Functional Approach to Tree-Adjoining Grammars and Semantic Interpretation: an Abstract Categorial Grammar Account

Sylvain Pogodalla*

September 19, 2017

Abstract

We present a functional interpretation of the substitution and adjunction operations of Tree Adjoining Grammars (TAGs). This allows us to encode TAGs within the formalism of Abstract Categorial Grammars (ACGs). In this encoding, 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 TAG can analyze. Because of the reversibility properties of ACGs, this provides a way to parse and generate with the same TAG encoded grammar. We then show some examples of how we can use and extend this representation of derivation trees, in particular to analyze idioms, subordinate clauses, and scope ambiguities. All the examples can be run with the ACG toolkit.

1 Motivations

1.1 Tree-Adjoining Grammar and Semantic Representation

The Tree-Adjoining Grammar (TAG) formalism (Joshi, Levy, and Takahashi 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, the *substitution* and the *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

*E-mail: sylvain.pogodalla@inria.fr

have been extensively studied[1] and provide TAG with appealing features for natural language processing.

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 locality* (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, Vijay-Shanker, and D. Weir 1995; Rambow, Vijay-Shanker, and D. Weir 2001), some others revisit the derivation tree definition (Schabes and Shieber 1994; Shieber 1994; Kallmeyer 2002; Joshi, Kallmeyer, and Romero 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; Kallmeyer and Romero 2008), functional tree interpretation (Pogodalla 2004a; Pogodalla 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; Pogodalla 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)

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

encoding. But we introduce an intermediate level, of generalized derivations, on which we base our syntax-semantics interface. 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).

Contrary to Pogodalla (2004a) (and Kallmeyer and Romero (2004) and Kallmeyer and Romero (2008)), the semantic modelling we propose does not rely on an intermediate underspecification language. We show instead that it is not required to model long distance dependencies or quantification. We also introduce and precisely describe the syntax-semantics modelling for adjectives (without multiple adjunctions), control verbs, subordinate clauses. We also discuss the encoding of features, and we show how the approach easily extends, without requiring the design of new parsing algorithms to other phenomena such as idioms[2] 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.

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.[3] 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 performed in polynomial time, including for the modellings that go beyond TAG analysis (except the type-raising account of quantification) without having to design new parsers.

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 example grammars of this article.[4]

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 **??**, we introduce the ACGs that are necessary for the encoding of the syntax-semantics interface: the ACG relating strings and derived trees in Section **??**; the ACG relating derived trees and generalized derivation trees in Section **??** and Section **??**. We show that these generalized derivations over-generate wrt. 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.

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

[3] The toolkit is available at `http://calligramme.loria.fr/acg/#Software`.

[4] The example files are available at `https://hal.inria.fr/hal-01242154/file/RR-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

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

In Section ?? 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 4).

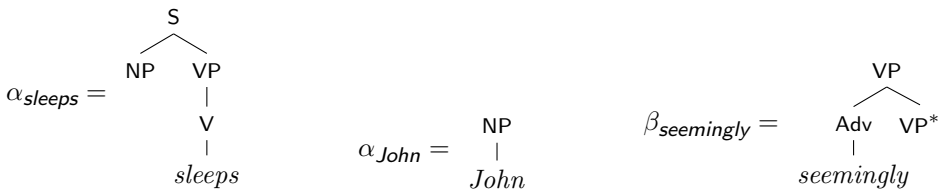
Then, in Section 5, we take advantage of the architecture we propose and give examples of2 modellings that this framework offers and that are beyond TAG. We finally discuss the relation of our approach to the syntax-semantics interface for TAG with other ones, in particular the ones using synchronous grammars or feature unification (Section 6).

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. Figure 1 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.[5] Figure 2(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 Fig. 2(b), where the Gorn address labels the edge between the two nodes, each of them being labeled by the name of the trees.

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 Fig. 3(a) shows. The associated derivation tree of Fig. 3(b) records the adjunction with a dotted edge



(a) Initial tree with one substitution node

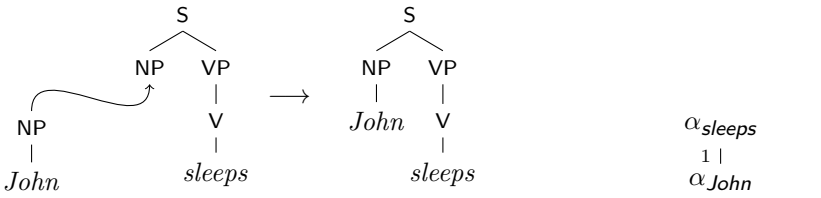
(b) Initial tree with no substitution node

(c) Auxiliary tree

Figure 1: TAG elementary trees

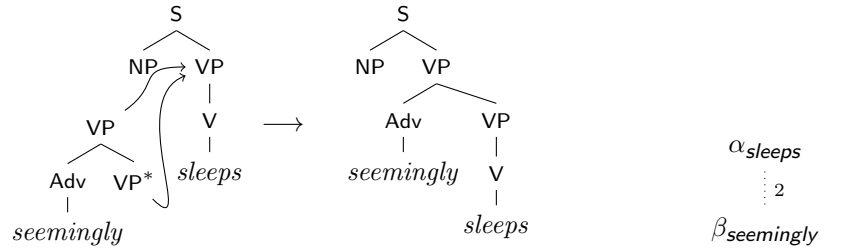
Then Fig. 4 shows a TAG analysis of *John seemingly sleeps* with both operations and the associated derivation tree.

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



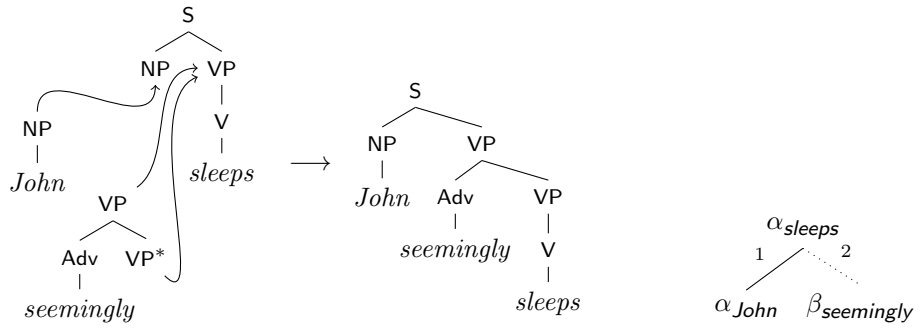
(a) Substitution (b) Derivation tree

Figure 2: Substitution operation



(a) Adjunction (b) Derivation tree

Figure 3: Adjunction operation



(a) Adjunction and substitution (b) Derivation tree

Figure 4: 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. As most of the examples we present rely on λ-terms and their types, and running them with the ACG toolkit basically uses their syntax, we formally present them.

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:

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

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

Definition 3 (λ-Terms)**.** Let X be an infinite countable 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 $\lambda^{\circ} x.t \in \Lambda(\Sigma)$;
- 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°) and a (usual) intuitionistic λ-abstraction (denoted by λ). There also are 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°), 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.

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

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

$$\frac{\Gamma; \Delta \vdash_\Sigma t : \beta}{\Gamma, x : \alpha; \Delta \vdash_\Sigma t : \beta}\ (\text{weak.})$$

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

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

Table 1: Typing rules for deriving typing judgments

Remark 1. 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:

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

where $c : \alpha \to \alpha \to \beta$ is a constant of the signature Σ. 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.

Definition 5 (Linear and Almost Linear λ-Terms)**.** A term without any λ where each λ° 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° 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.

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

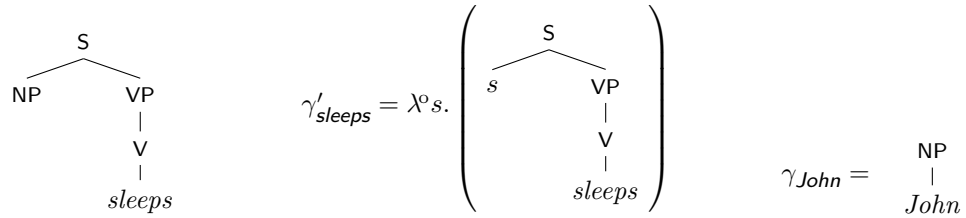
- $\mathsf{ord}(a) = 1$ if $a \in A$
- $\mathsf{ord}(\alpha \multimap \beta) = \max(1 + \mathsf{ord}(\alpha), \mathsf{ord}(\beta))$ otherwise

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

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

2.2.1 Substitution as Function Application

The ability for the tree of Fig. 5(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



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

Figure 5: Functional interpretation of the substitution operation

can represent it as the function γ'_{sleeps} shown in Fig. 5(b) with $\gamma'_{sleeps} : T \multimap T$. A tree where no substitution can occur can be represented as $\gamma_{John} : T$ (see Fig. 5(c)).

