms labeling the nodes of the derivation

$$
\begin{array}{ll}
c_{man} & :=_{sem.} \lambda^{o} Q.\lambda^{o} q.Q\ \mathbf{man}\ q \\
c_{someone} & :=_{sem.} \lambda^{o} Q.\exists x.(\mathbf{human}\ x) \wedge (Q\ x) \\
c_{everyone} & :=_{sem.} \lambda^{o} Q.\forall x.(\mathbf{human}\ x) \Rightarrow (Q\ x) \\
c_{some} & :=_{sem.} \lambda^{o} P\ Q.\exists x.(P\ x) \wedge (Q\ x) \\
c_{every} & :=_{sem.} \lambda^{o} P\ Q.\forall x.(P\ x) \Rightarrow (Q\ x)
\end{array}
$$

Table 10: Interpretation by $\mathscr{G}_{sem.}$ of the $\Sigma_{derivations}$ vocabulary—quantification

tree encodes, when necessary, the inversion of the scope of the elements. This is reminiscent of the transformation of derivation trees into semantic dependency graphs of Candito and Kahane (1998) or Kallmeyer and Kuhlmann (2012). To this end, the latter implements a tree transduction-based approach (macro-tree transduction). Maskharashvili and Pogodalla (2013) discuss the relation with the present approach, relying on the encoding of macro-tree transduction within second-order ACGs (Yoshinaka 2006).

Remark 11. There are several ways to get the object scope reading. So far, the relative scopes of the subject and the object are bound to the semantic interpretation of the verb (see the semantic interpretation of c_{loves} in Table 8 on page 39). So a possibility consists in introducing a new constant c^{ows}_{loves} whose semantic interpretation reverses the scope, as Equation (25) shows.

$$c^{\mathrm{ows}}_{loves} :=_{sem.} \lambda^{o} adv_{\mathsf{S}}\ adv_{\mathsf{VP}}\ subj\ obj.adv_{\mathsf{S}}\ (obj\ (\lambda y.subj\ (adv_{\mathsf{VP}}\ (\lambda x.\mathbf{love}\ x\ y)))) \quad (25)$$

Another possibility, that would go beyond what is introduced in this article, would be to use Multi-Component TAG (MCTAG, Weir (1988)) as Williford (1993) proposed. In both cases, some care should be taken in order not to introduce spurious ambiguities. In Section 8.3, we provide another modelling that allows us to get this reading, and we relate it to other approaches.

5.3.3 Multiple adjunctions

The representation of TAG derivation trees as abstract terms of an ACG corresponds to the standard notion of derivation trees (Vijay-Shanker 1987). Schabes and Shieber (1994) call it *dependent* and advocate for an alternative *independent* notion. With dependent derivations, and in our approach, multiple adjunction on the same node is forbidden. So the analysis of (26) requires first the adjunction of β_{big} into β_{black}, and the adjunction of the result into α_{dog}. Figure 25(a) shows the resulting derivation tree. On the other hand, the independent adjunction shown

in Figure 25(b) only specifies that both adjectives adjoin at the N node of the initial tree, specifying both the derived tree of Figure 24(a) and the derived tree of Figure 24(b).

(26) big black dog

(27) black big dog

Figure 23: TAG elementary trees for adjectives

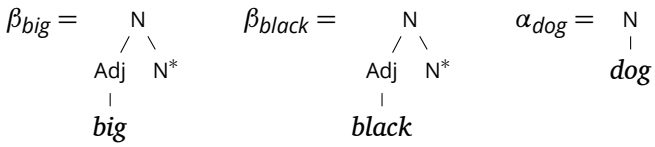
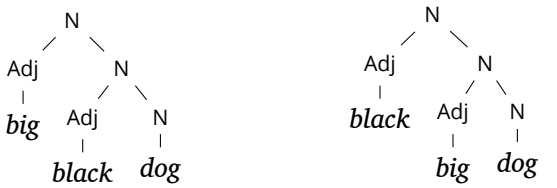


Figure 24: Alternative notions of derived trees



(a) Derived tree for (26)

(b) Derived tree for (27)

Figure 25: Alternative notions of derivation trees

(a) Dependent derivation tree

(b) Independent derivation tree

Schabes and Shieber (1994) present several arguments in favour of multiple adjunction for auxiliary trees encoding modification (as opposed to auxiliary trees encoding predication) and independent

derivations. We only discuss here the semantic argument they provide.[25] The main concern again has to do with the relation between derivation trees and semantic dependencies. The dependent derivation of Figure 25(a) reflects "cascaded modifications" of the head, rather than more expected "separate modifications", the latter being only available through multiple adjunction. We show that we can actually achieve this effect in our framework, without multiple adjunction, by specifying a semantic interpretation for adjectives that encodes such a behavior.

We consider the extension of $\Sigma_{derivations}$ with the constants of Table 11 on the following page and their interpretations by $\mathscr{G}_{derived\ trees}$ and $\mathscr{G}_{sem.}$ of Table 12 (p. 46). The types of the the constants modelling adjectives follow the types proposed for constants modelling nouns. The modification they introduce builds a NP from a N, and can itself take a $(\mathsf{N} \multimap \mathsf{NP})$ modification (adjunction) into account. Consequently, they are of type $(\mathsf{N} \multimap \mathsf{NP}) \multimap \mathsf{N} \multimap \mathsf{NP}$. As we did for the types of the constants modelling nouns, we could enumerate the possible types taking the determiner feature into account. Because the adjunction of an adjective does not change the determiner feature, its value at the root node of the auxiliary tree only depends on what is possibly adjoined to it. So we could possibly have four constants with the following types: $(\mathsf{NP}[-] \multimap \mathsf{NP}[-]) \multimap (\mathsf{NP}[-] \multimap \mathsf{NP}[-])$, $(\mathsf{NP}[-] \multimap \mathsf{NP}[+]) \multimap (\mathsf{NP}[-] \multimap \mathsf{NP}[+])$, $(\mathsf{NP}[+] \multimap \mathsf{NP}[-]) \multimap (\mathsf{NP}[+] \multimap \mathsf{NP}[-])$, and $(\mathsf{NP}[+] \multimap \mathsf{NP}[+]) \multimap \mathsf{NP}[+] \multimap \mathsf{NP}[+]$. But if we do not provide a term for a fake adjunction $(\mathsf{NP}[-] \multimap \mathsf{NP}[-])$, $(\mathsf{NP}[+] \multimap \mathsf{NP}[+])$, or $(\mathsf{NP}[+] \multimap \mathsf{NP}[-])$ (as in the example grammar we have), such terms can never be used in a S derivation. So we only keep the constants that have type $(\mathsf{NP}[-] \multimap \mathsf{NP}[+]) \multimap (\mathsf{NP}[-] \multimap \mathsf{NP}[+]) = (\mathsf{N} \multimap \mathsf{NP}) \multimap \mathsf{N} \multimap \mathsf{NP}$.

[25] The two other main arguments relate to the addition of adjoining constraints and to the addition of statistical parameters. Adding the latter to ACGs as a general framework is ongoing work, and the effects on the particular case of the TAG into ACG encoding will be considered from this perspective (Huot 2017). The argument about adjoining constraints that fail to escape intervening adjunctions is not related to the syntax-semantics interface and deserves a discussion that is beyond the scope of this article. For instance, the example of the [+] determiner feature (Section 5.3.2) that can percolate from the determiner (outmost adjunction) to the noun, despite the intervening adjunctions of the adjectives, shows that selectional restrictions can be implemented with long-distance effects.

Table 11: $\Sigma_{derivations}$ additional constants

$$
\begin{array}{ll}
c_{big} & : (\mathsf{N} \multimap \mathsf{NP}) \multimap \mathsf{N} \multimap \mathsf{NP} \\
c_{black} & : (\mathsf{N} \multimap \mathsf{NP}) \multimap \mathsf{N} \multimap \mathsf{NP} \\
c_{dog} & : (\mathsf{N} \multimap \mathsf{NP}) \multimap \mathsf{NP}
\end{array}
$$

Table 12: Interpretation by $\mathscr{G}_{sem.}$ and $\mathscr{G}_{derived\ trees}$ of the $\Sigma_{derivations}$ vocabulary—multiple adjunction

$$
\begin{array}{ll}
c_{big} & :=_{derived\ trees} \lambda^o a\ n.a\ (\mathsf{N}_2\ (\mathsf{Adj}_1\ \mathit{big})\ n) \\
c_{black} & :=_{derived\ trees} \lambda^o a\ n.a\ (\mathsf{N}_2\ (\mathsf{Adj}_1\ \mathit{black})\ n) \\
c_{dog} & :=_{derived\ trees} \lambda^o d.d\ (\mathsf{N}_1\ \mathit{dog}) \\
c_{big} & :=_{sem.} \lambda^o Q\ n.\lambda^o q.Q\ (\lambda x.(n\ x) \wedge (\mathbf{big}\ x))\ q \\
c_{black} & :=_{sem.} \lambda^o Q\ n.\lambda^o q.Q\ (\lambda x.(n\ x) \wedge (\mathbf{black}\ x))\ q \\
c_{dog} & :=_{sem.} \lambda^o Q.\lambda^o q.Q\ \mathbf{dog}\ q
\end{array}
$$

(28) shows the interpretations of the term $c_{25} : (\mathsf{N} \multimap \mathsf{NP}) \multimap \mathsf{NP}$ (an expression missing a determiner of type $(\mathsf{N} \multimap \mathsf{NP})$ to provide a NP) that encodes the derivation tree of Figure 25(a) (p. 44). The interpretation by $\mathscr{G}_{sem.}$ indeed provides a separate modification of the same variable x as argument both of **big** and **black** (a similar account would also be available in synchronous TAG).

$$
\begin{aligned}
c_{25} &= \lambda^o D.c_{dog}\ (c_{black}\ (c_{big}\ D)) \\
&:=_{derived\ trees} \lambda^o D.D\ (\mathsf{N}_2\ (\mathsf{Adj}_1\ \mathit{big})\ (\mathsf{N}_2\ (\mathsf{Adj}_1\ \mathit{black})\ (\mathsf{N}_1\ \mathit{dog}))) \qquad (28) \\
&:=_{sem.} \lambda^o D.\lambda^o q.D\ (\lambda x.((\mathbf{big}\ x) \wedge (\mathbf{black}\ x)) \wedge (\mathbf{dog}\ x))\ q
\end{aligned}
$$

Remark 12. By not introducing a constant $I_{\mathsf{N}} : \mathsf{N} \multimap \mathsf{NP}$ in $\Sigma_{derivations}$, we require actual adjunctions of determiners (of type $(\mathsf{N} \multimap \mathsf{NP})$, e.g., c_{some}) on nouns or on nouns modified by adjectives.

6 COMPLETING THE TAG INTO ACG ENCODING

So far, the abstract signatures we used, in particular $\Sigma_{derivations}$, introduce constants that are of order strictly greater than 2. This comes in particular from the modelling of auxiliary trees as functions (typically of type $X \multimap X$), hence from having constants of higher-order type modelling the ability for a tree of getting an auxiliary tree as argument. From a theoretical point of view, we know this encoding cannot faithfully model TAG: TAG languages are polynomially parsable, and 3rd-order ACGs can generate languages in NP. Remark 8 (p. 31) gives an example of an unexpected result of this encoding: there is no

way to distinguish the $\mathsf{s} \multimap \mathsf{s}$ type of an abstract constant modelling an auxiliary tree of foot node s from an abstract constant modelling an elementary tree of root s with a substitution node s.

6.1 *A vocabulary for TAG derivations: the Σ_{TAG} signature*

In order to allow for the distinction between these types, we introduce *atomic types* (e.g., s_A) that will be interpreted as the modifier types of the constants modelling auxiliary trees. So in addition to the ACG $\mathscr{G}_{derived\ trees} = \langle \Sigma_{derivations}, \Sigma_{trees}, \mathscr{L}_{derived\ trees}, \mathsf{s} \rangle$, we also define a higher-order signature $\Sigma_{TAG} = \langle A_{TAG}, C_{TAG}, \tau_{TAG} \rangle$ such that $A_{TAG} = A_{derivations} \cup \bigcup_{X \in A_{derivations}} X_A$.

For any $\delta_{anchor} \in \mathscr{I} \cup \mathscr{A}$ an elementary tree of $\mathscr{G}$, C_{anchor} is a constant in C_{TAG} with type $X_A^1 \multimap \cdots \multimap X_A^n \multimap Y^1 \multimap \cdots \multimap Y^m \multimap \alpha$ where:

- the X^i are the labels of the n internal nodes of δ_{anchor} labeled with a non-terminal where an adjunction is possible (by convention we use the breadth-first traversal);
- the Y^i are the labels of the m leaves of δ_{anchor} labeled with non-terminals, *not counting the foot node if δ_{anchor} is an auxiliary tree*, of δ_{anchor} (by convention, we use the left-right order traversal);
- let Z be the label of the root node of δ_{anchor}. $\alpha = Z$ if $\delta_{anchor} \in \mathscr{I}$ is an initial tree, and $\alpha = Z_A$ with Z the label of the foot node if $\delta_{anchor} \in \mathscr{A}$ is an auxiliary tree.

From the elementary trees of Figure 2 (p. 8), for instance, we get the constants typed as (29) shows.

$$\begin{aligned} C_{sleeps} &: \mathsf{s}_A \multimap \mathsf{VP}_A \multimap \mathsf{NP} \multimap \mathsf{s} \\ C_{John} &: \mathsf{NP} \\ C_{seemingly} &: \mathsf{VP}_A \multimap \mathsf{VP}_A \end{aligned} \tag{29}$$

Moreover, for each non-terminal X of the TAG grammar where an adjunction can occur, we also define $I_X : X_A$. These constants play a similar role as the I_X constants in $\Sigma_{derivations}$: they are used when a TAG derivation does not involve adjunctions on sites where it would be possible to have them.

Then the set of typed constants of Σ_{TAG} is $C_{TAG} = \{C_{anchor} | \ \delta_{anchor} \in \mathscr{I} \cup \mathscr{A}\} \cup \{I_X | \ X \in V^0\}$ and τ_{TAG} is the associated typing function de-

fined as above. The typing provided by Σ_{TAG} now distinguishes the type of the encoding of an elementary tree of root S with a substitution node S (type $\mathsf{S} \multimap \mathsf{S}$) and the encoding of an auxiliary tree with root and foot nodes labeled by S (type S_A, see Remark 8, p. 31).