Applying the function γ'_{sleeps} to the simple tree γ_{John} of Fig. 5(c) and performing β-reduction gives the expected result as (1) shows.

$$\gamma'_{sleeps}\ \gamma_{John} = \left(\lambda^{o} s. \begin{array}{c} \text{S} \\ s \quad \text{VP} \\ \quad\ \ \text{V} \\ \quad\ \ sleeps \end{array} \right) \begin{array}{c} \text{NP} \\ John \end{array} \rightarrow_{\beta} \begin{array}{c} \text{S} \\ \text{NP} \quad \text{VP} \\ John \quad\ \ \text{V} \\ \quad\ \ sleeps \end{array} \tag{1}$$

2.2.2 Adjunction as Function Application

In order to deal with the adjunction operation, we first observe what happens to the auxiliary tree in Fig. 3: a subtree of the tree it is adjoined to (the one rooted by VP) is substituted at its VP* foot node. This means that the auxiliary tree of Fig. 6 also behaves as a function from trees to trees and can be represented as in Fig. 7(a) 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 Fig. 7(b). Note the higher-order type of γ''_{sleeps}.

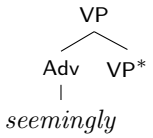
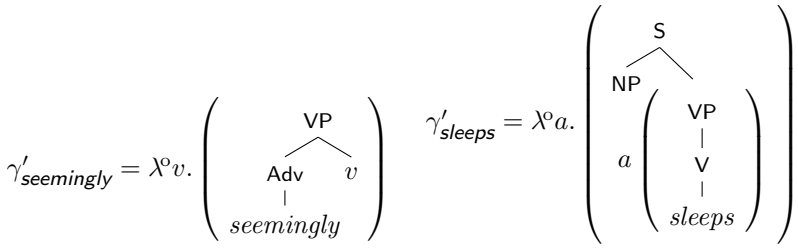


Figure 6: TAG auxiliary tree

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

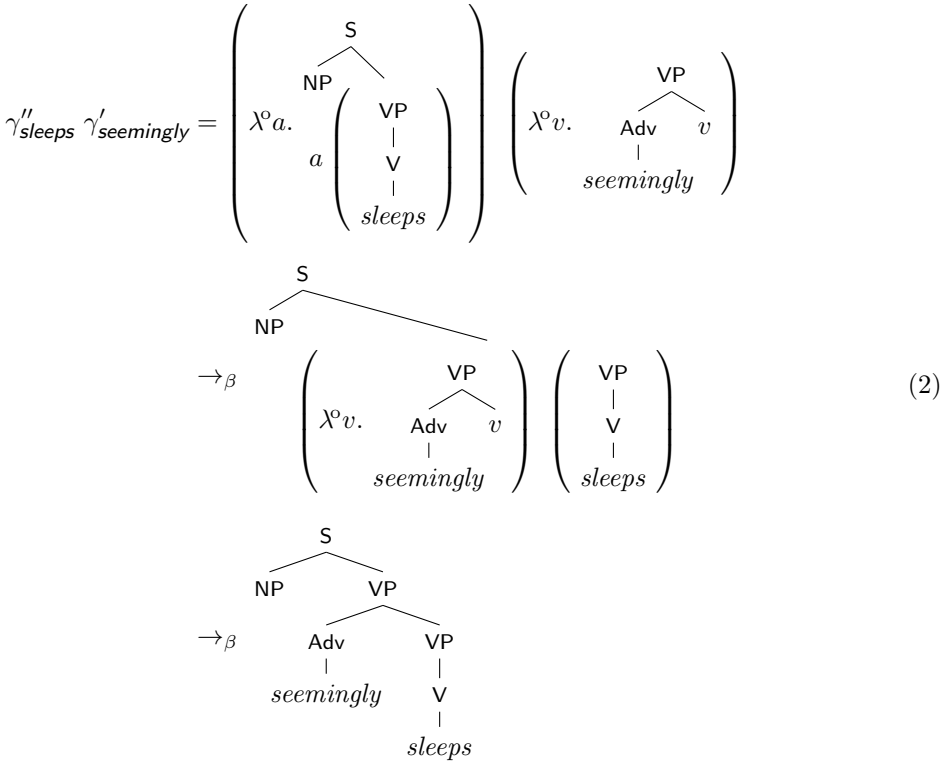


(a) Function from trees to trees

(b) Elementary tree ready to accept an adjunction

Figure 7: Functional interpretation of the adjunction operation

as (2) shows.



(2)

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 Fig. 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 Fig. 4 with the β-reduction of $\gamma'''_{sleeps}\ \gamma'_{seemingly}\ \gamma_{John}$ shown in (3).

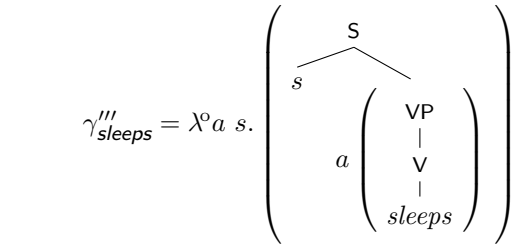
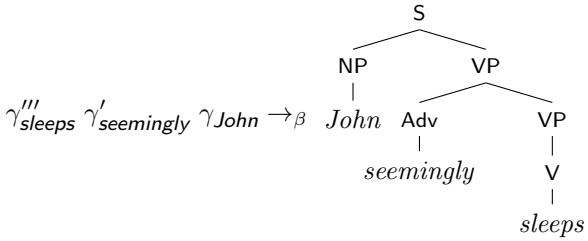
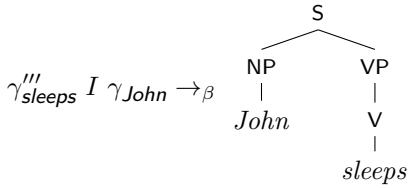


Figure 8: Elementary tree representation available to substitution and adjunction operations



(3)

Remark 2 (No adjunction)**.** Typing γ'''_{sleeps} with $(T \multimap T) \multimap T \multimap T$ makes it require an adjunction (the first $(T \multimap T)$ parameter) 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 $I = \lambda^{o} x.x : T \multimap T$. Then (4) holds.



(4)

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 Fig. 9 shows.[6]

Remark 3 (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 Sect. **??**. 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 Fig. 10 in order to analyze sentences such as *John usually seemingly sleeps* as in Fig. 11 with the term $\gamma_{sleeps}\ I\ (\gamma_{seemingly}\ (\gamma_{usually}\ I))\ \gamma_{John}$.

[6] 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.

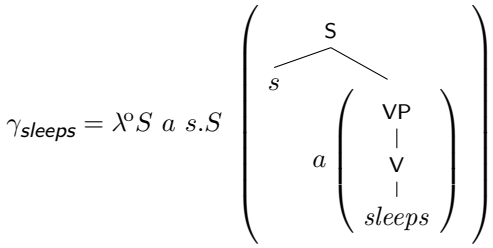


Figure 9: Encoding of α_{sleeps} available to substitution and adjunctions both at the VP and at the S nodes

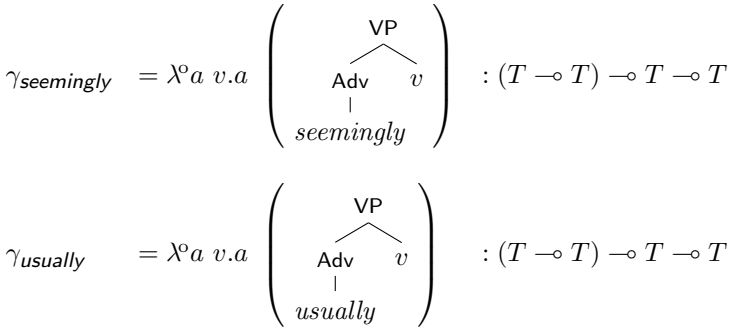


Figure 10: Auxiliary tree representation available to adjunction operations

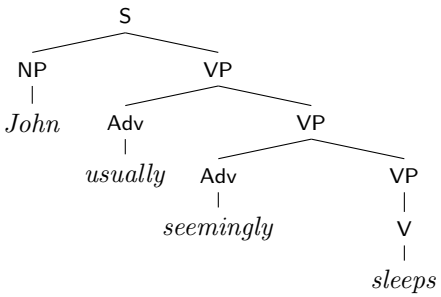


Figure 11: A TAG analysis of *John usually seemingly sleeps*

2.3 Trees and Strings as λ-terms

So far, we did not make it 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 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 $\stackrel{\Delta}{=}$ 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 $\sigma \stackrel{\Delta}{=} (o \multimap o)$. We also define two other terms:

- $+ \stackrel{\Delta}{=} \lambda^o f\ g.\lambda^o z.f(g\ z)$ (the function composition, used with an infix notation) to represent the *concatenation*;
- $\epsilon \stackrel{\Delta}{=} \lambda^o x.x$ (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 4. If a and b are two strings, $a{+}b = \lambda^o z.a\ (b\ z)$. In this article, we usually don't 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, Levy, and Takahashi 1975). Hence, the same node label could be used to represent nodes of different arity. For instance, in Fig. 1(a), the VP node has arity 1 whereas the VP node of Fig. 1(c) has arity 2.

We prefer to represent trees as terms over a ranked alphabet as in (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 $\mathcal{F}_a = (\mathcal{F}, arity)$, where $arity$ is a mapping from $\mathcal{F}$ to $\mathbb{N}$, we use the following higher-order signature $\Sigma^{\mathcal{F}_a}_{trees} = \langle A_T, \mathcal{F}, \tau^{\mathcal{F}_a}_T \rangle$ where:

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

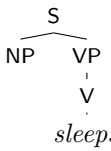
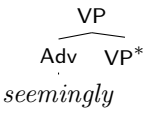
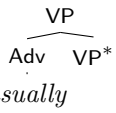
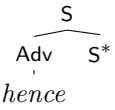
	Terms of $\Lambda(\Sigma_{trees})$	Corresponding TAG tree
γ_{John}	$= \mathsf{NP}_1\ John$ $: T$	
γ_{sleeps}	$= \lambda^{\circ} S\ a\ s.S\ (\mathsf{S}_2\ s\ (a\ (\mathsf{VP}_1\ (\mathsf{V}_1\ sleeps))))$ $: (T \multimap T) \multimap (T \multimap T) \multimap T \multimap T$	
$\gamma_{seemingly}$	$= \lambda^{\circ}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

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

For instance, the TAG elementary trees δ_{anchor}[7] of our running examples can be modeled as the functions (or terms) γ_{anchor} built on the signature Σ_{trees} as Table 2 shows.[8] Then (5) shows that $\gamma_4 = \gamma_{sleeps}\ I\ (\gamma_{seemingly}\ I)\ \gamma_{John}$ corresponds (modulo β) to the tree of Fig. 4.

$$\begin{aligned} \gamma_4 &= \gamma_{sleeps}\ I\ (\gamma_{seemingly}\ I)\ \gamma_{John} \\ &\rightarrow_{\beta} \mathsf{S}_2\ (\mathsf{NP}_1\ John)\ (\mathsf{VP}_2\ (\mathsf{Adv}_1\ seemingly)\ (\mathsf{VP}_1\ (\mathsf{V}_1\ sleeps))) \end{aligned} \tag{5}$$

We now show how to relate the tree represented by the term $\gamma_{sleeps}\ I\ (\gamma_{seemingly}\ (\gamma_{usually}\ I))\ \gamma_{John} : T$ to the string *John usually seemingly sleeps* represented by the term $John + usually + seemingly + sleeps : \sigma$ using an *interpretation* of the former defined with an ACG.

3 Abstract Categorial Grammars

A key feature of ACGs is to provide the user direct control over 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

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

[8] 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.

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, 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 $\hat{G}(\lambda^{\circ} x.t) = \lambda^{\circ} x.\hat{G}(t)$;

- F and G are such that for all $c \in C_1$, $\quad \vdash_{\Sigma_2} G(c) : F(\tau_1(c))$ 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.

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

Given an ACG $\mathscr{G}_{name} = \langle \Sigma_1, \Sigma_2, \mathscr{L}_{name}, \mathsf{S} \rangle$, we use the following notational variants to express that the interpretation of the type α is the type β (resp. the interpretation of the term t is the term u): $\mathscr{L}_{name}(\alpha) = \beta$, $\mathscr{G}_{name}(\alpha) = \beta$, $\alpha :=_{name} \beta$, and $[\![\alpha]\!]_{name} = \beta$ (resp. $\mathscr{L}_{name}(t) = u$, $\mathscr{G}_{name}(t) = u$, $t :=_{name} u$, 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

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

The *object language* is defined by

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

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 $t \in \mathcal{A}(\mathscr{G})$ such that $u = \mathscr{L}(t)$. 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 allows us to relate the derived trees and the semantic expressions that have the same derivation structures. We use this in Sect. **??** 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 $\mathscr{G}_{yield} \circ \mathscr{G}_{derived\ trees}$ composition in Figure 12, provides modularity. It also allows one to control admissible intermediate structures. For instance, the abstract language of $\mathscr{G}_{yield}$ may contain too many structures. If the object language of $\mathscr{G}_{derived\ trees}$ is a strict subset of this abstract language, then the object language of $\mathscr{G}_{yield} \circ \mathscr{G}_{derived\ trees}$ is a subset of the object language of $\mathscr{G}_{yield}$. We take advantage of this property in Sect. **??** 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 Sect. **??**. This ACG composition corresponds to the typical ACG way to control admissible parse structures.

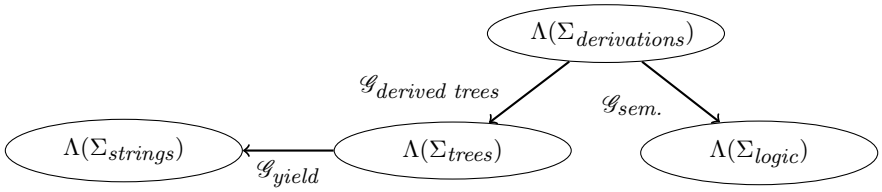


Figure 12: ACG composition for control and syntax-semantic interface

3.2 Formal Properties of ACGs

The formal properties of ACGs have been intensively studied. 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.

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

Second-order ACGs is a particular class of 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 sums up some of the formal properties of second-order ACGs (de Groote and Pogodalla 2004; Salvati 2006; Kanazawa and Salvati 2007; Kanazawa 2009).

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

Table 3: The hierarchy of second-order ACGs

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). It 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.[9]

The computational properties of lexicon inversion for ACGs have been studied for different classes of λ-terms.[10] 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, van Noord, et al. 1989) in a very general setting.

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).[11] 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 only implements abstract term interpretation, but no parsing.

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

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

[11] 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 $\mathcal{G} = (\mathcal{I}, \mathcal{A})$ where $\mathcal{I}$ is the set of initial trees and $\mathcal{A}$ the set of auxiliary trees. The labels of the trees in $\mathcal{I} \cup \mathcal{A}$ 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 $\mathcal{V}$ is the associated ranked alphabet.

4.1 Derived Trees and Strings

In the constructions of Sect. 2.3, we introduced two higher-order signatures: $\Sigma_{strings} = \Sigma^C_{strings}$ and $\Sigma_{trees} = \Sigma^{\mathcal{V}}_{trees}$. 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 $X_n \in V \setminus C$, $\mathscr{L}_{yield}(X_n) = \lambda^{\circ} x_1 \ \dots \ x_n . x_1 + \dots + x_n$ (a tree labeled by a non-terminal symbol is interpreted by the concatenation of 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, (**??**) shows that the yield of the tree represented by $\gamma_4 = \gamma_{\mathit{sleeps}}\ I\ (\gamma_{\mathit{seemingly}}\ I)\ \gamma_{\mathit{John}}$ (p. 14) actually is *John*+*seemingly*+*sleeps* (which can be rephrased as $\gamma_{\mathit{sleeps}}\ I\ (\gamma_{\mathit{seemingly}}\ I)\ \gamma_{\mathit{John}} \coloneqq_{yield} \mathit{John} + \mathit{seemingly} + \mathit{sleeps}$).

$$\begin{aligned} \mathscr{G}_{yield}(\gamma_{\mathit{John}}) &= \mathscr{G}_{yield}(\mathsf{NP}_1\ \mathit{John}) \\ &\rightarrow_{\beta} \mathit{John} \end{aligned} \tag{6}$$

$$\mathscr{G}_{yield}(\gamma_{\mathit{sleeps}}\ I\ (\gamma_{\mathit{seemingly}}\ I)) = \mathscr{G}_{yield}(\lambda^{\circ} s.s + \mathit{seemingly} + \mathit{sleeps}) \tag{7}$$

$$\mathscr{G}_{yield}(\gamma_4) = \mathit{John} + \mathit{seemingly} + \mathit{sleeps} \tag{8}$$

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, $\gamma_{??} = \gamma_{\mathit{seemingly}}\ I\ \gamma_{\mathit{John}}$ is a well-typed term of type T corresponding to the tree of Fig. **??** as (**??**) shows. Consequently, its interpretation *seemingly*+ *John* belongs to the object language.