We now need to relate the terms of $\Lambda(\Sigma_{TAG})$ to the terms of $\Lambda(\Sigma_{derivations})$ by a suitable interpretation.

6.2 *Interpreting Σ_{TAG} into $\Lambda(\Sigma_{derivations})$: the $\mathscr{G}_{TAG}$ ACG*

We now can relate Σ_{TAG} and $\Lambda(\Sigma_{derivations})$ through a new ACG $\mathscr{G}_{TAG} = \langle \Sigma_{TAG}, \Sigma_{derivations}, \mathscr{L}_{TAG}, \mathsf{S} \rangle$ where $\mathscr{L}_{TAG}$ is such that:

- for all $\alpha \in A_{TAG}$, if $\alpha = X_A$ then $\mathscr{L}_{TAG}(\alpha) = \mathscr{L}_{TAG}(X_A) = X \multimap X$, otherwise $\alpha = X \in A_{derivations}$ and $\mathscr{L}_{TAG}(\alpha) = \mathscr{L}_{TAG}(X) = X$;
- for all $C_{anchor} \in C_{TAG}$, $\mathscr{L}_{TAG}(C_{anchor}) = c_{anchor}$.

By construction of the constants $c_{anchor} \in C_{derivations}$ (Section. 4.3), and by construction of the constants $C_{anchor} \in C_{TAG}$, $\mathscr{L}_{TAG}$ is well defined.

Table 13 (p. 50) sums up the constants corresponding to the elementary trees introduced so far as well as their interpretations. Because constants are interpreted as constants, the terms of $\Lambda(\Sigma_{TAG})$ and their interpretations are isomorphic. However, some terms of $\Lambda(\Sigma_{derivations})$ have no antecedent by $\mathscr{L}_{TAG}$. For instance, the term $c_{sleeps}\ (c_{matters}\ I_{\mathsf{VP}})\ I_{\mathsf{VP}}\ c_{John} : \mathsf{S} \in \mathscr{A}(\mathscr{G}_{derived\ trees})$, where $c_{matters} : (\mathsf{VP} \multimap \mathsf{VP}) \multimap \mathsf{S} \multimap \mathsf{S}$ corresponds to the initial tree $\alpha_{matters}$ of Figure 26, as in *To arrive on time matters considerably* (see XTAG Research Group 2001, Section 6.31), has no antecedent. This is because the type of $c_{matters}\ I_{\mathsf{VP}} : \mathsf{S} \multimap \mathsf{S}$ in $\mathscr{G}_{derived\ trees}$, while it encodes an initial tree, is the same as the type of a term encoding an adjunction on a S node (see Remark 8 p. 31). But this is not true anymore at the level of $\mathscr{G}_{TAG}$ where $C_{matters}\ I_{\mathsf{VP}} : \mathsf{S} \multimap \mathsf{S}$ but the type of a term encoding an adjunction on a S node is now S_A.

So Σ_{TAG} allows us to add control on the admissible derivation structures that $\Sigma_{derivations}$ can provide. The general architecture is now the one of Figure 15 (p. 32). Moreover, this architecture allows us to provide, by function composition, a semantic interpretation to the constants of Σ_{TAG}. Interestingly, this semantic interpretation derives from the more general constructions that $\Sigma_{derivations}$ enables.

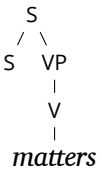


Figure 26: Initial tree for *matters* (the S leaf is a substitution node)

Σ_{TAG} strictly follows the abstract signature definition proposed in (de Groote 2002) to encode the syntactic part of TAGs. The correctness of our encoding follows from the fact that the ACG $\mathscr{G}_{derived\ trees} \circ \mathscr{G}_{TAG}$ we get by the function composition is the ACG defined by de Groote (2002).

Remark 13. Because the modelling of adjunction is now controlled by the interpretation of the types X_A from Σ_{TAG}, we see that we can have more freedom in the type that is given in $\Sigma_{derivations}$. For instance, we can set $\mathsf{N}_A :=_{TAG} \mathsf{N} \multimap \mathsf{NP}$. We can use even more complex interpretations if it helps explaining the semantic interpretation. For instance, in Section 7.2 we introduce a type $\mathsf{S}'_A :=_{TAG} (\mathsf{NP} \multimap \mathsf{S}) \multimap \mathsf{S}$ to model control verbs.

We can now consider the ACGs $\mathscr{G}_{yield} \circ \mathscr{G}_{derived\ trees} \circ \mathscr{G}_{TAG}$, that interprets terms of $\Lambda(\Sigma_{TAG})$ into strings, and the ACG $\mathscr{G}_{sem.} \circ \mathscr{G}_{TAG}$, that interprets terms of $\Lambda(\Sigma_{TAG})$ into logical formulas (see Figure 15, p. 32). Because Σ_{TAG} is second-order, these two ACGs are second-order (while $\mathscr{G}_{yield} \circ \mathscr{G}_{derived\ trees}$ and $\mathscr{G}_{sem.}$ are not, since $\Sigma_{derivations}$ is not second-order). Hence the parsing result apply and we may parse terms with ACGs that have Σ_{TAG} as abstract vocabulary, in particular with the ACG toolkit. The ACG example files we provide can, for instance, parse the string *every* + *big* + *black* + *dog* + *usually* + *barks*. It can also parse the logical formula $\forall x.(((\mathbf{dog}\ x) \wedge (\mathbf{black}\ x)) \wedge (\mathbf{big}\ x)) \Rightarrow (\mathbf{usually}\ (\mathbf{bark}\ x))$. Note that, as a λ-term, a logical formula can generally not be replaced by a logically equivalent formula. This is an instance of the problem of logical-form equivalence (Shieber 1993) that will need to be addressed, for instance using sets of λ-terms as input (Kanazawa 2017, Section 4.2). More examples are available in the example files.

Table 13: Σ_{TAG} constants and their interpretation by $\mathscr{L}_{TAG}$

Types and constants of Σ_{TAG}		Their interpretations in $\Lambda(\Sigma_{derivations})$	
NP		NP	
S		S	
VP		VP	
N		N	
WH		WH	
VP_A		$\mathsf{VP} \multimap \mathsf{VP}$	
S_A		$\mathsf{S} \multimap \mathsf{S}$	
N_A		$\mathsf{N} \multimap \mathsf{NP}$	
C_{John}	$: \mathsf{NP}$	c_{John}	$: \mathsf{NP}$
C_{sleeps}	$: \mathsf{S}_A \multimap \mathsf{VP}_A \multimap \mathsf{NP} \multimap \mathsf{S}$	c_{sleeps}	$: (\mathsf{S} \multimap \mathsf{S}) \multimap (\mathsf{VP} \multimap \mathsf{VP}) \multimap \mathsf{NP} \multimap \mathsf{S}$
$C_{seemingly}$	$: \mathsf{VP}_A \multimap \mathsf{VP}_A$	$c_{seemingly}$	$: (\mathsf{VP} \multimap \mathsf{VP}) \multimap \mathsf{VP} \multimap \mathsf{VP}$
$C_{usually}$	$: \mathsf{VP}_A \multimap \mathsf{VP}_A$	$c_{usually}$	$: (\mathsf{VP} \multimap \mathsf{VP}) \multimap \mathsf{VP} \multimap \mathsf{VP}$
C_{hence}	$: \mathsf{S}_A \multimap \mathsf{S}_A$	c_{hence}	$: (\mathsf{S} \multimap \mathsf{S}) \multimap \mathsf{S} \multimap \mathsf{S}$
I_{S}	$: \mathsf{S}_A$	I_{S}	$: \mathsf{S} \multimap \mathsf{S}$
I_{VP}	$: \mathsf{VP}_A$	I_{VP}	$: \mathsf{VP} \multimap \mathsf{VP}$
$C_{matters}$	$: \mathsf{VP}_A \multimap \mathsf{S} \multimap \mathsf{S}$	$c_{matters}$	$: (\mathsf{VP} \multimap \mathsf{VP}) \multimap \mathsf{S} \multimap \mathsf{S}$
C_{who}	$: \mathsf{NP}$	c_{who}	$: \mathsf{NP}$
C_{loves}	$: \mathsf{S}_A \multimap \mathsf{VP}_A \multimap \mathsf{NP} \multimap \mathsf{NP}$	c_{loves}	$: (\mathsf{S} \multimap \mathsf{S}) \multimap (\mathsf{VP} \multimap \mathsf{VP}) \multimap \mathsf{NP} \multimap \mathsf{NP} \multimap \mathsf{S}$
$C_{to\ love}$	$: \mathsf{S}_A \multimap \mathsf{VP}_A \multimap \mathsf{NP} \multimap \mathsf{NP}$	$c_{to\ love}$	$: (\mathsf{S} \multimap \mathsf{S}) \multimap (\mathsf{VP} \multimap \mathsf{VP}) \multimap \mathsf{NP} \multimap \mathsf{NP} \multimap \mathsf{S}$
$C_{to\ love?}$	$: \mathsf{S}_A \multimap \mathsf{VP}_A \multimap \mathsf{NP} \multimap \mathsf{NP}$	$c_{to\ love?}$	$: (\mathsf{S} \multimap \mathsf{S}) \multimap (\mathsf{VP} \multimap \mathsf{VP}) \multimap \mathsf{NP} \multimap \mathsf{NP} \multimap \mathsf{S}$
C_{claims}	$: \mathsf{S}_A \multimap \mathsf{VP}_A \multimap \mathsf{NP} \multimap \mathsf{S}_A$	c_{claims}	$: (\mathsf{S} \multimap \mathsf{S}) \multimap (\mathsf{VP} \multimap \mathsf{VP}) \multimap \mathsf{NP} \multimap (\mathsf{S} \multimap \mathsf{S})$
C_{seems}	$: \mathsf{VP}_A \multimap \mathsf{VP}_A$	c_{seems}	$: (\mathsf{VP} \multimap \mathsf{VP}) \multimap (\mathsf{VP} \multimap \mathsf{VP})$
$C_{does\ think}$	$: \mathsf{S}_A \multimap \mathsf{VP}_A \multimap \mathsf{NP} \multimap \mathsf{S}_A$	$c_{does\ think}$	$: (\mathsf{S} \multimap \mathsf{S}) \multimap (\mathsf{VP} \multimap \mathsf{VP}) \multimap \mathsf{NP} \multimap (\mathsf{S} \multimap \mathsf{S})$
C_{man}	$: \mathsf{N}_A \multimap \mathsf{NP}$	c_{man}	$: (\mathsf{N} \multimap \mathsf{NP}) \multimap \mathsf{NP}$
$C_{someone}$	$: \mathsf{NP}$	$c_{someone}$	$: \mathsf{NP}$
$C_{everyone}$	$: \mathsf{NP}$	$c_{everyone}$	$: \mathsf{NP}$
C_{some}	$: \mathsf{N}_A$	c_{some}	$: \mathsf{N} \multimap \mathsf{NP}$
C_{every}	$: \mathsf{N}_A$	c_{every}	$: \mathsf{N} \multimap \mathsf{NP}$
C_{big}	$: \mathsf{N}_A \multimap \mathsf{N}_A$	c_{big}	$: (\mathsf{N} \multimap \mathsf{NP}) \multimap \mathsf{N} \multimap \mathsf{NP}$
C_{black}	$: \mathsf{N}_A \multimap \mathsf{N}_A$	c_{black}	$: (\mathsf{N} \multimap \mathsf{NP}) \multimap \mathsf{N} \multimap \mathsf{NP}$
C_{dog}	$: \mathsf{N}_A \multimap \mathsf{NP}$	c_{dog}	$: (\mathsf{N} \multimap \mathsf{NP}) \multimap \mathsf{NP}$

7 ADJOINING CONSTRAINTS AND FEATURES

It is part of the TAG formalism to specify if an internal node may, may not, or has to receive any adjunction. The latter case is called an *obligatory adjoining (OA)* constraint. In case an internal node can be subject to an adjunction operation, it is also possible to specify a restricted set of auxiliary trees, with relevant root and foot nodes, that can adjoin. This constraint is called a *selective adjoining (SA)* constraint. There are different ways to specify such constraints in TAG.

One is to add *features* to the formalism. ACGs do not provide a concise way to express the abstract representation of type constraints that features offer. There have been some proposal with dependent types (de Groote and Maarek 2007; de Groote *et al.* 2007; Pompigne 2013), but the underlying calculus does not have the expected good properties. So selection restriction has to be expressed by introducing as many types as necessary (see Section 5.3.2 for determiners and Section 5.3.3 for adjectives). To avoid the drawback of a growing size of the grammar, the addition of features, in particular morph-syntactic ones, to ACGs remains desirable.

Note, however, that we do not want to consider features that are used to model the syntax-semantics interface (Kallmeyer and Romero 2004, 2008), since we use the interpretation of derivation trees instead. We discuss the relation between the two approaches in Section 7.3.

7.1 *Obligatory adjoining constraints*

Section 5.3.3 presents an instance of an adjoining constraint, namely an Obligatory Adjoining (OA) constraint. In order to form a NP, a noun needs to be adjoined (directly or through adjectival modifications) a determiner of type $(\mathsf{N} \multimap \mathsf{NP})$. The obligatory nature of the adjunction is reflected by the fact that the abstract vocabulary does not provide any constant $I_{\mathsf{N}} : \mathsf{N} \multimap \mathsf{NP}$ simulating a fake adjunction.

7.2 *Selective adjoining constraints*

In Section 5.3.2, we saw an instance of using features in a TAG analysis: noun phrases can receive a determiner feature $[+]$ or $[-]$ indicating whether they are determined. The ACG way to account for this distinction indeed consists in introducing different (atomic) types. This corresponds to specifying local adjunction constraints by *enumeration* as in TAG and contrary to Feature-based TAG (FTAG, Vijay-Shanker and Joshi 1988; 1991).

As noticed in Remark 13 (p. 49), the key here is to model auxiliary trees using an atomic type X_A, so that the ACG is second-order, and to interpret this type as a functional type $X \multimap X'$ of $\Sigma_{derivations}$, without the actual requirement that X and X' are atomic or that $X = X'$. We illustrate such an encoding with the TAG analysis of control verbs.

TAG analyzes a sentence such as (30) with an adjunction of the

subject control verb *wants* on the reduced clause *to sleep* as Figure 28 shows. Figure 27 presents the elementary trees of the control verb and of the infinitive clause. This is similar to representing infinitive clauses as clauses without subject (Abeillé 2002). For sake of simplicity, we directly represent such a clause as an elementary tree with a PRO node.