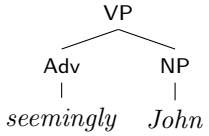


Figure 13: Tree in the abstract language of $\mathscr{G}_{yield}$

$$\begin{aligned} \gamma_{??} &= \gamma_{\mathit{seemingly}}\ I\ \gamma_{\mathit{John}} \\ &\rightarrow_{\beta} \mathsf{VP}_2\ (\mathsf{Adv}_1\ \mathit{seemingly})\ (\mathsf{NP}_1\ \mathit{John}) \\ &\coloneqq_{yield} \mathit{seemingly} + \mathit{John} \end{aligned} \tag{9}$$

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 $\mathcal{A}(\mathscr{G}_{yield})$ (see Fig. 12). Consequently, the object language of $\mathscr{G}_{yield} \circ \mathscr{G}_{derived\ trees}$ is a subset (strict in this case, as expected) of $\mathcal{O}(\mathscr{G}_{yield})$. $\mathscr{G}_{derived\ trees}$ 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 $\gamma_{\mathsf{seemingly}}$ 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 $\mathcal{G}$. Then, for any $\delta_{\mathsf{anchor}} \in \mathcal{I} \cup \mathcal{A}$ an elementary tree of $\mathcal{G}$, 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); [12]
- 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 $\delta_{\mathsf{anchor}} \in \mathcal{I}$ is an initial tree, and $\alpha = Z' \multimap Z$ with Z' the label of the foot node if $\delta_{\mathsf{anchor}} \in \mathcal{A}$ is an auxiliary tree.[13] 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 (**??**)[14] from the elementary trees of Fig. 1.

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

For each non-terminal X of the TAG grammar where an adjunction can occur, we also define $I_X : X \multimap X$ as in (**??**). 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 $C_{derivations} = (\cup_{\delta_{\mathsf{anchor}} \in \mathcal{I} \cup \mathcal{A}}\ c_{\mathsf{anchor}}) \cup (\cup_{X \in V^0} I_X)$ and $\tau_{derivations}$ is the associated typing function defined as above. The typing provided by $\Sigma_{derivations}$ now disallows the application of $c_{\mathsf{seemingly}}\ I_{\mathsf{VP}} : \mathsf{VP} \multimap \mathsf{VP}$ to $c_{\mathsf{John}} : \mathsf{NP}$.

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

[12] 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) and Vijay-Shanker and Joshi (1991)). See note **??**.

[13] In standard TAG, we typically have $Z = Z'$. However, we shall see examples in Sections **??** and 4 where such a distinction is relevant.

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

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

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

4.3.2 Interpretation of Derivations as Derived Trees: The $\mathscr{G}_{\mathit{derived\ trees}}$ ACG

In order to define $\mathscr{G}_{\mathit{derived\ trees}} = \langle \Sigma_{\mathit{derivations}}, \Sigma_{\mathit{trees}}, \mathscr{L}_{\mathit{derived\ trees}}, \mathsf{S} \rangle$ we are left with defining $\mathscr{L}_{\mathit{derived\ trees}}$. All the atomic types ($\mathsf{S}$, VP, etc.) are interpreted as trees (i.e., with the T type). And for a TAG elementary tree δ_{anchor}, the constant c_{anchor} is interpreted as γ_{anchor} (defined in Sect. 2.3.2). This leads us to the interpretations of Table **??**.

In Sect. **??**, we noticed that $\gamma_4 = \gamma_{\mathit{seemingly}}\ I\ \gamma_{\mathit{John}} : T \in \mathcal{A}(\mathscr{G}_{\mathit{yield}})$ (see **??**). By definition of the object language of an ACG, its interpretation $\mathscr{G}_{\mathit{yield}}(\gamma_4) = \mathit{seemingly} + \mathit{John}$ is such that $\mathscr{G}_{\mathit{yield}}(\gamma_4) \in \mathcal{O}(\mathscr{G}_{\mathit{yield}})$.

However, $\gamma_4 \notin \mathcal{O}(\mathscr{G}_{\mathit{derived\ trees}})$. Indeed, there is no c_4 such that $\mathscr{G}_{\mathit{derived\ trees}}(c_4) = \gamma_4$. A simple argument using the linearity of the interpretation shows that only $c_{\mathit{seemingly}}$ (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, $\mathit{seemingly} + \mathit{John} \notin \mathcal{O}(\mathscr{G}_{\mathit{yield}} \circ \mathscr{G}_{\mathit{derived\ trees}})$ as is expected from the TAG grammar.

4.3.3 $\mathscr{G}_{\mathit{derived\ trees}}$ Abstract Terms and Generalized Derivation Trees

It is interesting to note that abstract terms of $\mathscr{G}_{\mathit{derived\ trees}}$ describe the way the encoding of trees in Σ_{trees} can combine. We can see this combination in terms such as $\gamma_{??} = \gamma_{\mathit{sleeps}}\ I\ (\gamma_{\mathit{seemingly}}\ I)\ \gamma_{\mathit{John}}$, but it is in some sense an artifact of the definition we gave: $\gamma_{??}$ β-reduces to a tree that does not show this structure anymore. However, a term such as $c_{??} = c_{\mathit{sleeps}}\ I_{\mathsf{S}}\ (c_{\mathit{seemingly}}\ I_{\mathsf{VP}})\ c_{\mathit{John}}$ 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, $c_{??}$ keeps track of the adjunction of I_{S} on c_{sleeps}, of the adjunction of I_{VP} on $c_{\mathit{seemingly}}$, of the adjunction of the latter result on c_{sleeps}, and of the substitution of c_{John}. And the relation $\mathscr{G}_{\mathit{derived\ trees}}(c_{??}) = \gamma_{??}$ expresses the relation between the record of these operations and the resulting derived tree.

We can represent $c_{??}$ as a tree (see Fig. **??**): each node corresponds to a constant, possibly applied to the terms represented by the children of the node. It makes explicit the relation to the

TAG derivation trees (Fig. **??**). They are in a one to one correspondence despite the following 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.



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

Figure 14: Derivation representations

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;[15]
- 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.

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

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

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

Remark 5 ($\mathscr{G}_{derived\ trees}$ terms and description of trees)**.** Let us have a look at the $(X \multimap X)$ type of the parameter of an abstract constant of $\mathscr{G}_{derived\ trees}$ and at its interpretation. In c_{sleeps} for instance, the parameter with type $(\mathsf{VP} \multimap \mathsf{VP})$ is interpreted by the a variable of γ_{sleeps} (see Table **??**). The position of a in the term $S\ (\mathsf{S}_2\ s\ (a\ (\mathsf{VP}_1\ (\mathsf{V}_1\ sleeps))))$ makes it explicit that the result of a applied to $(\mathsf{VP}_1\ (\mathsf{V}_1\ sleeps))$, 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 parameter (here $\mathsf{VP}_1\ (\mathsf{V}_1\ sleeps)$), as in the tree descriptions of Vijay-Shanker (1992): the root node of the parameter 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.

[15] With some reservations that Sect. **??** clears up, though. See Rem. **??**.

Remark 6 (Generalized derivations and TAG derivation trees)**.** It should be noted that $\mathscr{G}_{\textit{derived trees}}$ and $\mathscr{G}_{\textit{yield}} \circ \mathscr{G}_{\textit{derived trees}}$ 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 $\mathscr{G}_{\textit{derived trees}}$ constrains the string language more than $\mathscr{G}_{\textit{yield}}$ 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 **??**, is to further control $\mathcal{A}(\mathscr{G}_{\textit{derived trees}})$ with another ACG $\mathscr{G}_{TAG}$ such that $\mathcal{O}(\mathscr{G}_{TAG}) \subset \mathcal{A}(\mathscr{G}_{\textit{derived trees}})$ just as $\mathscr{G}_{\textit{derived trees}}$ allows us to control $\mathcal{A}(\mathscr{G}_{\textit{yield}})$. The general architecture is then the one that Fig. **??** 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 **??**.

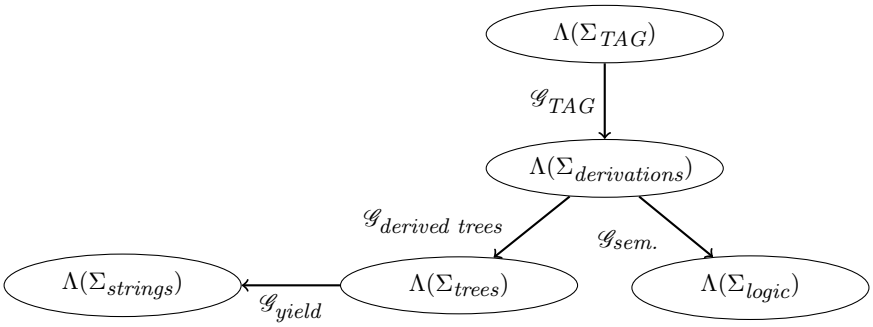


Figure 15: ACG composition for control and syntax-semantics interface

5 Semantic Construction

In the previous section, we defined a signature $\Sigma_{\textit{derivations}}$ to represent derivation structures as terms of $\Lambda(\Sigma_{\textit{derivations}})$. We now use this signature as pivot to transfer these structures into semantic representations. From a practical point of view, as mentioned in Sect. 3.1, it amounts to defining an ACG $\mathscr{G}_{sem.} = \langle \Sigma_{\textit{derivations}}, \Sigma_{logic}, \mathscr{L}_{sem.}, \mathsf{S} \rangle$ and composing it with $\mathscr{G}_{\textit{derived trees}}$ thanks to the shared abstract vocabulary $\Sigma_{\textit{derivations}}$. 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, Koller, and Niehren 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.

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 **??** with $A_{logic} = \{e, t\}$ the atomic types for *entities* and *truth values* respectively. As usual, we note 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

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**	$: e \multimap t$

Table 5: The vocabulary Σ_{logic}

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

(a) Interpretation of the atomic types

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

(b) Interpretation of the constants

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

use the non-linear implication as a lot of semantic formula (e.g., adjectives, quantifiers...) use non linearity of entities. But we stay within the fragment of almost linear terms as only terms of atomic type are duplicated.

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 **??**. The interpretation of the constants follows, as in Table **??**.[16] 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 $\mathscr{G}_{sem.}(c_{\mathit{anchor}})$ (e.g., the type of c_{John} is NP, hence $[\![c_{\mathit{John}}]\!]_{sem.} : [\![\mathsf{NP}]\!]_{sem.} = (e \to t) \multimap t$). But the reader can check that this proviso holds.

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

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

[16] 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).

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 as expected in the 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) 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 Fig. **??** and results in the derived tree and derivation tree of Fig. **??** for (**??**). 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 (**??**) as the derivation tree, shown Fig. **??**, makes $\alpha_{\mathit{to\ love}}$ 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, (**??**) and the derivation tree of Fig. **??** 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**).

To semantically account for these phenomena, we first extend $\Sigma_{\mathit{derivations}}$ and $\mathscr{G}_{\mathit{derived\ trees}}$ to represent the trees of Fig. **??**. Table **??** shows the new constants and their interpretations. The terms $c_{??}$, $c_{??}$, and $c_{??}$ in (**??**) represent the derivation trees of Fig. **??**, **??**, and **??** respectively. We leave it to the reader to check that the $\mathscr{G}_{\mathit{derived\ trees}}$ interpretations of theses terms are the derived trees of the corresponding figures **??**, **??**, and **??** respectively.

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

We now need to define the $\mathscr{G}_{\mathit{sem.}}$ interpretation that provides the expected semantic dependencies. Table **??** shows the lexical semantics fulfilling the requirements. The constant c_{loves} scopes over c_{claims} in the term $c_{??} = c_{\mathit{loves}}\ (c_{\mathit{claims}}\ I_{\mathsf{S}}\ I_{\mathsf{VP}}\ c_{\mathit{Paul}})\ I_{\mathsf{VP}}\ c_{\mathit{John}}\ c_{\mathit{Mary}}$, as does α_{loves} over α_{claims} in the derivation tree of Fig. **??**. However, looking at $[\![c_{\mathit{loves}}]\!]_{\mathit{sem.}}$ in Table **??**, 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 $(c_{\mathit{claims}}\ I_{\mathsf{S}}\ I_{\mathsf{VP}}\ c_{\mathit{Paul}})$, 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 $[\![c_{\mathit{to\ love?}}]\!]_{\mathit{sem.}}$, 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 $\alpha_{\mathit{to\ love?}}$ (see Fig. **??**), typically $\lambda^{\circ}P.\mathbf{WHO}\ P : (e \multimap t) \multimap t$, making **WHO** eventually take scope over all the other predicates.

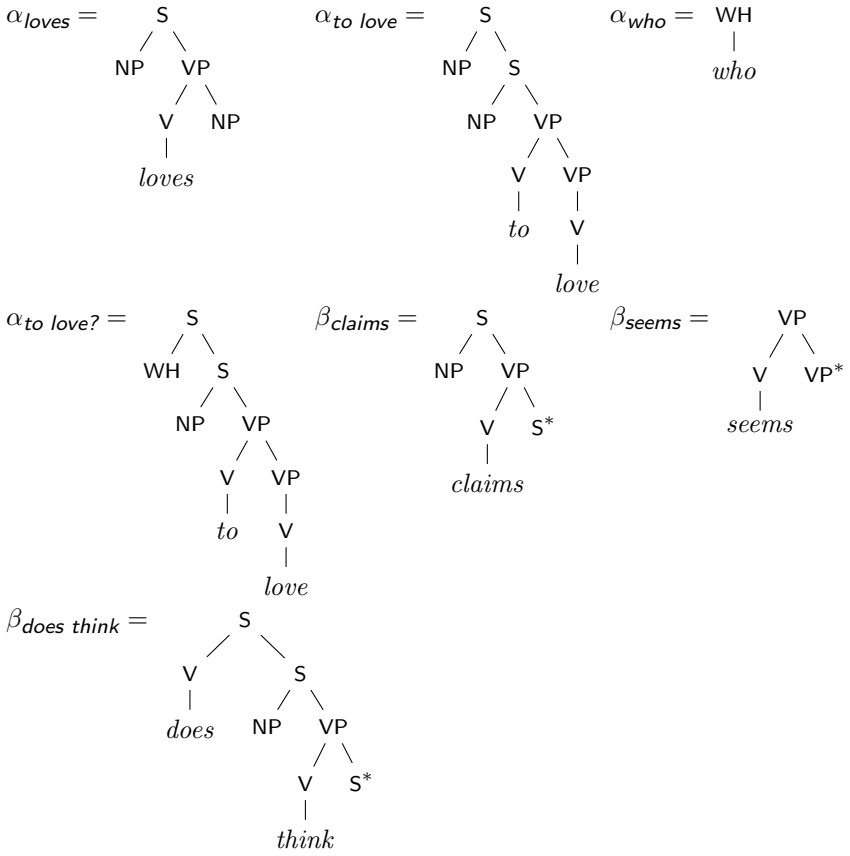
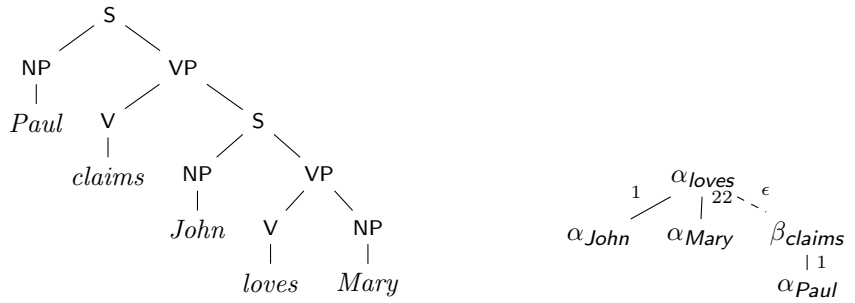


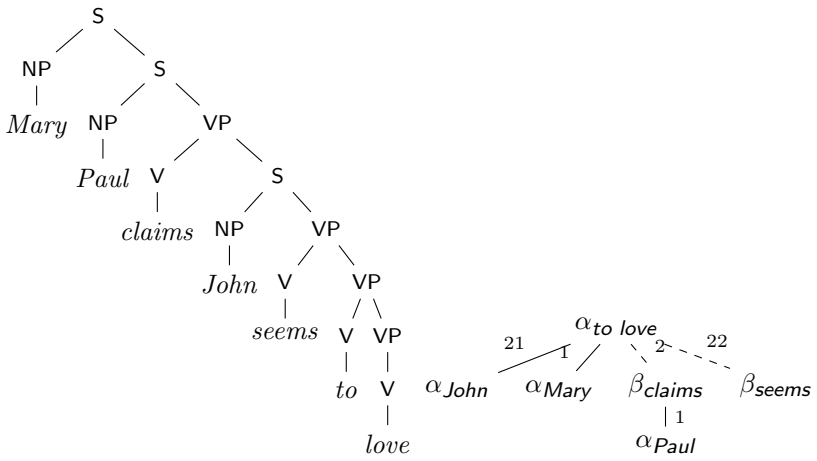
Figure 16: TAG elementary trees for long distance dependencies