Control and adjunction are enforced using a control feature on the S root node of the complement tree (`control` in XTAG (XTAG Research Group 2001, p. 98), or some semantic index `idx` in (Gardent and Parmentier 2005; Gardent 2008)) that is to be provided by the foot node of the auxiliary tree (the control verb) which is adjoined. Moreover, in the auxiliary tree, the control feature on the foot node is co-indexed with a control feature on the subject NP (for subject control, as in (30)) or on the object NP (for object control). Figure 28 shows the derived and the derivation trees for (30).

(30) John wants to sleep

Figure 27: Elementary trees for control verbs

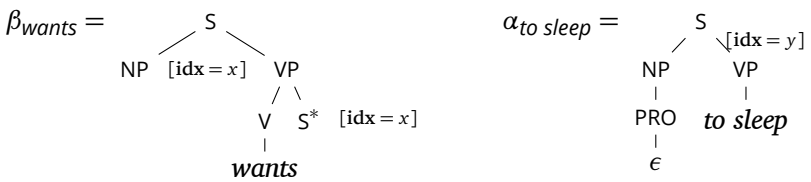
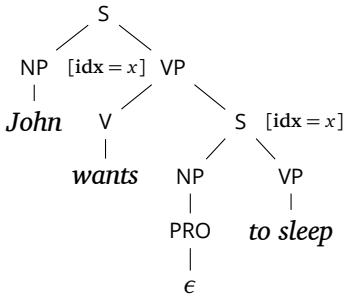
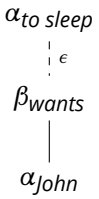


Figure 28: TAG analysis of control



(a) Derived tree



(b) Derivation tree

In our ACG encoding, we model S nodes with a control feature with the functional type NP ⊸ S, expressing that such a clause is missing its subject. Consequently, the type of the constant c_{wants} that models the auxiliary tree β_{wants} is (S ⊸ S) ⊸ (VP ⊸ VP) ⊸ NP ⊸ (NP ⊸ S) ⊸ S.

Types and constants of Σ_{TAG}		Their interpretations in $\Lambda(\Sigma_{derivations})$	
S'_A		$(\mathsf{NP} \multimap \mathsf{S}) \multimap \mathsf{S}$	
C_{wants}	$: \mathsf{S}_A \multimap \mathsf{VP}_A \multimap \mathsf{NP} \multimap \mathsf{S}'_A$	c_{wants}	$: (\mathsf{S} \multimap \mathsf{S}) \multimap (\mathsf{VP} \multimap \mathsf{VP}) \multimap \mathsf{NP} \multimap (\mathsf{NP} \multimap \mathsf{S}) \multimap \mathsf{S}$
$C_{to\ sleep}$	$: \mathsf{S}'_A \multimap \mathsf{S}$	$c_{to\ sleep}$	$: ((\mathsf{NP} \multimap \mathsf{S}) \multimap \mathsf{S}) \multimap \mathsf{S}$

Table 14: $\mathscr{G}_{TAG}$ extension—control verbs

$$c_{wants} :=_{derived\ trees} \lambda^{o} adv_{\mathsf{S}}\ adv_{\mathsf{VP}}\ subj\ pred.\ adv_{\mathsf{S}}\ (\mathsf{S}_2\ subj\ (adv_{\mathsf{VP}}\ (\mathsf{VP}_2\ (\mathsf{V}_1\ wants)\ (pred\ (\mathsf{PRO}_1\ \epsilon)))))$$
$$c_{to\ sleep} :=_{derived\ trees} \lambda^{o} cont.cont(\lambda^{o} subj.\mathsf{S}_2\ (\mathsf{NP}_1\ subj)\ (\mathsf{VP}_2\ (\mathsf{V}_1\ to)\ (\mathsf{VP}_1\ sleep)))$$

Table 15: $\mathscr{G}_{derived\ trees}$ extension—control verbs

$$\mathbf{want} : e \multimap t \multimap t$$
$$c_{wants} :=_{sem.} \lambda^{o} adv_{\mathsf{S}}\ adv_{\mathsf{VP}}\ subj\ pred.adv_{\mathsf{S}}\ (subj\ (adv_{\mathsf{VP}}\ (\lambda x.\mathbf{want}\ x\ (pred\ (\lambda^{o} P.P\ x)))))$$
$$c_{to\ sleep} :=_{sem.} \lambda^{o} cont.cont(\lambda^{o} subj.subj\ (\lambda x.\mathbf{sleep}\ x))$$

Table 16: Σ_{logic} and $\mathscr{G}_{sem.}$ extension—control verbs

The end part $(\mathsf{NP} \multimap \mathsf{S}) \multimap \mathsf{S}$ of this type corresponds to the functional interpretation of the adjunction of control verbs, modeled at the Σ_{TAG} level with the atomic type S'_A. The difference between the type $(\mathsf{NP} \multimap \mathsf{S})$ of the argument and the type S of the result corresponds to the different feature set attached to the root node and to the foot node of the auxiliary tree.

Then we model the feature sharing between the subject NP and the S foot node of the control verb in the semantic interpretation of the latter, as the first line of Table 16 shows: the first argument x of $\mathbf{want}\ x\ (pred\ (\lambda^{o} P.P\ x))$ also appears (type raised as $(\lambda^{o} P.P\ x)$) as argument of *pred*, the latter corresponding to the semantics of the infinitive clause without subject.

Let C_{28} of $\Lambda(\Sigma_{TAG})$ in (31) represent the derivation tree of Figure 28, and let c_{28} be its interpretation in $\Lambda(\Sigma_{derivations})$. We can further interpret c_{28} in $\Lambda(\Sigma_{trees})$ (resp. in $\Lambda(\Sigma_{logic})$) in order to have a

term representing the associated derived tree (resp. semantics).

$$
\begin{aligned}
C_{28} &= C_{\textit{to sleep}}\ (C_{\textit{wants}}\ I_{\mathsf{S}}\ I_{\mathsf{VP}}\ C_{\textit{John}}) \\
&:=_{\textit{derivations}} c_{28} \\
c_{28} &= c_{\textit{to sleep}}\ (c_{\textit{wants}}\ I_{\mathsf{S}}\ I_{\mathsf{VP}}\ c_{\textit{John}}) \\
&:=_{\textit{derived trees}} \mathsf{S}_2\ (\mathsf{NP}_1\ \textit{John})\ (\mathsf{VP}_2\ (\mathsf{V}_1\ \textit{wants}) \\
&\qquad\qquad (\mathsf{S}_2\ (\mathsf{NP}_1\ (\mathsf{PRO}_1\ \epsilon))\ (\mathsf{VP}_2\ (\mathsf{V}_1\ \textit{to})\ (\mathsf{VP}_1\ \textit{sleep})))) \\
&:=_{\textit{sem.}} \textbf{want john (sleep john)}
\end{aligned}
\tag{31}
$$

7.3 *Feature sharing and semantic computation*

As the previous section shows, features in TAG are taken into account in the ACG encoding using the typing discipline on the one hand, and using the (semantic) interpretation on the other hand, in particular when some value has to be shared in order to express the modifications performed by adjunction operations.

Unification based approaches to semantic construction in TAG typically rely on feature sharing (Gardent and Kallmeyer 2003; Gardent and Parmentier 2005; Kallmeyer and Romero 2008) in order to compositionally build the semantic representation of a sentence. In our approach, the semantic representation results from the interpretation of the derivation tree.

However, Vijay-Shanker and Joshi (1988, p. 718) already noticed that "This treatment [of variable instantiation on adjunction] can be obtained if we think of the auxiliary tree as corresponding to functions over feature structures (by λ-abstracting the variable corresponding to the feature structure for the tree that will appear below the foot node). Adjunction corresponds to applying this function to the feature structure corresponding to the subtree below the node where [it] takes place." This functional view on auxiliary trees is indeed at stake in our approach when representing auxiliary trees as functions. While the typing exerts control over the admissible derivation structures, the associated computations are managed using interpretations, to compute the derived trees as well as the logical formulas.

8 DERIVATION TREES AND SEMANTIC INTERPRETATIONS

Looking at Figure 15 (p. 32), we can consider each of the sets of λ-terms as independent combinatorial systems of the grammar architecture that Jackendoff (2002) describes: "Language comprises a number of independent combinatorial systems which are aligned with each other by means of a collection of interface systems. Syntax is among the combinatorial systems, but far from the only one."

Among those systems, $\Lambda(\Sigma_{derivations})$ and $\Lambda(\Sigma_{TAG})$ play a central role as their structures are the one that are interpreted as derived trees (and as strings, by functional composition) and as logical formulas. This is *not* the role of the syntactic trees of $\Lambda(\Sigma_{trees})$. This emphasises that the relevant syntactic algebra to provide compositional analyses for TAG, as it has been noticed very early, is not the one of derived trees, but the one of derivation trees. This section further explores the modelling power it provides.

In particular, the composition of *the inverse of a function* and a function defines the relation (the "interface") between $\Lambda(\Sigma_{trees})$ and $\Lambda(\Sigma_{logic})$ as $\mathscr{G}_{sem.} \circ \mathscr{G}^{-1}_{derived\ trees}$. In general, such a composition *is not a function*, allowing for relating a derived tree (even more a string) with several logical formulas, and vice versa. This follows the observation of Culicover and Jackendoff (2005) that "The combinatorial principles of syntax and semantics are independent; there is no 'rule-to-rule' homomorphism. (…) [T]he mapping between syntactic and semantic combinatoriality is many-to-many." However, we implement the many-to-many relation with homomorphisms and inverses of homomorphisms.

In this section, we illustrate the power of this architecture that makes derivation structures a full grammatical object with three phenomena: idioms, subordinate conjunctions with reduced clauses, and scope ambiguity. For idioms, we use the fact that derivation structures are first-class citizens of the formalism. While this could also be expressed in TAG (for instance following the interpretation of derivation trees provided by Shieber 1994), this naturally fits our architecture. For subordinate conjunctions, we rely on the fact that the typing of abstract terms does not need to stick to the tree structure, and in particular to the Gorn addresses, unlike derivation trees in TAG, extending

the modelling capabilities. Finally, for scope ambiguity, we show how our approach can take into account analyses from other formalisms, such as categorial and type-logical grammars. We do this remaining in the ACG model, contrary to the TAG extension (TAG with cosubstitution, Barker 2010) to which it corresponds. Other examples that go beyond TAG capabilities are discussed in Section 9, in particular for discourse parsing.

8.1 *Idioms*

Because TAGs provide whole fragments of phrase structures, they can encode the rigid parts of idioms as well as the ones that are subject to possible modifications. Moreover, the role of the derivation structure as a bridge to semantic interpretation nicely captures the relation between a composed syntax and an atomic meaning. With the ACG encoding of TAG, Kobele (2012) shows that we can introduce a constant term that is interpreted as the combination (by adjunction or substitution) of several elementary trees. It goes beyond the previous approaches (Abeillé and Schabes 1989; Shieber and Schabes 1990; Abeillé 1995) in that the derived tree does not need to be an elementary tree of the grammar, but is instead the result of a partial derivation.

We illustrate this with (32) (p. 56). Figure 29 presents the initial trees that allow us to analyze (32) as the literal expression, with a compositional meaning built out of the composition of the initial trees α_{John}, α_{kicked}, α_{the}, and α_{bucket}. We actually syntactically analyze the idiomatic expression the same way, except that the combination of α_{kicked}, α_{the}, and α_{bucket} is also considered as the interpretation of the constant $c_{kicked\ the\ bucket}$. We use $c_{kicked\ the\ bucket}$ in the idiomatic derivation in Figure 30(c) to stress that there is no corresponding elementary tree. The ACG abstract term we get really corresponds to this derivation, as the term $C_{30(c)}$ of Equation (33) on page 57 shows. In both cases, the derived tree is the same (Figure 30(a)). However, the derivation trees differ, as Figure 30(b) and Figure 30(c) show.

(32) John kicked the bucket

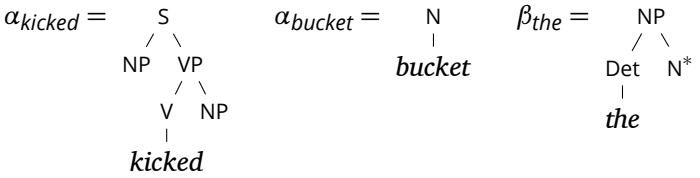
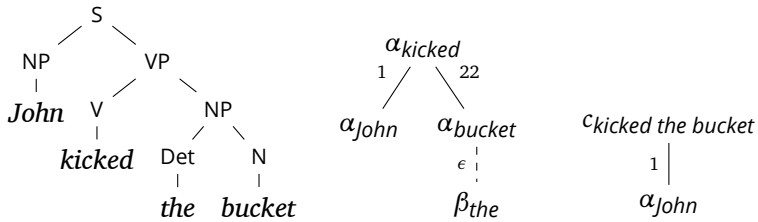


Figure 29: Elementary trees to analyze "kick the bucket"



(a) Derived tree (b) Literal derivation (c) Idiomatic derivation

Figure 30: Derived tree and derivation trees for *John kicked the bucket*

Table 18 (p. 58) shows how the interpretation of the constant abstract term for the idiom is interpreted, syntactically (by $\mathscr{G}_{derived\ trees}$) but not semantically (by $\mathscr{G}_{sem.}$), as the interpretation of the partial derivation $\lambda^o s\ a\ subj.c_{kicked}\ s\ a\ subj\ (c_{bucket}\ c_{the})$. Then, according to the lexicons of Tables 17, 18, and 19, (33), (34), (35), and (36) hold (next page). They show that the two terms $C_{30(b)}$ and $C_{30(c)}$ have the same interpretations as derived tree by $\mathscr{G}_{derived\ trees}$. But they have two different interpretations as logical formulas by the ACG $\mathscr{G}_{sem.}$.