(a) Derived tree

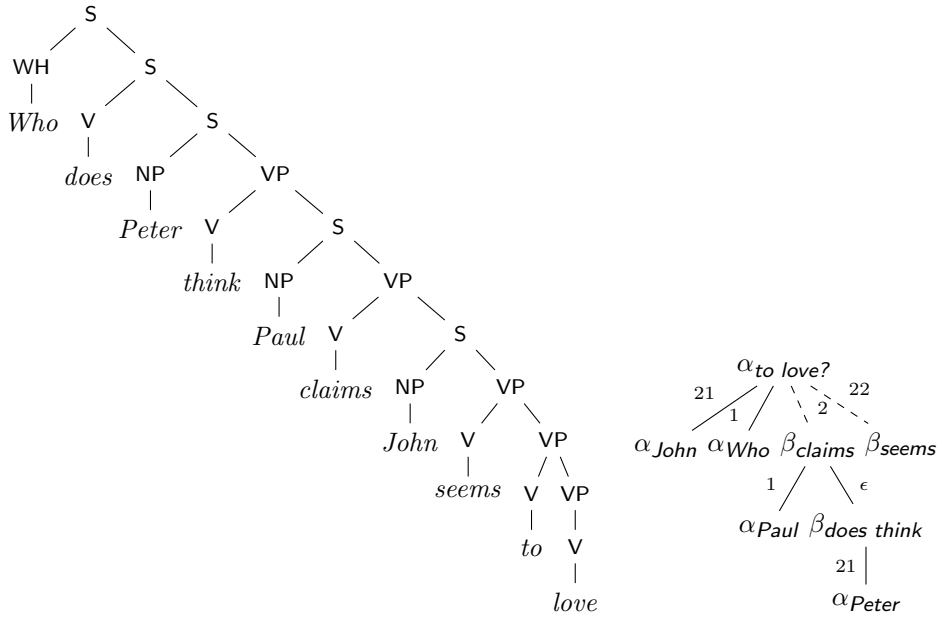
(b) Derivation tree

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*

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

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

Equation (**??**) shows that the $\mathscr{G}_{sem.}$ interpretation builds the expected semantics with the required scope inversions.

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

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 Fig. **??** provide the TAG elementary trees for the TAG analysis for (**??**) (resp. for (**??**)) shown in Fig. **??** (resp. in Fig. **??**).

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

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

(20) everyone loves someone

(21) every man loves some woman

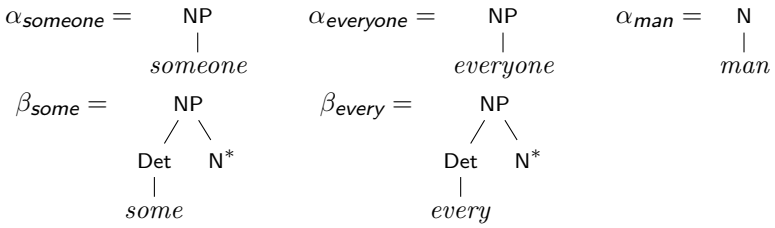


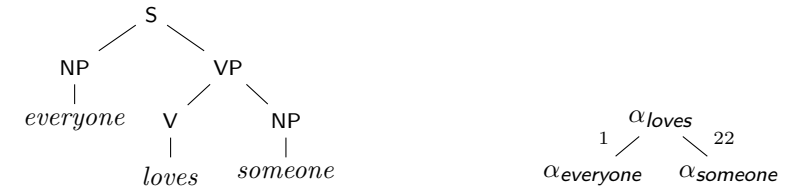
Figure 20: Determiners and quantifiers

Remark 7. We follow standard TAG analysis for determiners (Abeillé 1993; XTAG Research Group 2001) where the latter adjoin on initial trees anchored by nouns. While the auxiliary trees β_{some} and β_{every} of Fig. **??** 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 *plus* positive (NP[+]) determiner features, while the N label is a shorthand for the NP TAG category *plus* negative (NP[−]) determiner features. Kasper et al. (1995) and others (Rogers 1999; Kahane, Candito, and de Kercadio 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 4, 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 label 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 **??**).

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 parameter type (i.e., substitution node),[17] we keep only the constant with the last

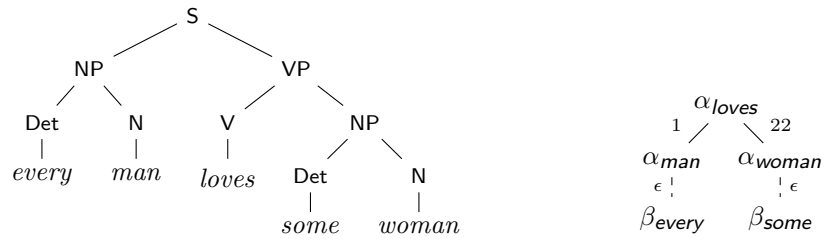
[17] This of course depends on the grammar. In any case, if there were such a constant, it would not allow for performing first the adjunction of a determiner on the noun.

type.



(a) Derived tree (b) Derivation tree

Figure 21: TAG analysis of *everyone loves someone*



(a) Derived tree (b) Derivation tree

Figure 22: TAG analysis of *everyone loves someone*

The derivation trees of Fig. **??** and **??** 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 seeming mismatch can be dealt with, we extend $\Sigma_{derivations}$ with the constants of Table **??**. This table also provides the interpretation of these constants by $\mathscr{G}_{derived\ trees}$ modelling the elementary trees of Fig. **??**. The terms of (**??**) belong to $\mathcal{A}(\mathscr{G}_{derived\ trees})$ and represent the derivation trees of Fig. **??** and **??**. Equation (**??**) shows they are interpreted as the derived trees of the same figures.

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

$$\begin{aligned} c_{??} \coloneqq_{derived\ trees} \mathsf{S}_2\ (\mathsf{NP}_1\ everyone)\ (\mathsf{VP}_2\ (\mathsf{V}_1\ loves)\ (\mathsf{NP}_1\ someone)) \\ c_{??} \coloneqq_{derived\ trees} \mathsf{S}_2\ (\mathsf{NP}_2\ (\mathsf{Det}_1\ every)\ (\mathsf{N}_1\ man)) \\ (\mathsf{VP}_2\ (\mathsf{V}_1\ loves)\ (\mathsf{NP}_2\ (\mathsf{Det}_1\ some)\ (\mathsf{N}_1\ woman))) \end{aligned} \tag{23}$$

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

$$\begin{aligned} c_{??} &\coloneqq_{sem.} \forall x.(\mathbf{human}\ x) \Rightarrow (\exists y.(\mathbf{human}\ y) \wedge (\mathbf{love}\ x\ y)) \\ c_{??} &\coloneqq_{sem.} \forall x.(\mathbf{man}\ x) \Rightarrow (\exists y.(\mathbf{woman}\ y) \wedge (\mathbf{love}\ x\ y)) \end{aligned} \tag{24}$$

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}	$: \mathsf{N} \multimap \mathsf{NP}$	$\lambda^{o} n.\mathsf{NP}_2\ (\mathsf{Det}_1\ every)\ n$

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

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

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

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 tree encodes, when necessary, the inversion of the scope of the elements. This is reminiscent from 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).

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, multiple adjunction on the same node is forbidden. So the analysis of (**??**) requires first the adjunction of β_{big} into β_{black}, and the adjunction of the result into α_{dog}. Figure **??** shows the resulting derivation tree. On the other hand, the independent adjunction shown in Fig. **??** only specifies that both adjectives adjoin at the N node of the initial tree, specifying both the derived tree of Fig. **??** and the derived tree of Fig. **??**.

(25) big black dog

(26) black big dog

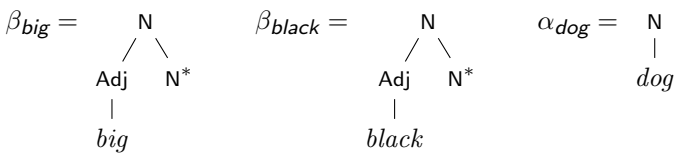


Figure 23: TAG elementary trees for adjectives



(a) Derived tree for (??)

(b) Derived tree for (??)

Figure 24: Alternative notions of derived trees



(a) Dependent derivation tree

(b) Independent derivation tree

Figure 25: Alternative notions of derivation trees

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 it provides.[18] The main concern again has to do with the relation between derivation trees and semantic dependencies. The dependent derivation of Fig. **??** reflects "cascaded modifications" of the head, rather than more expected "separated modifications". We can however achieve this latter effect by specifying a semantic interpretation for adjectives that encodes such a behavior.

We consider the extension of $\Sigma_{derivations}$ with the constants of Table **??** and their interpretations by $\mathscr{G}_{derived\ trees}$ and $\mathscr{G}_{sem.}$ of Table **??**. 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. As the adjunction of an adjective does not modify it, the modification it induces is the same as the one an adjunction at the root node of the auxiliary tree of the adjective. 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 $(\mathsf{NP}[-] \multimap \mathsf{NP}[-])$ or $(\mathsf{NP}[+] \multimap \mathsf{NP}[+])$ (at least 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 $(\mathsf{NP}[-] \multimap \mathsf{NP}[+]) \multimap (\mathsf{NP}[-] \multimap \mathsf{NP}[+]) = (\mathsf{N} \multimap \mathsf{NP}) \multimap \mathsf{N} \multimap \mathsf{NP}$.