$$\begin{aligned} C_{30(b)} &= C_{kicked}\ I_{\mathsf{S}}\ I_{\mathsf{VP}}\ C_{John}\ (C_{bucket}\ C_{the}) \\ C_{30(c)} &= C_{kicked\ the\ bucket}\ I_{\mathsf{S}}\ I_{\mathsf{VP}}\ C_{John} \end{aligned} \tag{33}$$

$$\begin{aligned} \mathscr{G}_{TAG}(C_{30(b)}) &= c_{kicked}\ I_{\mathsf{S}}\ I_{\mathsf{VP}}\ c_{John}\ (c_{bucket}\ c_{the}) \\ \mathscr{G}_{TAG}(C_{30(c)}) &= c_{kicked\ the\ bucket}\ I_{\mathsf{S}}\ I_{\mathsf{VP}}\ c_{John} \end{aligned} \tag{34}$$

$$\begin{aligned} \mathscr{G}_{derived\ trees} \circ \mathscr{G}_{TAG}(C_{30(b)}) &= \mathsf{S}_2\ (\mathsf{NP}_1\ \textit{John})\ (\mathsf{VP}_2\ (\mathsf{V}_1\ \textit{kicked}) \\ &\qquad (\mathsf{NP}_2\ (\mathsf{Det}_1\ \textit{the})\ (\mathsf{N}_1\ \textit{bucket}))) \end{aligned} \tag{35}$$

$$= \mathscr{G}_{derived\ trees} \circ \mathscr{G}_{TAG}(C_{30(c)})$$

$$\begin{aligned} \mathscr{G}_{sem.}(C_{30(b)}) &= \exists!\, x.(\mathbf{bucket}\ x) \wedge (\mathbf{kick\ john}\ x) \\ \mathscr{G}_{sem.}(C_{30(c)}) &= \mathbf{die\ john} \end{aligned} \tag{36}$$

Table 17: TAG elementary tree encoding as $\Lambda(\Sigma_{trees})$ terms

	Terms of $\Lambda(\Sigma_{trees})$	Corresponding TAG elementary tree
γ_{kicked}	$\triangleq \lambda^{o} S\ a\ s\ o.S\ (\mathsf{S}_2\ s\ (a\ (\mathsf{VP}_2\ (\mathsf{V}_1\ \textit{kicked})\ o)))$ $: (T \multimap T) \multimap (T \multimap T) \multimap T \multimap T \multimap T$	α_{kicked}
γ_{bucket}	$\triangleq \lambda^{o} d.d\ (\mathsf{N}_1\ \textit{bucket})$ $: (T \multimap T) \multimap T$	α_{bucket}
γ_{the}	$\triangleq \lambda^{o} n.\mathsf{NP}_2\ (\mathsf{Det}_1\ \textit{the})\ n$ $: T \multimap T$	β_{the}
$\gamma_{kicked\ the\ bucket}$	$\triangleq \lambda^{o} S\ a\ s.S\ (\mathsf{S}_2\ s\ (a\ (\mathsf{VP}_2\ (\mathsf{V}_1\ \textit{kicked})$ $(\mathsf{NP}_2\ (\mathsf{Det}_1\ \textit{the})(\mathsf{N}_1\ \textit{bucket})))))$ $: (T \multimap T) \multimap (T \multimap T) \multimap T \multimap T$	None: composed from $\alpha_{kicked}, \alpha_{the}$, and α_{bucket}

Table 18: Constants of $\Sigma_{derivations}$ and their interpretation by $\mathcal{G}_{derived\ trees}$ and $\mathcal{G}_{sem.}$

c_{kicked}	$: (\mathsf{S} \multimap \mathsf{S}) \multimap (\mathsf{VP} \multimap \mathsf{VP}) \multimap \mathsf{NP} \multimap \mathsf{NP} \multimap \mathsf{S}$ $:=_{derived\ trees} \gamma_{kicked}$ $:=_{sem.} \lambda^{o} s\ a\ \textit{subj}\ \textit{obj}.s\ (\textit{subj}\ (a\ (\lambda x.\textit{obj}\ (\lambda y.\mathbf{kick}\ x\ y))))$
c_{the}	$: \mathsf{N} \multimap \mathsf{NP}$ $:=_{derived\ trees} \gamma_{the}$ $:=_{sem.} \lambda^{o} P\ Q.\exists!\, x.(P\ x) \wedge (Q\ x)$
c_{bucket}	$: (\mathsf{N} \multimap \mathsf{NP}) \multimap \mathsf{NP}$ $:=_{derived\ trees} \gamma_{bucket}$ $:=_{sem.} \lambda^{o} Q.Q\ \mathbf{bucket}$
$c_{kicked\ the\ bucket}$	$: (\mathsf{S} \multimap \mathsf{S}) \multimap (\mathsf{VP} \multimap \mathsf{VP}) \multimap \mathsf{NP} \multimap \mathsf{S}$ $:=_{derived\ trees} \lambda^{o} s\ a\ \textit{subj}.(\mathcal{G}_{derived\ trees}(c_{kicked}))\ s\ a\ \textit{subj}$ $((\mathcal{G}_{derived\ trees}(c_{bucket}))\ (\mathcal{G}_{derived\ trees}(c_{the})))$ $:=_{derived\ trees} \lambda^{o} s\ a\ \textit{subj}.\gamma_{kicked}\ s\ a\ \textit{subj}\ (\gamma_{bucket}\ \gamma_{the})$ $:=_{sem.} \lambda^{o} s\ a\ \textit{subj}.s\ (\textit{subj}\ (a\ (\lambda x.\mathbf{die}\ x)))$

8.2 *Subordinate conjunctions*

We saw in Section 7.2 that infinitive clauses behave like clauses missing a subject. In this case, the matrix clause (control verb) adjoins on the infinitive clause. As the latter is argument of the modifier, we could use an extra S'_A type that was interpreted as $(\mathsf{NP} \multimap \mathsf{S}) \multimap \mathsf{S}$ and make the modifier fill the semantic subject by its own subject.

In the case of subordinate conjunctions as in (37), it is the subordinate clause that adjoins on the matrix clause and uses it as argument, as Figure 31 shows: the substitution node S is meant for the reduced infinitive clause, and the foot node for adjoining into the matrix clause. But if the latter is interpreted as a full proposition, there is no way to *decompose* it so that its subject also fills the semantic subject position

C_{kicked}	$: \mathsf{S}_A \multimap \mathsf{VP}_A \multimap \mathsf{NP} \multimap \mathsf{NP} \multimap \mathsf{S}$
	$:=_{TAG}\ c_{kicked}$
C_{the}	$: \mathsf{N}_A$
	$:=_{TAG}\ c_{the}$
C_{bucket}	$\mathsf{N}_A \multimap \mathsf{NP}$
	$:=_{TAG}\ c_{bucket}$
$C_{kicked\ the\ bucket}$	$\mathsf{S}_A \multimap \mathsf{VP}_A \multimap \mathsf{NP} \multimap \mathsf{S}$
	$:=_{TAG}\ c_{kicked\ the\ bucket}$

Table 19: Constants of Σ_{TAG} and their interpretation by $\mathscr{G}_{TAG}$

of the subordinate clause.

(37) In order to arrive on time, John left early

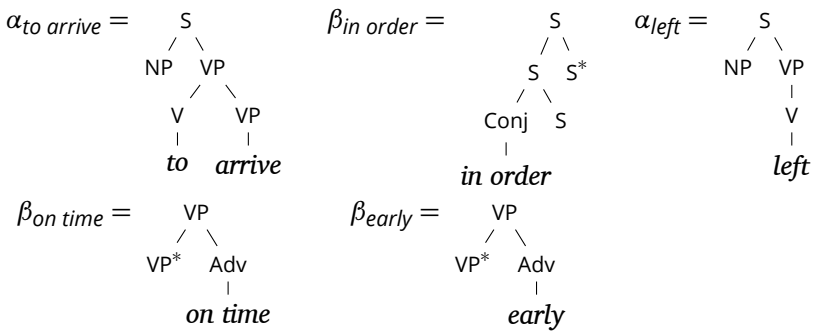


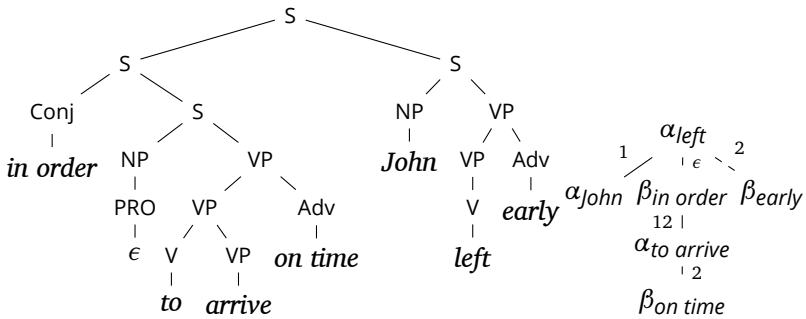
Figure 31: Auxiliary trees for subordinate conjuncts

The solution we propose uses the flexible link between the derivation and the derived trees. The constraints ACGs set on this link have to do with the type, not with the term (provided the typing is preserved). In particular:

- there is no need for an adjunction on a S_n node of a term in $\Lambda(\Sigma_{trees})$ to be the image of a term (in $\Lambda(\Sigma_{derivations})$) of type S. We already used this feature;
- there is no need for an *actual node* in the derived tree to allow for an adjunction.

In order to implement the solution, terms for verbs such as c_{left} in Table 20 have an additional argument of type $((\mathsf{NP} \multimap \mathsf{S}) \multimap (\mathsf{NP} \multimap \mathsf{S}))$ corresponding to the type of the auxiliary trees of subordinate clauses. The latter results for instance from the substitution of an infinitive clause of type $(\mathsf{NP} \multimap \mathsf{S})$ into the term standing for the initial tree of a

Figure 32: Derived tree and derivation trees for *In order to arrive on time, John left early*



(a) Derived tree (b) Derivation tree

Table 20: Constants of $\Sigma_{derivations}$ and their interpretation by $\mathcal{G}_{derived\ trees}$ and $\mathcal{G}_{sem.}$

c_{left}	$: (\mathsf{S} \multimap \mathsf{S}) \multimap ((\mathsf{NP} \multimap \mathsf{S}) \multimap (\mathsf{NP} \multimap \mathsf{S})) \multimap (\mathsf{VP} \multimap \mathsf{VP}) \multimap \mathsf{NP} \multimap \mathsf{S}$
	$:=_{derived\ trees} \gamma_{left}$
	$:=_{sem.} \lambda^{o} s\ sub\ a\ subj.sub\ (\lambda^{o} subj'.s\ (subj'\ (a\ (\lambda x.\mathbf{leave}\ x))))\ subj$
$c_{to\ arrive}$	$: (\mathsf{VP} \multimap \mathsf{VP}) \multimap \mathsf{NP} \multimap \mathsf{S}$
	$:=_{derived\ trees} \gamma_{to\ arrive}$
	$:=_{sem.} \lambda^{o} adv_{\mathsf{VP}}\ subj.subj\ (adv_{\mathsf{VP}}\ (\lambda x.\mathbf{arrive}\ x))$
$c_{in\ order}$	$: (\mathsf{NP} \multimap \mathsf{S}) \multimap ((\mathsf{NP} \multimap \mathsf{S}) \multimap (\mathsf{NP} \multimap \mathsf{S}))$
	$:=_{derived\ trees} \gamma_{in\ order}$
	$:=_{sem.} \lambda^{o} P\ Q\ subj.subj(\lambda x.\mathbf{goal}\ (P\ (\lambda^{o} p.p\ x))\ (Q\ (\lambda^{o} p.p\ x)))$
$I_{\mathsf{NP} \multimap \mathsf{S}}$	$: (\mathsf{NP} \multimap \mathsf{S}) \multimap (\mathsf{NP} \multimap \mathsf{S})$
	$:=_{derived\ trees} \lambda^{o} x.x$
	$:=_{sem.} \lambda^{o} x.x$
c_{early}	$: \mathsf{VP} \multimap \mathsf{VP}$
	$:=_{derived\ trees} \lambda^{o} x.\mathsf{VP}_2\ x\ (\mathsf{Adv}_1\ early)$
	$:=_{sem.} \lambda^{o} p.\lambda x.\mathbf{early}(p\ x)$
$c_{on\ time}$	$: \mathsf{VP} \multimap \mathsf{VP}$
	$:=_{derived\ trees} \lambda^{o} x.\mathsf{VP}_2\ x\ (\mathsf{Adv}_1\ on\ time)$
	$:=_{sem.} \lambda^{o} p.\lambda x.\mathbf{on\_time}(p\ x)$

subordinate conjunction such as $c_{in\ order}$. We can consider this additional argument as an additional possibility to get an adjunction on the S root node (the same node where a $\mathsf{S} \multimap \mathsf{S}$ adjunction is possible). As usual, in case no actual adjunction of a subordinate clause occurs, we use the $I_{\mathsf{NP} \multimap \mathsf{S}}$ constant which is interpreted (syntactically and semantically) as the identity.

We can observe in $\mathcal{G}_{derived\ trees}(c_{left}) = \gamma_{left}$ how the subordinate clause is inserted. The latter correspond to the *sub* argument in γ_{left} in Table 22 on the next page. It takes as argument the whole S rooted

Table 21: Constants of Σ_{TAG}) and their interpretation by $\mathscr{G}_{TAG}$

S''_A	$:=_{TAG} (\mathsf{NP} \multimap \mathsf{S}) \multimap (\mathsf{NP} \multimap \mathsf{S})$
S_{WS}	$:=_{TAG} \mathsf{NP} \multimap \mathsf{S}$
C_{left}	$: \mathsf{S}_A \multimap \mathsf{S}''_A \multimap \mathsf{VP}_A \multimap \mathsf{NP} \multimap \mathsf{S}$
	$:=_{TAG} c_{left}$
$C_{to\ arrive}$	$: \mathsf{VP}_A \multimap \mathsf{S}_{\mathsf{WS}}$
	$:=_{TAG} c_{to\ arrive}$
$C_{in\ order}$	$: \mathsf{S}_{\mathsf{WS}} \multimap \mathsf{S}''_A$
	$:=_{TAG} c_{in\ order}$
$I_{\mathsf{S}''}$	$: \mathsf{S}''_A$
	$:=_{TAG} I_{\mathsf{NP} \multimap \mathsf{S}}$
C_{early}	$: \mathsf{VP}_A$
	$:=_{TAG} c_{early}$
$C_{on\ time}$	$: \mathsf{VP}_A$
	$:=_{TAG} c_{on\ time}$

Table 22: TAG elementary tree encoding as $\Lambda(\Sigma_{trees})$ terms

	Terms of $\Lambda(\Sigma_{trees})$	Corresponding TAG elementary tree
γ_{left}	$= \lambda^o S\ sub\ a\ subj\ obj.$	α_{left}
	$S\ (sub\ (\lambda^o s'.\mathsf{S}_2\ s'\ (a\ (\mathsf{VP}_1\ (\mathsf{V}_1\ left))))\ subj)$	
	$: (T \multimap T) \multimap ((T \multimap T) \multimap (T \multimap T)) \multimap (T \multimap T) \multimap T \multimap T \multimap T$	
$\gamma_{to\ arrive}$	$= \lambda^o a\ s.\mathsf{S}_2\ (\mathsf{NP}_1\ s)\ (a\ (\mathsf{VP}_2\ (\mathsf{V}_1\ to)\ (\mathsf{VP}_1\ arrive)))$	$\alpha_{to\ arrive}$
	$: (T \multimap T) \multimap T \multimap T$	
$\gamma_{in\ order}$	$= \lambda^o sub\ matrix\ subj.$	$\beta_{in\ order}$
	$\mathsf{S}_2\ (\mathsf{S}_2\ (\mathsf{Conj}_1\ in\ order)\ (sub\ (\mathsf{PRO}_1\ \epsilon)))\ (matrix\ subj)$	
	$: (T \multimap T) \multimap (T \multimap T) \multimap T \multimap T$	

subtree over which the NP subject is abstracted (with $\lambda^o s'$) and the actual subject *subj* of the matrix clause. So it is the subordinate clause that is responsible for first applying the matrix clause to its subject before plugging in the resulting tree at the foot node. We can observe this behavior in $\gamma_{in\ order}$: the *sub* argument corresponds to the infinitive subordinate clause to be substituted in $\beta_{in\ order}$, while the *matrix* argument corresponds to the matrix clause into which it adjoins and to which the *subj* argument is given as the subterm (*matrix subj*) shows.

As before, the higher-order types at the level of $\Sigma_{derivations}$ are interpretations of atomic types of Σ_{TAG}. In particular, we introduce the atomic type $\mathsf{S}''_A :=_{TAG} (\mathsf{NP} \multimap \mathsf{S}) \multimap (\mathsf{NP} \multimap \mathsf{S})$ (resp. $\mathsf{S}_{\mathsf{WS}} :=_{TAG} \mathsf{NP} \multimap \mathsf{S}$) for the reduced subordinate clauses (resp. for the infinitive clause that takes place in subordinate clauses) as Table 21 shows.

With the lexicon of Tables 20, 21, and 22, we can build terms that

correspond to the derivation and derived trees of Figure 32 as (38), (39), (40) show.[26] We compute the semantic interpretation as in (41).

$$C_{32(b)} = C_{left}\ I_{\mathsf{S}}\ (C_{in\ order}\ (C_{to\ arrive}\ C_{on\ time}))\ C_{early}\ C_{John} \tag{38}$$

$$\mathscr{G}_{TAG}(C_{32(b)}) = c_{left}\ I_{\mathsf{S}}\ (c_{in\ order}\ (c_{to\ arrive}\ c_{on\ time}))\ c_{early}\ c_{John} \tag{39}$$

$$\begin{aligned}\mathscr{G}_{derived\ trees} \circ \mathscr{G}_{TAG}(C_{32(b)}) = \\ &\mathsf{S}_2\ (\mathsf{S}_2\ (\mathsf{Conj}_1\ \textit{in order})\ (\mathsf{S}_2\ (\mathsf{NP}_1\ (\mathsf{PRO}_1\ \epsilon))(\mathsf{VP}_2 \\ &(\mathsf{VP}_2\ (\mathsf{V}_1 \textit{to})\ (\mathsf{VP}_1\ \textit{arrive}))(\mathsf{Adv}_1\ \textit{on time})))) \\ &(\mathsf{S}_2\ (\mathsf{NP}_1\ \textit{John})\ (\mathsf{VP}_2\ (\mathsf{VP}_1\ (\mathsf{V}_1\ \textit{left}))\ (\mathsf{Adv}_1\ \textit{early})))\end{aligned} \tag{40}$$

$$\mathscr{G}_{sem.} \circ \mathscr{G}_{TAG}(C_{32(b)}) = \mathbf{goal}\ (\mathbf{on\_time}\ (\mathbf{arrive\ john}))\ (\mathbf{early}\ (\mathbf{leave\ john})) \tag{41}$$

Because $\mathscr{G}_{TAG}$ still is second-order, parsing is available. Parsing the logical term t_{44}^{logic} (see (42)) results in the term $t_{44} : \mathsf{S}$ of $\mathscr{A}(\mathscr{G}_{sem.} \circ \mathscr{G}_{TAG})$. This is the same term of $\mathscr{A}(\mathscr{G}_{yield} \circ \mathscr{G}_{derived\ trees} \circ \mathscr{G}_{TAG})$ that we get when parsing t_{44}^{string}.

$$t_{44}^{\text{logic}} = \exists x.(\mathbf{man}\ x) \wedge (\mathbf{goal}(\mathbf{on\_time}(\mathbf{arrive}\ x))(\mathbf{early}(\mathbf{leave}\ x))) \tag{42}$$

$$t_{44}^{\text{string}} = in + order + to + arrive + on + time + a + man + left + early \tag{43}$$

$$t_{44} = C_{left}\ I_{\mathsf{S}}(C_{in\ order}\ (C_{to\ arrive}\ C_{on\ time))}\ C_{early}\ (C_{man}\ C_{a)} \tag{44}$$

8.3 *Scope ambiguity and non-functional form-meaning relation*

The phenomena we have modeled so far make use of derivation structures (either in $\Lambda(\Sigma_{derivations})$ or in $\Lambda(\Sigma_{TAG})$) that are very close (homomorphic) to TAG derivation trees. As we can see on Figure 15 (p. 32), the relation between TAG derivations as terms of $\Lambda(\Sigma_{TAG})$ and terms of $\Lambda(\Sigma_{logic})$ is functional (encoded by the composition $\mathscr{G}_{sem.} \circ \mathscr{G}_{TAG}$). The non-functional relation is between terms of $\Lambda(\Sigma_{trees})$ and terms of $\Lambda(\Sigma_{logic})$ (encoded by the function $\mathscr{G}_{sem.} \circ \mathscr{G}^{-1}_{derived\ trees}$). So because there are two derivation trees for *John kicked the bucket*, there

[26] Note that all terms corresponding to initial trees where the adjunction of a subordinate clause can occur should have the extra argument added. For sake of simplicity, only C_{left} and c_{left} are modified here.

are two possible semantic interpretations. But with only one derivation tree for *every man loves some woman*, there is only one possible semantic interpretation. A possible solution to this problem is to use an underspecified representation formalism instead of higher-order logic to represent the semantics, as in (Pogodalla 2004a).

We present here another solution. It uses the power of higher-order typing of the abstract terms in order to provide TAGs with a relation between TAG derivation trees and meaning that *is not functional*. Nevertheless, our grammatical architecture only appeal to homomorphisms. We introduce an abstract vocabulary Σ_{CoTAG} and two ACGs. The first one, $\mathcal{G}_{CoTAG}$, maps terms of $\Lambda(\Sigma_{CoTAG})$ to terms of $\Lambda(\Sigma_{TAG})$, i.e., TAG derivation trees. The second one, $\mathcal{G}_{co\text{-}sem.}$, maps terms of $\Lambda(\Sigma_{CoTAG})$ to terms of $\Lambda(\Sigma_{logic})$. It then provides a relation between $\Lambda(\Sigma_{TAG})$ and $\Lambda(\Sigma_{logic})$ as $\mathcal{G}_{co\text{-}sem.} \circ \mathcal{G}^{-1}_{CoTAG}$ as Figure 33 (p. 63) shows. The derivation tree (in $\Lambda(\Sigma_{TAG})$) of *every man loves some woman*, for instance, will have *two* antecedents in $\Lambda(\Sigma_{CoTAG})$, hence two semantic interpretations.

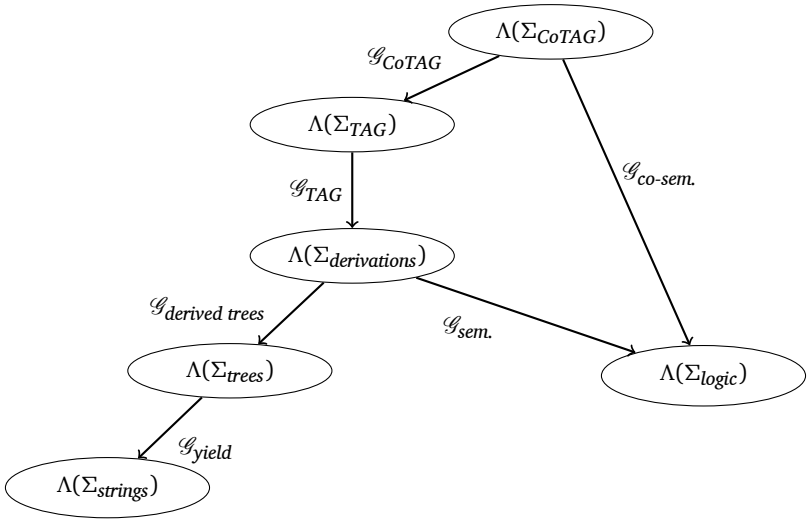


Figure 33: ACG composition for TAG and CoTAG

The idea is to use the type-raising methods of categorial and type-logical grammars. So, corresponding to a term $C_{everyone} : \mathsf{NP}$ in Σ_{TAG}, we have a term $L_{everyone} : (\mathsf{NP} \multimap \mathsf{S}) \multimap \mathsf{S}$ in Σ_{CoTAG} such that $\mathcal{G}_{CoTAG}(L_{everyone}) = \lambda^{o} P.P\ C_{everyone}$. More generally, whenever a term of type A occurring within a constituent of type B can take scope

Table 23: Constants of Σ_{CoTAG} and their interpretation by $\mathscr{G}_{CoTAG}$

L_{John}	$: \mathsf{NP}$	$:=_{CoTAG} C_{John}$
L_{loves}	$: \mathsf{S}_A \multimap \mathsf{VP}_A \multimap \mathsf{NP} \multimap \mathsf{NPS}$	$:=_{CoTAG} C_{loves}$
$L_{everyone}$	$: (\mathsf{NP} \multimap \mathsf{S}) \multimap \mathsf{S}$	$:=_{CoTAG} \lambda^{o} P.P\ C_{everyone}$
$L_{someone}$	$: (\mathsf{NP} \multimap \mathsf{S}) \multimap \mathsf{S}$	$:=_{CoTAG} \lambda^{o} P.P\ C_{someone}$
L_{every}	$: \mathsf{N}_A$	$:=_{CoTAG} C_{every}$
L_{some}	$: \mathsf{N}_A$	$:=_{CoTAG} C_{some}$
L_{man}	$: \mathsf{N}_A \multimap (\mathsf{NP} \multimap \mathsf{S}) \multimap \mathsf{S}$	$:=_{CoTAG} \lambda^{o} det\ P.P\ (C_{man}\ det)$
L_{woman}	$: \mathsf{N}_A \multimap (\mathsf{NP} \multimap \mathsf{S}) \multimap \mathsf{S}$	$:=_{CoTAG} \lambda^{o} det\ P.P\ (C_{woman}\ det)$

over this term, we associate to $C_{scoping} : A_1 \multimap \ldots \multimap A_n \multimap A$ in Σ_{TAG} a term $L_{scoping} : A_1 \multimap \ldots \multimap A_n \multimap (A \multimap B) \multimap B$ in Σ_{CoTAG} such that $\mathscr{G}_{CoTAG}(L_{scoping}) = \lambda^{o} x_1 \ldots x_n . \lambda^{o} P.P\ (C_{scoping}\ x_1 \cdots x_n)$. For other lexical items $C_{lex.\ item} : \alpha$, we have $L_{lex.item} : \alpha$ such that $\mathscr{G}_{CoTAG}(L_{lex.item}) = C_{lex.\ item}$. And for any atomic type A, $\mathscr{G}_{CoTAG}(A) = A$.

Table 23 exemplifies the approach for quantified noun phrases (note that proper nouns, for instance, are not type-raised). Atomic types in Σ_{CoTAG} are the same as in Σ_{TAG}. With L_{21}^{sws} and L_{21}^{ows} as defined in (45) and (46) respectively, we indeed have $\mathscr{G}_{CoTAG}(L_{21}^{\mathrm{sws}}) = \mathscr{G}_{CoTAG}(L_{21}^{\mathrm{ows}})$, i.e., two different abstract terms of $\Lambda(\Sigma_{CoTAG})$ that are mapped to the same term of $\Lambda(\Sigma_{TAG})$ (TAG derivation tree).

$$\begin{aligned} L_{21}^{\mathrm{sws}} &= (L_{man}\ L_{every}) \\ &\qquad (\lambda^{o} x.(L_{woman}\ L_{some})\ (\lambda^{o} y.L_{loves}\ I_{\mathsf{S}}\ I_{\mathsf{VP}}\ x\ y)) \\ \mathscr{G}_{CoTAG}(L_{21}^{\mathrm{sws}}) &= (\lambda^{o} P.P\ (C_{man}\ C_{every}))\ (\lambda^{o} x.(\lambda^{o} P.P\ (C_{woman}\ C_{some})) \\ &\qquad (\lambda^{o} y.C_{loves}\ I_{\mathsf{S}}\ I_{\mathsf{VP}}\ x\ y)) \\ &= C_{loves}\ I_{\mathsf{S}}\ I_{\mathsf{VP}}\ (C_{man}\ C_{every})\ (C_{woman}\ C_{some}) \end{aligned} \tag{45}$$

$$\begin{aligned} L_{21}^{\mathrm{ows}} &= (L_{woman}\ L_{some}) \\ &\qquad (\lambda^{o} y.(L_{man}\ L_{every})\ (\lambda^{o} x.L_{loves}\ I_{\mathsf{S}}\ I_{\mathsf{VP}}\ x\ y)) \\ \mathscr{G}_{CoTAG}(L_{21}^{\mathrm{ows}}) &= (\lambda^{o} P.P\ (C_{woman}\ C_{some}))\ (\lambda^{o} y.(\lambda^{o} P.P\ (C_{man}\ C_{every})) \\ &\qquad (\lambda^{o} x.C_{loves}\ I_{\mathsf{S}}\ I_{\mathsf{VP}}\ x\ y)) \\ &= C_{loves}\ I_{\mathsf{S}}\ I_{\mathsf{VP}}\ (C_{man}\ C_{every})\ (C_{woman}\ C_{some}) \end{aligned} \tag{46}$$

In order to get two semantic interpretations from the two abstract terms of $\Lambda(\Sigma_{CoTAG})$, we need to directly provide them with a semantic lexicon. For if we keep on interpreting them through Σ_{TAG} and

Table 24: Constants of Σ_{CoTAG} and their interpretation by $\mathscr{G}_{co\text{-}sem.}$

NP	$:=_{co\text{-}sem.} e$
VP	$:=_{co\text{-}sem.} e \to t$
L_{John}	$:=_{co\text{-}sem.} \mathbf{john}$
L_{loves}	$:=_{co\text{-}sem.} \lambda adv_{\mathsf{S}} adv_{\mathsf{VP}} subj obj.adv_{\mathsf{S}}\ (adv_{\mathsf{VP}}\ (\lambda x.\mathbf{love}\ x\ obj)\ subj)$
$L_{everyone}$	$:=_{co\text{-}sem.} \lambda Q.\forall x.(\mathbf{human}\ x) \Rightarrow (Q\ x)$
$L_{someone}$	$:=_{co\text{-}sem.} \lambda Q.\exists x.(\mathbf{human}\ x) \wedge (Q\ x)$
L_{every}	$:=_{co\text{-}sem.} \lambda P\ Q.\forall x.(P\ x) \Rightarrow (Q\ x)$
L_{some}	$:=_{co\text{-}sem.} \lambda P\ Q.\exists x.(P\ x) \wedge (Q\ x)$
L_{man}	$:=_{co\text{-}sem.} \lambda det.det\ \mathbf{man}$
L_{woman}	$:=_{co\text{-}sem.} \lambda det.det\ \mathbf{woman}$

$\Sigma_{derivations}$, because the two terms L_{21}^{sws} and L_{21}^{ows} are interpreted as a single term in $\Lambda(\Sigma_{TAG})$, we still would get a single interpretation. In other words, we do not want the diagram of Figure 33 to commute.