(**??**) shows the interpretations of the term $c_{??} : (\mathsf{N} \multimap \mathsf{NP}) \multimap \mathsf{NP}$ (an expression missing a determiner of type $(\mathsf{N} \multimap \mathsf{NP})$ to provide a NP) that encodes the derivation tree of Fig. **??**. The

[18] The two other main arguments relate to the addition of adjoining constraints and to the addition of statistical parameters.

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

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

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

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

interpretation by $\mathscr{G}_{\mathit{sem.}}$ indeed provides a separated modification of the same variable x.

$$\begin{array}{ll} c_{??} & = \lambda^{\circ} D.c_{\mathit{dog}}\ (c_{\mathit{black}}\ (c_{\mathit{big}}\ D)) \\ & :=_{\mathit{derived\ trees}} \lambda^{\circ} D.D\ (\mathsf{N}_2\ (\mathsf{Adj}_1\ \mathit{big})\ (\mathsf{N}_2\ (\mathsf{Adj}_1\ \mathit{black})\ (\mathsf{N}_1\ \mathit{dog}))) \\ & :=_{\mathit{sem.}} \lambda^{\circ} D.\lambda^{\circ} q.D\ (\lambda x.((\mathbf{big}\ x) \wedge (\mathbf{black}\ x)) \wedge (\mathbf{dog}\ x))\ q \end{array} \quad (27)$$

Remark 8. By not introducing a constant $I_{\mathsf{N}} : \mathsf{N} \multimap \mathsf{NP}$ in $\Sigma_{\mathit{derivations}}$, 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 $\Sigma_{\mathit{derivations}}$, 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 parameter. 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 **??** (p. **??**) 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 $\mathscr{G}_{\mathit{derived\ trees}} = \langle \Sigma_{\mathit{derivations}}, \Sigma_{\mathit{trees}}, \mathscr{L}_{\mathit{derived\ trees}}, \mathsf{S} \rangle$, we also define a higher-order signature $\Sigma_{TAG} = \langle A_{TAG}, C_{TAG}, \tau_{TAG} \rangle$ such that $A_{TAG} = A_{\mathit{derivations}} \cup \bigcup_{X \in A_{\mathit{derivations}}} X_A$.

For any $\delta_{\mathit{anchor}} \in \mathcal{I} \cup \mathcal{A}$ an elementary tree of $\mathcal{G}$, 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 $\delta_{\mathit{anchor}} \in \mathcal{I}$ is an initial tree, and $\alpha = Z_A$ with Z the label of the foot node if $\delta_{\mathit{anchor}} \in \mathcal{A}$ is an auxiliary tree.

We get for instance the constants typed as (**??**) shows from the elementary trees of Fig. 1.

$$\begin{aligned} C_{\mathit{sleeps}} &: \mathsf{S}_A \multimap \mathsf{VP}_A \multimap \mathsf{NP} \multimap \mathsf{S} \\ C_{\mathit{John}} &: \mathsf{NP} \\ C_{\mathit{seemingly}} &: \mathsf{VP}_A \multimap \mathsf{VP}_A \end{aligned} \tag{28}$$

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 $\Sigma_{\mathit{derivations}}$: 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} with

$$C_{TAG} = (\cup_{\delta_{\mathit{anchor}} \in \mathcal{I} \cup \mathcal{A}} C_{\mathit{anchor}}) \cup (\cup_{X \in V^0} I_X)$$

and τ_{TAG} is the associated typing function defined 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 **??**, p. **??**).

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

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

We now can relate Σ_{TAG} and $\Lambda(\Sigma_{\mathit{derivations}})$ through a new ACG $\mathscr{G}_{TAG} = \langle \Sigma_{TAG}, \Sigma_{\mathit{derivations}}, \mathscr{L}_{TAG}, \mathsf{S} \rangle$ 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 $\alpha = X \in A_{\mathit{derivations}}$ and $\mathscr{L}_{TAG}(\alpha) = \mathscr{L}_{TAG}(X) = X$;

- for all $C_{\mathit{anchor}} \in C_{TAG}$, $\mathscr{L}_{TAG}(C_{\mathit{anchor}}) = c_{\mathit{anchor}}$.

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

Table **??** 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 $\Lambda(\Sigma_{\mathit{derivations}})$ have no antecedent by $\mathscr{L}_{TAG}$. For instance $c_{\mathit{sleeps}}\ (c_{\mathit{matters}}\ I_{\mathsf{VP}})\ I_{\mathsf{VP}}\ c_{\mathit{John}} : \mathsf{S} \in \mathcal{A}(\mathscr{G}_{\mathit{derived\ trees}})$, where $c_{\mathit{matters}} : (\mathsf{VP} \multimap \mathsf{VP}) \multimap \mathsf{S} \multimap \mathsf{S}$ corresponds to the initial tree $\alpha_{\mathit{matters}}$ of Fig. **??**, 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 $c_{\mathit{matters}}\ I_{\mathsf{VP}} : \mathsf{S} \multimap \mathsf{S}$ in $\mathscr{G}_{\mathit{derived\ trees}}$, while it encodes an initial tree, is the same as the type of a term encoding an adjunction on a S node (see Remark **??** p. **??**). But this is not true anymore at the level of $\mathscr{G}_{TAG}$ where $C_{\mathit{matters}}\ I_{\mathsf{VP}} : \mathsf{S} \multimap \mathsf{S}$ but the type of a term encoding an adjunction on a S node is now S_A.

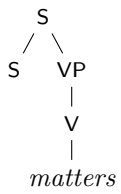


Figure 26: Initial tree for *matters* (the S leaf is a substitution node)

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 Fig. **??** (p. **??**). 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.

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 9. 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 $\mathsf{N}_A \coloneqq_{TAG} \mathsf{N} \multimap \mathsf{NP}$. We can use even more complex interpretations if it helps explaining the semantic interpretation. For instance, in Section 4.2 we introduce a type $\mathsf{S}'_A \coloneqq_{TAG} (\mathsf{NP} \multimap \mathsf{S}) \multimap \mathsf{S}$ to model control verbs.

Since Σ_{TAG} is second-order, we may parse terms with ACGs that have it as abstract vocabulary 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. More examples are available in the example files.