Table 24 defines the $\mathscr{G}_{co\text{-}sem.}$ interpretation into terms of $\Lambda(\Sigma_{logic})$. Contrary to $\mathscr{G}_{sem.}$ where NP's are interpreted with the higher-order type $(e \to t) \multimap t$, because quantified noun phrases are given the type $(\mathsf{NP} \multimap \mathsf{S}) \multimap \mathsf{S}$ in Σ_{CoTAG}, we now interpret NP as e. All the other interpretations, in particular for verbs, are defined accordingly.[27]

We can now compute the semantic interpretation of L_{21}^{sws} and L_{21}^{ows} by $\mathscr{G}_{co\text{-}sem.}$. (47) shows that these two terms are mapped to two logical formulas corresponding to the subject wide scope reading on the one hand, and to the object wide scope reading on the other hand.

$$\mathscr{G}_{co\text{-}sem.}(L_{21}^{\text{sws}}) = \forall x.(\mathbf{man}\ x) \Rightarrow (\exists x'.(\mathbf{woman}\ x') \wedge (\mathbf{love}\ x\ x')) \quad (47)$$

$$\mathscr{G}_{co\text{-}sem.}(L_{21}^{\text{ows}}) = \exists x.(\mathbf{woman}\ x) \wedge (\forall x'.(\mathbf{man}\ x') \Rightarrow (\mathbf{love}\ x'\ x)) \quad (48)$$

This approach to scope ambiguity, first proposed in (Pogodalla 2007b,a) is used in (Kobele and Michaelis 2012) to provide an ACG formalization of the cosubstitution operation for TAG (Barker 2010). This also makes explicit Barker's (2010) claim that "cosubstitution is a version of the continuation-based approaches to scope-taking (…)" And indeed, the type $(\mathsf{NP} \multimap \mathsf{S}) \multimap \mathsf{S}$ corresponds to making the continuation of a noun phrase (i.e., its scope) part of its interpretation.

The $\mathscr{G}_{CoTAG}$ is not a second-order ACG. In this very case, because of lexicalization, we know that parsing is decidable, but it can be

[27] Note, however, that, because at the abstract level we only have linear types, in order to allow for non linearity at the object level, we have to uniformly interpret $\multimap$ as $\to$, so that the image of $(\mathsf{NP} \multimap \mathsf{S}) \multimap \mathsf{S}$ is $(e \to t) \multimap t$. Another possibility would be to use the exponential connectives of linear logic.

complex. Salvati (2007) presents a lexicalized third-order ACG whose membership problem reduces to an NP-complete problem. There is currently no implementation of parsing for such grammars in the ACG toolkit. The identification of fragments that are both linguistically relevant and computationally tractable is ongoing work.

This extension with more abstract levels can also be used to model (non-local) MCTAG. And one level more can set control on MCTAG (similar to the control that $\mathscr{G}_{TAG}$ adds on $\mathscr{A}(\mathscr{G}_{derived\ trees})$) to stay within the polynomial parsable languages of set-local MCTAG (Pogodalla 2009).

9 RELATED APPROACHES

Moving to ACGs to encode TAGs and to build TAG semantic representations offers several advantages. First, we saw in Section 3.2 that we can benefit from parsing algorithms and optimization techniques grounded on well established fields such as type-theory and Datalog.

A second advantage, also concerning the parsing algorithms, is to offer a *inherently reversible* framework (Dymetman 1994). Kanazawa's Datalog reduction (2007; 2017) indeed makes no hypothesis on the object language: it can be a language of strings, of trees, or of any kind of (almost linear) λ-terms. In the latter case, it can represent usual logical semantic formulas. While in NLP *parsing* usually refers to building a parse structure (or a semantic term) from a string representation and *generation* (or *surface realization*) refers to building a string from a semantic representation, they both rely on ACG parsing (i.e., recovering the abstract structure from an object term), and the algorithms are the same.

This makes an important difference between the ACG approach and the synchronous approaches to semantic construction. If both are based on (or can be reformulated using) a tree transduction, the latter does not offer a built-in transformation to β-reduced terms (which may definitely not be trees but rather graphs) at the semantic level. When parsing strings, synchronous grammars (Nesson and Shieber 2006; Nesson 2009) build semantic trees that correspond to λ-terms *before* the β-reduction. Processing such trees to produce the β-reduced form is straightforward. However the inverse process, the one that is of interest for generation, is not. It actually corresponds to the mor-

phism inversion found in ACG parsing. The ACG framework tells us that this inversion is possible, and, for the second-order case, it is indeed implemented.

Koller and Kuhlmann (2011; 2012) also propose parsing by morphism inversion using interpreted regular tree grammars, and the approach completely fits the synchronous approach. But, as for synchronous TAG, this formalism is not well-suited to deal with semantics represented with logical formulas. To parse a term t indeed requires that the set of trees that are interpreted as t is regular. For instance, if the string algebra comes with the 2-ary concatenation operation, this set is the set of all the bracketings of the string to parse (Koller and Kuhlmann 2011) (the string algebra Koller and Kuhlmann (2012) propose for the TAG encoding is different, in order to keep the complexity bound for parsing low). Applying the same approach to logical representations based on λ-calculus would mean to represent all terms that are β-equivalent to the term we want to parse by a regular tree grammar. It is not clear how it can be done.

Semantic representation with λ-terms, and more generally the ACG type-theoretic settings,[28] also provides tight links with formal logical semantics. The various grammatical formalisms ACGs can encode may be linked to various semantic theories. This concerns both semantic theories, such as event semantics (Davidson 2001) (for a type theoretic account, see Blom *et al.* 2012) or dynamic semantics (Kamp and Reyle 1993; Groenendijk and Stokhof 1991) (for a type theoretic semantics account, see de Groote 2006; Martin and Pollard 2014),[29] and phenomena at the syntax-semantics interface where approaches based on underspecification (Pogodalla 2004a,b) or based on type theory and higher-order logic (Pogodalla 2007b,a; Kobele and Michaelis 2012) can be expressed.

Because they only use unification,[30] the approaches to TAG semantics by unification (Gardent and Kallmeyer 2003; Kallmeyer and

[28] Like other categorial grammars (van Benthem 1986; Carpenter 1997; Steedman 2001; Steedman and Baldridge 2011).

[29] Note however that the semantic calculi are somewhat extended with additional operators and then do not fulfill the requirements allowing for reversibility. This is a research program on its own.

[30] More specifically, they use first-order matching, as one of the feature is always a variable.

Romero 2004, 2008) do not easily extend to higher-order semantics: only conjunctions of proposition are allowed, and no application. A first consequence is that the actual representation language needs to be embedded into a reified logical language (typically a labeled underspecified representation language). For instance, the semantics of an adverb adjoining to a VP node cannot be represented as a function from $(e \to t)$ to $(e \to t)$. It is represented as a proposition expressing that some property holds of a label which gets its value by unification with the label corresponding to the semantics of the VP in the verb initial tree. When dealing with higher-order representations, as for dynamic semantics of discourse (de Groote 2006; Martin and Pollard 2014), it becomes awkward to assign values of arguments with unification and to compute the semantic representation by β-reduction.

Moreover, ACGs uniformly deal with the interpretation of derivation trees, either as strings, derived trees, or semantic representations. Consequently, the same parsing algorithms apply. This is not the case for the unification-based approaches and the reversibility of the grammars is not ensured.

Another benefit of the ACG approach to the syntax-semantics interface over the synchronous TAG or over the unification-based approach is that, by construction, it is compositional, and the homomorphism requirement between the syntactic and the semantic categories holds. For instance, in synchronous TAG, in a pair of syntactic and semantic trees, it is possible to link a node X (in the syntactic tree) with a node of type α (in the semantic tree), while having a pair of auxiliary trees whose syntactic tree has a foot and a root node labeled by X, but the nodes in the semantic trees are labeled by $\beta \neq \alpha$, yielding semantic trees that are not well-typed. A similar thing can happen in unification-based approaches if the semantic features to be unified are not the same. This is not possible in ACG and the ACG toolkit would raise a typing error, in the same way that statically typed programming languages ensure type-safeness.

Finally, the modularity of the ACG framework allows us to look at TAG as fragments of a larger class of grammars. Multi-Component TAG (MCTAG, Weir (1988)) can also been described using a similar architecture (Pogodalla 2009). It is also possible to add operations to the substitution and adjunction ones that would otherwise be difficult, if not impossible, to express as TAG (or MCTAG) operations. Such op-

erations can be used in order to link a TAG phrase grammar with a TAG discourse grammar without requiring an intermediate processing step (Danlos *et al.* 2015, 2016), contrary to D-LTAG (Webber and Joshi 1998; Forbes *et al.* 2003; Webber 2004; Forbes-Riley *et al.* 2006) or D-STAG (Danlos 2009, 2011). But, provided the encoding remains in the second-order ACG class, the grammars remain reversible and there is no need to design new parsing algorithms.

10 CONCLUSION

We have presented a model of the syntax-semantics interface for TAGs hinging upon the ACG framework. We demonstrated, with the help of classical TAG syntax-semantics examples and new modellings, that this framework offers a lot of flexibility and expressiveness. In particular, we built on the modular properties of ACG that results from the two notions of composition between grammars it provides. These composition modes have been used for the syntax-semantics interface on the one hand, and for restricting the derivations to actual TAG derivations using a second-order ACG on the other hand. This allowed us to apply the ACG parsing results and to make the grammar reversible so that both parsing and syntactic realization are available.

Moreover, we showed that new modellings can be proposed that extend the standard TAG analyses without the requirement of designing new parsing algorithms. This was illustrated with phenomena such as idioms or subordinate conjunctions. We also showed that we can bring into TAG accounts from type-logical frameworks, such as the modelling of scope ambiguities.

This shows how relevant ACGs are as models of the syntax-semantics interface in general, and for TAG in particular. Relying on the work we presented here, we can consider modelling other standard extensions of TAGs, such as MCTAG. We can also consider relating TAG to other type-theoretic approaches to formal semantics modelling, e.g., discourse, knowledge and beliefs, time, etc. Finally, we believe this can give a new perspective on ways to model challenging phenomena in TAG, such as coordination.

REFERENCES

Anne ABEILLÉ (1990), French and English determiners: Interaction of morphology, syntax and semantics in Lexicalized Tree Adjoining Grammars, in *Proceedings of the First International Workshop on Tree Adjoining Grammar and Related Frameworks (TAG+1)*, pp. 17–20, Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloß Dagstuhl, Germany, ACL anthology: W90-0215.

Anne ABEILLÉ (1993), *Les nouvelles syntaxes*, Armand Colin Éditeur, Paris.

Anne ABEILLÉ (1995), The Flexibility of French Idioms: A Representation with Lexicalised Tree Adjoining Grammar, in Martin EVERAERT, Erik-Jan VAN DER LINDEN, André SCHENK, and Rob SCHREUDER, editors, *Idioms*, chapter 1, pp. 15–42, Psychology Press, Taylor & Francis Group.

Anne ABEILLÉ (2002), *Une grammaire électronique du français*, Sciences du langage, CNRS Éditions.

Anne ABEILLÉ and Yves SCHABES (1989), Parsing Idioms in Lexicalized TAGs, in *Proceedings of the Fourth Conference of the European Chapter of the Association for Computational Linguistics (EACL 1989)*, pp. 1–9, Association for Computational Linguistics, Manchester, England, ACL anthology: E89-1001.

Hendrik Pieter BARENDREGT (1984), *The lambda calculus*, volume 103 of *Studies in Logic and the Foundations of Mathematics*, North-Holland.

Chris BARKER (2010), Cosubstitution, derivational locality, and quantifier scope, in Srinivas BANGALORE, Robert FRANK, and Maribel ROMERO, editors, *Proceedings of the Tenth International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+10)*, pp. 135–142, Linguistics Department, Yale University, New Haven, CT, US, `http://aclweb.org/anthologyW10-4417`, ACL anthology: W10-44.

Katalin BIMBÓ (2015), The decidability of the intensional fragment of classical linear logic, *Theoretical Computer Science*, 597:1–17, 10.1016/j.tcs.2015.06.019.

Chris BLOM, Philippe DE GROOTE, Yoad WINTER, and Joost ZWARTS (2012), Implicit Arguments: Event Modification or Option Type Categories?, in Maria ALONI, Vadim KIMMELMAN, Floris ROELOFSEN, Galit W. SASSOON, Katrin SCHULZ, and Matthijs WESTERA, editors, *Logic, Language and Meaning*, volume 7218 of *Lecture Notes in Computer Science*, pp. 240–250, Springer Berlin Heidelberg, ISBN 978-3-642-31481-0, 10.1007/978-3-642-31482-7_25.

Johan BOS (1995), Predicate Logic Unplugged, in *Proceedings of the Tenth Amsterdam Colloquium*, `http://www.let.rug.nl/bos/pubs/Bos1996AmCo.pdf`.

Pierre BOURREAU (2012), *Jeux de Typage et Analyse de λ-Grammaires Non-Contextuelles*, Ph.D. thesis, Université Bordeaux I, HAL open archive: tel-00733964.

Pierre BOURREAU (2013), Traitements d'ellipses : deux approches par les grammaires catégorielles abstraites, in *Actes de la 20e conférence sur le Traitement Automatique des Langues Naturelles*, pp. 215–228, Association pour le Traitement Automatique des Langues, Les Sables d'Olonne, France, `http://www.atala.org/taln_archives/TALN/TALN-2013/taln-2013-long-016`.

Pierre BOURREAU and Sylvain SALVATI (2011), A Datalog Recognizer for Almost Affine λ-CFGs, in Makoto KANAZAWA, András KORNAI, Marcus KRACHT, and Hiroyuki SEKI, editors, *The Mathematics of Language*, volume 6878 of *Lecture Notes in Computer Science*, pp. 21–38, Springer Berlin Heidelberg, ISBN 978-3-642-23210-7, 10.1007/978-3-642-23211-4_2.

Marie-Hélène CANDITO (1996), A principle-based hierarchical representation of LTAGs, in *Proceedings of the 16th International Conference on Computational Linguistics (COLING*, pp. 194–199, ACL anthology: C96-1034.

Marie-Hélène CANDITO (1999), *Représentation modulaire et paramétrable de grammaires électroniques lexicalisées : application au français et à l'italien*, Ph.D. thesis, Université Paris 7, `http://www.linguist.univ-paris-diderot.fr/~mcandito/Publications/candito-these.pdf`.

Marie-Hélène CANDITO and Sylvain KAHANE (1998), Can the TAG Derivation Tree Represent a Semantic Graph? An Answer in the Light of Meaning-Text Theory, in *Proceedings of the Fourth International Workshop on Tree Adjoining Grammars and Related Framework (TAG+4)*, volume 98-12 of *IRCS Technical Report Series*, University of Pennsylvania, ACL anthology: W98-0106.

Bob CARPENTER (1997), *Type-Logical Semantics*, The MIT Press.

John CHEN, Srinivas BANGALORE, and K. VIJAY-SHANKER (2006), Automated extraction of Tree-Adjoining Grammars from treebanks, *Natural Language Engineering*, 12(3):251–299, 10.1017/S1351324905003943.

Hubert COMON, Max DAUCHET, Rémi GILLERON, Christoph. LÖDING, Florent JACQUEMARD, Denis LUGIEZ, Sophie TISON, and Marc TOMMASI (2007), Tree Automata Techniques and Applications, `http://www.grappa.univ-lille3.fr/tata`, release October, 12th 2007.

Benoît CRABBÉ (2005), Grammatical Development with Xmg, in Philippe BLACHE, Edward STABLER, Joan BUSQUETS, and Richard MOOT, editors, *Logical Aspects of Computational Linguistics: 5th International Conference, LACL 2005, Bordeaux, France, April 28-30, 2005. Proceedings*, volume 3492 of *LNCS/LNAI*, pp. 84–100, Springer Berlin Heidelberg, 10.1007/11422532_6.

Benoît CRABBÉ, Denys DUCHIER, Claire GARDENT, Joseph Le ROUX, and Yannick PARMENTIER (2013), XMG: eXtensible MetaGrammar, *Computational Linguistics*, 39(3):591–629, ACL anthology: J13-3005.

Peter W. CULICOVER and Ray JACKENDOFF (2005), *Simpler Syntax*, Oxford University Press.

Haskell Brooks CURRY (1961), Some Logical Aspects of Grammatical Structure, in Roman JAKOBSON, editor, *Structure of Language and its Mathematical Aspects: Proceedings of the Twelfth Symposium in Applied Mathematics*, pp. 56–68, American Mathematical Society.

Laurence DANLOS (2009), D-STAG : un formalisme d'analyse automatique de discours basé sur les TAG synchrones, *T.A.L.*, 50(1):111–143, HAL open archive: inria-00524743.

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*, pp. 64–84, Springer, 10.1007/978-3-642-20169-1_5.

Laurence DANLOS, Aleksandre MASKHARASHVILI, and Sylvain POGODALLA (2015), Grammaires phrastiques et discursives fondées sur les TAG : une approche de D-STAG avec les ACG, in *TALN 2015 - 22e conférence sur le Traitement Automatique des Langues Naturelles*, Actes de TALN 2015, pp. 158–169, Association pour le Traitement Automatique des Langues, Caen, France, HAL open archive: hal-01145994.

Laurence DANLOS, Aleksandre MASKHARASHVILI, and Sylvain POGODALLA (2016), Interfacing Sentential and Discourse TAG-based Grammars, in *The 12th International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+12)*, Proceedings of the 12th International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+12), Düsseldorf, Germany, HAL open archive: hal-01328697. ACL anthology: W16-3303.

Donald DAVIDSON (2001), *Essays on Actions and Events*, volume 1 of *Philosophical Essays of Donald Davidson*, Clarendon Press. Oxford University Press.

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*, pp. 148–155, ACL anthology: P01-1033.

Philippe DE GROOTE (2002), Tree-Adjoining Grammars as Abstract Categorial Grammars, in Frank (2002), pp. 145–150, ACL anthology: W02-2220.

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*, 10.3765/salt.v16i0.2952.

Philippe DE GROOTE (2015), Abstract Categorial Parsing as Linear Logic Programming, in *Proceedings of the 14th Meeting on the Mathematics of Language (MoL 2015)*, pp. 15–25, Association for Computational Linguistics, Chicago, United States, HAL open archive: hal-01188632. ACL anthology: W15-2302.

Philippe DE GROOTE and Makoto KANAZAWA (2013), A Note on Intensionalization, *Journal of Logic, Language and Information*, 22(2):173–194, 10.1007/s10849-013-9173-9, HAL open archive: hal-00909207.

Philippe DE GROOTE and Sarah MAAREK (2007), Type-theoretic extensions of Abstract Categorial Grammars, in *New Directions in Type-Theoretic Grammars, proceedings of the workshop*, pp. 18–30, `http://let.uvt.nl/general/people/rmuskens/ndttg/ndttg2007.pdf`.

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, 10.1007/s10849-004-2114-x, HAL open archive: inria-00112956.

Philippe DE GROOTE, Ryo YOSHINAKA, and Sarah MAAREK (2007), On Two Extensions of Abstract Categorial Grammars, in Nachum DERSHOWITZ and Andrei VORONKOV, editors, *Logic for Programming, Artificial Intelligence, and Reasoning, 14th International Conference, LPAR 2007, Yerevan, Armenia, October 15-19, 2007, Proceedings*, volume 4790 of *Lecture Notes in Computer Science*, pp. 273–287, Springer, ISBN 978-3-540-75558-6, 10.1007/978-3-540-75560-9_21.

Éric Villemonte DE LA CLERGERIE (2005), From metagrammars to factorized TAG/TIG parsers, in *Proceedings of the Ninth International Workshop on Parsing Technology*, pp. 190–191, Association for Computational Linguistics, Vancouver, British Columbia, ACL anthology: W05-1522.

Marc DYMETMAN (1994), Inherently Reversible Grammars, in Tomek STRZALKOWSKI, editor, *Reversible Grammars in Natural Language Processing*, chapter 2, pp. 33–57, Kluwer Academic Publishers.

Markus EGG, Alexander KOLLER, and Joachim NIEHREN (2001), The Constraint Language for Lambda Structures, *Journal of Logic, Language, and Information*, 10(4):457–485, 10.1023/A:1017964622902.

Katherine FORBES, Eleni MILTSAKAKI, Rashmi PRASAD, Anoop SARKAR, Aravind K. JOSHI, and Bonnie Lynn WEBBER (2003), D-LTAG System: Discourse Parsing with a Lexicalized Tree-Adjoining Grammar, *Journal of Logic, Language and Information*, 12(3):261–279, ISSN 0925-8531, 10.1023/A:1024137719751, special Issue: Discourse and Information Structure.

Katherine FORBES-RILEY, Bonnie Lynn WEBBER, and Aravind K. JOSHI (2006), Computing Discourse Semantics: The Predicate-Argument Semantics of Discourse Connectives in D-LTAG, *Journal of Semantics*, 23(1):55–106, 10.1093/jos/ffh032.

Robert FRANK, editor (2002), *Proceedings of the Sixth International Workshop on Tree Adjoining Grammar and Related Frameworks (TAG+6)*, Università di Venezia, ACL anthology: W02-22.

Claire GARDENT (2008), Integrating a unification-based semantics in a large scale Lexicalised Tree Adjoininig Grammar for French, in *Proceedings of the 22nd International Conference on Computational Linguistics (Coling 2008)*, pp. 249–256, Manchester, UK, ACL anthology: C08-1032.

Claire GARDENT and Laura KALLMEYER (2003), Semantic construction in Feature-Based TAG, in *Proceedings of the 10th Meeting of the European Chapter of the Association for Computational Linguistics (EACL)*, pp. 123–130, ACL anthology: E03-1030.

Claire GARDENT and Yannick PARMENTIER (2005), Large Scale Semantic Construction for Tree Adjoining Grammars, in Philippe BLACHE, Edward STABLER, Joan BUSQUETS, and Richard MOOT, editors, *Logical Aspects of Computational Linguistics*, volume 3492 of *LNCS*, pp. 131–146, Sptringer, 10.1007/11422532_9.

Jean-Yves GIRARD (1987), Linear Logic, *Theoretical Computer Science*, 50(1):1–102, 10.1016/0304-3975(87)90045-4.

Jeroen GROENENDIJK and Martin STOKHOF (1991), Dynamic Predicate Logic, *Linguistics and Philosophy*, 14(1):39–100, 10.1007/BF00628304.

Chung-hye HAN and Giorgio SATTA, editors (2012), *Proceedings of the 11th International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+11)*, INRIA Paris Rocquencourt & Université Paris Diderot, Paris, France, ACL anthology: W12-4600.

Mathieu HUOT (2017), *Conservative extensions of Montague semantics*, Master's thesis, ENS Cachan, Université Paris-Saclay.

Ray JACKENDOFF (2002), *Foundations of Language: Brain, Meaning, Grammar, Evolution*, Oxford University Press.

Aravind K. JOSHI (1985), Tree-adjoining grammars: How much context sensitivity is required to provide reasonable structural descriptions?, in David R. DOWTY, Lauri KARTTUNEN, and Arnold M. ZWICKY, editors, *Natural Language Parsing*, pp. 206–250, Cambridge University Press.

Aravind K. JOSHI (1994), Preface, *Computational Intelligence*, 10(4):VII–XV, ISSN 1467-8640, 10.1111/j.1467-8640.1994.tb00002.x.

Aravind K. JOSHI, Laura KALLMEYER, and Maribel ROMERO (2003), Flexible Composition in LTAG: Quantifier Scope and Inverse Linking, in Harry BUNT, Ielka VAN DER SLUIS, and Roser MORANTE, editors, *Proceedings of the Fifth International Workshop on Computational Semantics IWCS-5*.

Aravind K. JOSHI, Leon S. LEVY, and Masako TAKAHASHI (1975), Tree Adjunct Grammars, *Journal of Computer and System Sciences*, 10(1):136–163, 10.1016/S0022-0000(75)80019-5.

Aravind K. JOSHI and Yves SCHABES (1997), Tree-adjoining grammars, in Grzegorz ROZENBERG and Arto K. SALOMAA, editors, *Handbook of formal languages*, volume 3, chapter 2, Springer.

Sylvain KAHANE, Marie-Hélène CANDITO, and Yannick DE KERCADIO (2000), An alternative description of extractions in TAG, in *Proceedings of the Fifth International Workshop on Tree Adjoining Grammars and Related Framework (TAG+5)*, Université Paris 7, Jussieu, Paris, France, ACL anthology: W00-2016.

Laura KALLMEYER (2002), Using an Enriched TAG Derivation Structure as Basis for Semantics, in Frank (2002), pp. 127–136, ACL anthology: W02-2218.

Laura KALLMEYER (2010), *Parsing Beyond Context-Free Grammars*, Cognitive Technologies, Springer-Verlag Berlin Heidelberg, 10.1007/978-3-642-14846-0.

Laura KALLMEYER and Aravind K. JOSHI (2003), Factoring Predicate Argument and Scope Semantics: Underspecified Semantics with LTAG, *Research on Language and Computation*, 1(1-2):3–58, ISSN 1570-7075, 10.1023/A:1024564228892.

Laura KALLMEYER and Marco KUHLMANN (2012), A Formal Model for Plausible Dependencies in Lexicalized Tree Adjoining Grammar, in Han and Satta (2012), pp. 108–116, ACL anthology: W12-4613.

Laura KALLMEYER and Maribel ROMERO (2004), LTAG Semantics with Semantic Unification, in Owen RAMBOW and Matthew STONE, editors, *Seventh International Workshop on Tree Adjoining Grammar and Related Formalisms - TAG+7*, pp. 155–162, Simon Fraser University, Vancouver, British Columbia, Canada, ACL anthology: W04-3321.

Laura KALLMEYER and Maribel ROMERO (2008), Scope and Situation Binding for LTAG, *Research on Language and Computation*, 6(1):3–52, 10.1007/s11168-008-9046-6.

Hans KAMP and Uwe REYLE (1993), *From Discourse to Logic*, Kluwer Academic Publishers.

Makoto KANAZAWA (2007), Parsing and Generation as Datalog Queries, in *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics (ACL)*, pp. 176–183, Association for Computational Linguistics, Prague, Czech Republic, ACL anthology: P07-1023.

Makoto KANAZAWA (2008a), Prefix-correct Earley parsing of mildly context-sensitive languages, Invited talk at the 15th Workshop on Logic, Language, Information and Computation (WoLLIC 2008). Edimburgh, Scotland.

Makoto KANAZAWA (2008b), A prefix-correct Earley recognizer for multiple context-free grammars, in Claire GARDENT and Anoop SARKAR, editors, *Proceedings of the Ninth International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+9)*, pp. 49–56, University of Tübingen, Tübingen, Germany, ACL anthology: W08-2307.

Makoto KANAZAWA (2009), Second-Order Abstract Categorial Grammars as Hyperedge Replacement Grammars, *Journal of Logic, Language, and Information*, 19(2):137–161, 10.1007/s10849-009-9109-6.

Makoto KANAZAWA (2017), Parsing and Generation as Datalog Query Evaluation, *IfCoLog Journal of Logics and their Applications*, 4(4):1103–1211, `http://www.collegepublications.co.uk/downloads/ifcolog00013.pdf#page=305`, special Issue Dedicated to the Memory of Grigori Mints.

Makoto KANAZAWA and Sylvain SALVATI (2007), Generating Control Languages with Abstract Categorial Grammars, in Gerald PENN, editor, *Proceedings of The 12th conference on Formal Grammar FG 2007*, CSLI Publications, `http://research.nii.ac.jp/~kanazawa/publications/control.pdf`.

Robert KASPER, Bernd KIEFER, Klaus NETTER, and K. VIJAY-SHANKER (1995), Compilation of HPSG to TAG, in *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics*, pp. 92–99, Association for Computational Linguistics, Cambridge, Massachusetts, USA, 10.3115/981658.981671, ACL anthology: P95-1013.

Gregory M. KOBELE (2007), Parsing Elliptical Structure, `https://home.uni-leipzig.de/gkobele/files/unpub/Kobele07ParsingEllipsis.pdf`.

Gregory M. KOBELE (2012), Idioms and Extended Transducers, in Han and Satta (2012), pp. 153–161, ACL anthology: W12-4618.

Gregory M. KOBELE and Jens MICHAELIS (2012), CoTAGs and ACGs, in Denis BÉCHET and Alexander DIKOVSKY, editors, *Logical Aspects of Computational Linguistics*, volume 7351 of *Lecture Notes in Computer Science*, pp. 119–134, Springer Berlin Heidelberg, ISBN 978-3-642-31261-8, 10.1007/978-3-642-31262-5_8.

Alexander KOLLER and Marco KUHLMANN (2011), A Generalized View on Parsing and Translation, in *Proceedings of the 12th International Conference on Parsing Technologies*, pp. 2–13, Association for Computational Linguistics, Dublin, Ireland, ACL anthology: W11-2902.

Alexander KOLLER and Marco KUHLMANN (2012), Decomposing TAG Parsing Algorithms Using Simple Algebraizations, in Han and Satta (2012), pp. 135–143, ACL anthology: W12-4616.

Marco KUHLMANN and Mathias MÖHL (2007), Mildly Context-Sensitive Dependency Languages, in *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pp. 160–167, Association for Computational Linguistics, Prague, Czech Republic, ACL anthology: P07-1021.

Joachim LAMBEK (1958), The Mathematics of Sentence Structure, *American Mathematical Monthly*, 65(3):154–170, 10.2307/2310058.

Scott MARTIN and Carl POLLARD (2014), A Dynamic Categorial Grammar, in Glyn MORRILL, Reinhard MUSKENS, Rainer OSSWALD, and Frank RICHTER, editors, *Formal Grammar*, volume 8612 of *Lecture Notes in Computer Science*, pp. 138–154, Springer Berlin Heidelberg, ISBN 978-3-662-44120-6, 10.1007/978-3-662-44121-3_9.

Aleksandre MASKHARASHVILI and Sylvain POGODALLA (2013), Constituency and Dependency Relationship from a Tree Adjoining Grammar and Abstract Categorial Grammar Perspective, in *6th International Joint Conference on Natural Language Processing*, pp. 1257–1263, The Asian Federation of Natural Language Processing, Nagoya, Japan, HAL open archive: hal-00868363. ACL anthology: I13-1179.

Richard MONTAGUE (1973), The Proper Treatment of Quantification in Ordinary English, in Jaakko HINTIKKA, Julius MORAVCSIK, and Patrick SUPPES, editors, *Approaches to Natural Language: Proceedings of the 1970 Stanford Workshop on Grammar and Semantics*, pp. 221–242, D. Reidel Publishing Co., Dordrecht, Holland.

Rebecca Nancy NESSON (2009), *Synchronous and Multicomponent Tree-Adjoining Grammars: Complexity, Algorithms, and Applications*, Ph.D. thesis, Harvard University, `https://pdfs.semanticscholar.org/754b/6acaf2660748967d1937a25222538207aabc.pdf`.

Rebecca Nancy NESSON and Stuart M. SHIEBER (2006), Simpler TAG Semantics Through Synchronization, in *Proceedings of the 11th Conference on Formal Grammar*, CSLI Publications, Malaga, Spain, `http://cslipublications.stanford.edu/FG/2006/nesson.pdf`.

Sylvain POGODALLA (2004a), Computing Semantic Representation: Towards ACG Abstract Terms as Derivation Trees, in *Seventh International Workshop on Tree Adjoining Grammar and Related Formalisms - TAG+7*, pp. 64–71, Vancouver, BC, Canada, HAL open archive: inria-00107768. ACL anthology: W04-3309.

Sylvain POGODALLA (2004b), Using and Extending the ACG technology: Endowing Categorial Grammars with an Underspecified Semantic Representation, in *Categorial Grammars*, pp. 197–209, Montpellier, France, HAL open archive: inria-00108117.

Sylvain POGODALLA (2007a), Ambiguïté de portée et approche fonctionnelle des TAG, in *Traitement Automatique des Langues Naturelles - TALN 2007*, pp. 325–334, Toulouse, France, HAL open archive: inria-00141913.

Sylvain POGODALLA (2007b), Generalizing a Proof-Theoretic Account of Scope Ambiguity, in *7th International Workshop on Computational Semantics - IWCS-7*, Tilburg, Netherlands, HAL open archive: inria-00112898.

Sylvain POGODALLA (2009), Advances in Abstract Categorial Grammars: Language Theory and Linguistic Modeling. ESSLLI 2009 Lecture Notes, Part II, HAL open archive: hal-00749297.

Florent POMPIGNE (2013), *Logical modelization of language and Abstract Categorial Grammars*, Theses, Université de Lorraine, HAL open archive: tel-00921040.

Owen RAMBOW, K. VIJAY-SHANKER, and David WEIR (1995), D-Tree Grammars, in *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics*, pp. 151–158, Association for Computational Linguistics, Cambridge, Massachusetts, USA, 10.3115/981658.981679, ACL anthology: P95-1021.

Owen RAMBOW, K. VIJAY-SHANKER, and David WEIR (2001), D-Tree Substitution Grammars, *Computational Linguistics*, 27(1):87–121, 10.1162/089120101300346813, ACL anthology: J01-1004.

Jim ROGERS (1999), Generalized Tree-Adjoining Grammar, in *Sixth Meeting on Mathematics of Language*, Orlando, Florida, `http://www.cs.earlham.edu/~jrogers/mol6.pdf`.

Sylvain SALVATI (2005), *Problèmes de filtrage et problèmes d'analyse pour les grammaires catégorielles abstraites*, Ph.D. thesis, Institut National Polytechnique de Lorraine.

Sylvain SALVATI (2006), Encoding second order string ACG with Deterministic Tree Walking Transducers, in Shuly WINTNER, editor, *Proceedings of The 11th conference on Formal Grammar FG 2006*, FG Online Proceedings, pp. 143–156, CSLI Publications, Malaga Espagne, `http://cslipublications.stanford.edu/FG/2006/salvati.pdf`.

Sylvain SALVATI (2007), On the complexity of Abstract Categorial Grammars, in *Proceedings of the 10th Conference on Mathematics of Language, MOL 10*, `http://wwwhomes.uni-bielefeld.de/mkracht/mol10/abstracts/acg_complexity.pdf`.

Sylvain SALVATI (2010), On the membership problem for non-linear Abstract Categorial Grammars, *Journal of Logic, Language and Information*, 19(2):163–183, 10.1007/s10849-009-9110-0.

Yves SCHABES and Stuart M. SHIEBER (1994), An Alternative Conception of Tree-Adjoining Derivation, *Computational Linguistics*, 20(1):91–124, ISSN 0891-2017, ACL anthology: J94-1004.

Stuart M. SHIEBER (1988), A Uniform Architecture for Parsing and Generation, in Dénes VARGHA, editor, *Proceedings of the 12th International Conference on Computational Linguistics*, volume 2, pp. 614–619, Budapest, ACL anthology: C88-2128.

Stuart M. SHIEBER (1993), The Problem of Logical-Form Equivalence, *Computational Linguistics*, 19(1):179–190, ACL anthology: J93-1008.

Stuart M. SHIEBER (1994), Restricting the Weak-Generative Capacity of Synchronous Tree-Adjoining Grammars, *Computational Intelligence*, 10(4):371–385, ISSN 1467-8640, 10.1111/j.1467-8640.1994.tb00003.x.

Stuart M. SHIEBER (2004), Synchronous Grammars as Tree Transducers, in *Proceedings of the 7th International Workshop on Tree Adjoining Grammar and Related Formalisms*, pp. 88–95, Vancouver, Canada, ACL anthology: W04-3312.

Stuart M. SHIEBER (2006), Unifying Synchronous Tree-Adjoining Grammars and Tree Transducers via Bimorphisms, in *Proceedings of the 11th Conference of the European Chapter of the Association for Computational Linguistics (EACL-06)*, pp. 377–384, Trento, Italy, ACL anthology: E06-1048.

Stuart M. SHIEBER (2014), Bimorphisms and synchronous grammars, *Journal of Language Modelling*, 2(1):51–104, 10.15398/jlm.v2i1.84.

Stuart M. SHIEBER and Yves SCHABES (1990), Synchronous Tree-Adjoining Grammars, in *Proceedings of the 13th International Conference on Computational Linguistics*, volume 3, pp. 253–258, Helsinki, Finland, 10.3115/991146.991191, `http://www.eecs.harvard.edu/~shieber/Biblio/Papers/synch-tags.pdf`.

Stuart M. SHIEBER, Gertjan VAN NOORD, Robert C. MOORE, and Fernando C. N. PEREIRA (1989), A Semantic-Head-Driven Generation Algorithm for Unification-Based Formalisms, in *Proceedings of the 27th Annual Meeting of the Association for Computational Linguistics*, pp. 7–17, Association for Computational Linguistics, Vancouver, British Columbia, Canada, 10.3115/981623.981625, ACL anthology: P89-1002.

Mark STEEDMAN (2001), *The Syntactic Process*, MIT Press.

Mark STEEDMAN and Jason BALDRIDGE (2011), Combinatory Categorial Grammar, in Robert BORSLEY and Kersti BÖRJARS, editors, *Non-Transformational Syntax: Formal and Explicit Models of Grammar*, chapter 5, Wiley-Blackwell.

Johan VAN BENTHEM (1986), *Essays in Logical Semantics*, volume 39 of *Studies in Linguistics and Philosophy*, Springer Netherlands, 10.1007/978-94-009-4540-1.

K. VIJAY-SHANKER (1987), *A Study of Tree Adjoining Grammars*, Ph.D. thesis, University of Pennsylvania.

K. VIJAY-SHANKER (1992), Using Descriptions of Trees in a Tree Adjoining Grammar, *Computational Linguistics*, 18(4):481–518.

K. VIJAY-SHANKER and Aravind K. JOSHI (1985), Some Computational Properties of Tree Adjoining Grammars, in *Proceedings of the 23rd Annual Meeting of the Association for Computational Linguistics*, pp. 82–93, Association for Computational Linguistics, Chicago, Illinois, USA, 10.3115/981210.981221, ACL anthology: P85-1011.

K. VIJAY-SHANKER and Aravind K. JOSHI (1988), Feature Structures Based Tree Adjoining Grammars, in Dénes VARGHA, editor, *Proceedings of the 12th International Conference on Computational Linguistics (COLING Budapest)*, volume 2, pp. 714–718, Budapest, ACL anthology: C88-2147.

K. VIJAY-SHANKER and Aravind K. JOSHI (1991), Unification-Based Tree Adjoining Grammars, Technical Report MS-CIS-91-25, University of

Pennsylvania Department of Computer and Information Science (CIS), `http://repository.upenn.edu/cis_reports/762`, paper 762.

Bonnie Lynn WEBBER (2004), D-LTAG: extending lexicalized TAG to discourse, *Cognitive Science*, 28(5):751–779, 10.1207/s15516709cog2805_6.

Bonnie Lynn WEBBER and Aravind K. JOSHI (1998), Anchoring a Lexicalized Tree-Adjoining Grammar for Discourse, in Manfred STEDE, Leo WANNER, and Eduard HOVY, editors, *Proccedings of the ACL/COLING workshop on Discourse Relations and Discourse Markers*, ACL anthology: W98-0315.

David J. WEIR (1988), *Characterizing Mildly Context-Sensitive Grammar Formalisms*, Ph.D. thesis, University of Pennsylvania.

Sean Michael WILLIFORD (1993), *Application of Synchronous Tree-Adjoining Grammar to Quantifier Scoping in English*, Bachelor's thesis, Harvard College, `http://nrs.harvard.edu/urn-3:HUL.InstRepos:10951941`.

Fei XIA (2001), *Automatic Grammar Generation from Two Different Perspectives*, Ph.D. thesis, University of Pennsylvania, `ftp://ftp.cis.upenn.edu/pub/fxia/thesis/thesis.pdf`.

Fei XIA, Chunghye HAN, Martha PALMER, and Aravind JOSHI (2000), Comparing Lexicalized Treebank Grammars Extracted from Chinese, Korean, and English Corpora, in *Second Chinese Language Processing Workshop*, pp. 52–59, Association for Computational Linguistics, Hong Kong, China, 10.3115/1117769.1117778, ACL anthology: W00-1208.

Fei XIA, Martha PALMER, and K. VIJAY-SHANKER (2005), Automatically Generating Tree Adjoining Grammars from Abstract Specifications, *Computational Intelligence*, 21(3):246–285, 10.1111/j.1467-8640.2005.00273.x.

XTAG RESEARCH GROUP (2001), A Lexicalized Tree Adjoining Grammar for English, Technical Report IRCS-01-03, IRCS, University of Pennsylvania, `ftp://ftp.cis.upenn.edu/pub/xtag/release-2.24.2001/tech-report.pdf`.

Ryo YOSHINAKA (2006), Linearization of Affine Abstract Categorial Grammars, in *proceedings of Fromal Grammar 2006*.

Ryo YOSHINAKA and Makoto KANAZAWA (2005), The Complexity and Generative Capacity of Lexicalized Abstract Categorial Grammars, in Philippe BLACHE, Edward STABLER, Joan BUSQUETS, and Richard MOOT, editors, *Proceedings of Logical Aspects of Computational Linguistics: 5th International Conference, LACL 2005*, volume 3492 of *LNCS*, pp. 330–348, Springer-Verlag GmbH, 10.1007/11422532_22.

This work is licensed under the Creative Commons Attribution 3.0 Unported License.
`http://creativecommons.org/licenses/by/3.0/`