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		$\mathsf{N} \multimap \mathsf{NP}$	
$C_{\textit{John}}$	$: \mathsf{NP}$	$c_{\textit{John}}$	$: \mathsf{NP}$
$C_{\textit{sleeps}}$	$: \mathsf{S}_A \multimap \mathsf{VP}_A \multimap \mathsf{NP} \multimap \mathsf{S}$	$c_{\textit{sleeps}}$	$: (\mathsf{S} \multimap \mathsf{S}) \multimap (\mathsf{VP} \multimap \mathsf{VP}) \multimap \mathsf{NP} \multimap \mathsf{S}$
$C_{\textit{seemingly}}$	$: \mathsf{VP}_A \multimap \mathsf{VP}_A$	$c_{\textit{seemingly}}$	$: (\mathsf{VP} \multimap \mathsf{VP}) \multimap \mathsf{VP} \multimap \mathsf{VP}$
$C_{\textit{usually}}$	$: \mathsf{VP}_A \multimap \mathsf{VP}_A$	$c_{\textit{usually}}$	$: (\mathsf{VP} \multimap \mathsf{VP}) \multimap \mathsf{VP} \multimap \mathsf{VP}$
$C_{\textit{hence}}$	$: \mathsf{S}_A \multimap \mathsf{S}_A$	$c_{\textit{hence}}$	$: (\mathsf{S} \multimap \mathsf{S}) \multimap \mathsf{S} \multimap \mathsf{S}$
I_{S}	$: \mathsf{S}_A$	I_{S}	$: \mathsf{S} \multimap \mathsf{S}$
I_{VP}	$: \mathsf{VP}_A$	I_{VP}	$: \mathsf{VP} \multimap \mathsf{VP}$
$C_{\textit{matters}}$	$: \mathsf{VP}_A \multimap \mathsf{S} \multimap \mathsf{S}$	$c_{\textit{matters}}$	$: (\mathsf{VP} \multimap \mathsf{VP}) \multimap \mathsf{S} \multimap \mathsf{S}$
$C_{\textit{that sleeps}}$	$: \mathsf{S}_A \multimap \mathsf{VP}_A \multimap \mathsf{NP} \multimap \mathsf{S}$	$c_{\textit{that sleeps}}$	$: (\mathsf{S} \multimap \mathsf{S}) \multimap (\mathsf{VP} \multimap \mathsf{VP}) \multimap \mathsf{NP} \multimap \mathsf{S}$
$C_{\textit{who}}$	$: \mathsf{NP}$	$c_{\textit{who}}$	$: \mathsf{NP}$
$C_{\textit{loves}}$	$: \mathsf{S}_A \multimap \mathsf{VP}_A \multimap \mathsf{NP} \multimap \mathsf{NP}$	$c_{\textit{loves}}$	$: (\mathsf{S} \multimap \mathsf{S}) \multimap (\mathsf{VP} \multimap \mathsf{VP}) \multimap \mathsf{NP} \multimap \mathsf{NP} \multimap \mathsf{S}$
$C_{\textit{to love}}$	$: \mathsf{S}_A \multimap \mathsf{VP}_A \multimap \mathsf{NP} \multimap \mathsf{NP}$	$c_{\textit{to love}}$	$: (\mathsf{S} \multimap \mathsf{S}) \multimap (\mathsf{VP} \multimap \mathsf{VP}) \multimap \mathsf{NP} \multimap \mathsf{NP} \multimap \mathsf{S}$
$C_{\textit{to love?}}$	$: \mathsf{S}_A \multimap \mathsf{VP}_A \multimap \mathsf{NP} \multimap \mathsf{NP}$	$c_{\textit{to love?}}$	$: (\mathsf{S} \multimap \mathsf{S}) \multimap (\mathsf{VP} \multimap \mathsf{VP}) \multimap \mathsf{NP} \multimap \mathsf{NP} \multimap \mathsf{S}$
$C_{\textit{claims}}$	$: \mathsf{S}_A \multimap \mathsf{VP}_A \multimap \mathsf{NP} \multimap \mathsf{S}_A$	$c_{\textit{claims}}$	$: (\mathsf{S} \multimap \mathsf{S}) \multimap (\mathsf{VP} \multimap \mathsf{VP}) \multimap \mathsf{NP} \multimap (\mathsf{S} \multimap \mathsf{S})$
$C_{\textit{seems}}$	$: \mathsf{VP}_A \multimap \mathsf{VP}_A$	$c_{\textit{seems}}$	$: (\mathsf{VP} \multimap \mathsf{VP}) \multimap (\mathsf{VP} \multimap \mathsf{VP})$
$C_{\textit{does think}}$	$: \mathsf{S}_A \multimap \mathsf{VP}_A \multimap \mathsf{NP} \multimap \mathsf{S}_A$	$c_{\textit{does think}}$	$: (\mathsf{S} \multimap \mathsf{S}) \multimap (\mathsf{VP} \multimap \mathsf{VP}) \multimap \mathsf{NP} \multimap (\mathsf{S} \multimap \mathsf{S})$
$C_{\textit{man}}$	$: \mathsf{N}_A \multimap \mathsf{NP}$	$c_{\textit{man}}$	$: (\mathsf{N} \multimap \mathsf{NP}) \multimap \mathsf{NP}$
$C_{\textit{someone}}$	$: \mathsf{NP}$	$c_{\textit{someone}}$	$: \mathsf{NP}$
$C_{\textit{everyone}}$	$: \mathsf{NP}$	$c_{\textit{everyone}}$	$: \mathsf{NP}$
$C_{\textit{some}}$	$: \mathsf{N}_A$	$c_{\textit{some}}$	$: \mathsf{N} \multimap \mathsf{NP}$
$C_{\textit{every}}$	$: \mathsf{N}_A$	$c_{\textit{every}}$	$: \mathsf{N} \multimap \mathsf{NP}$
$C_{\textit{big}}$	$: \mathsf{N}_A \multimap \mathsf{N}_A$	$c_{\textit{big}}$	$: (\mathsf{N} \multimap \mathsf{NP}) \multimap \mathsf{N} \multimap \mathsf{NP}$
$C_{\textit{black}}$	$: \mathsf{N}_A \multimap \mathsf{N}_A$	$c_{\textit{black}}$	$: (\mathsf{N} \multimap \mathsf{NP}) \multimap \mathsf{N} \multimap \mathsf{NP}$
$C_{\textit{dog}}$	$: \mathsf{N}_A \multimap \mathsf{NP}$	$c_{\textit{dog}}$	$: (\mathsf{N} \multimap \mathsf{NP}) \multimap \mathsf{NP}$

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

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.[19] So selection restriction has to be expressed by introducing as many types as necessary (see Section **??** for determiners and Section **??** for adjectives). This is not to be confused with the way features are used to model the syntax-semantics interface. Instead of this latter approach, we use the interpretation of derivation trees. The two approaches can however be related, as discussed in Section 4.3.

7.1 Obligatory Adjoining Constraints

Section **??** 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 **??**, 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) and Vijay-Shanker and Joshi (1991)).

As noticed in Remark **??**, 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 (6) with an adjunction of the control verb *wants* on the reduced clause *to sleep* as Fig. 14 shows. Figure 13 presents the elementary trees of the control verb and for the infinitive clause. However, the S node of the controlled verb, or of the infinitive clause (for sake of simplicity, we directly represent such a clause as an elementary tree with a PRO node. This is similar to representing infinitive clauses as clauses without subject (Abeillé 2002)), and the foot node of the control verbs both bear an additional control feature (`control` in XTAG (XTAG Research Group 2001), or some semantic index `idx` in (Gardent and Parmentier 2005; Gardent 2008)) that are to be unified. This feature on the foot node of the control verb is *shared* with the control feature of the controlling NP (the subject of *wants* in our example). Figure 14 shows the derived tree and the derivation trees for (6).

(29) John wants to sleep

In our ACG encoding, we model S nodes with a control feature with the functional type $\mathsf{NP} \multimap \mathsf{S}$, expressing that such a clause is missing its subject. Consequently, the type of the constant c_{wants}

[19] There has been some proposal with dependent types (de Groote and Maarek 2007; de Groote, Yoshinaka, and Maarek 2007; Pompigne 2013), but the underlying calculus does not have the expected good properties.

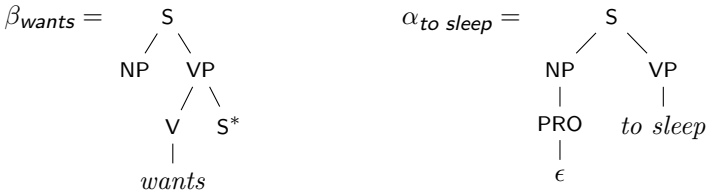
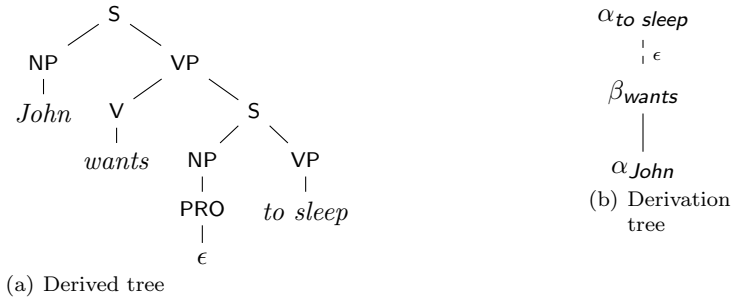


Figure 27: Elementary trees for control verbs



(a) Derived tree

(b) Derivation tree

Figure 28: TAG analysis of control

that models the auxiliary tree β_{wants} is $(\mathsf{S} \multimap \mathsf{S}) \multimap (\mathsf{VP} \multimap \mathsf{VP}) \multimap \mathsf{NP} \multimap (\mathsf{NP} \multimap \mathsf{S}) \multimap \mathsf{S}$. 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 of the type $(\mathsf{NP} \multimap \mathsf{S})$ of the argument and of the type S of the result corresponds to the different feature set attached to the root note 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 6 shows: the first argument x of **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_{14} of $\Lambda(\Sigma_{TAG})$ in (7) represent the derivation tree of Fig. 14, and let c_{14} be its interpretation in $\Lambda(\Sigma_{derivations})$. We can further interpret c_{14} in $\Lambda(\Sigma_{trees})$ (resp. in $\Lambda(\Sigma_{logic})$) in order to have a term representing the associated derived tree (resp. semantics).

$$\begin{aligned}
C_{14} &= C_{to\ sleep}\ (C_{wants}\ I_{\mathsf{S}}\ I_{\mathsf{VP}}\ C_{John}) \\
&:=_{derivations} c_{14} \\
c_{14} &= c_{to\ sleep}\ (c_{wants}\ I_{\mathsf{S}}\ I_{\mathsf{VP}}\ c_{John}) \\
&:=_{derived\ trees} \mathsf{S}_2\ (\mathsf{NP}_1\ John)\ (\mathsf{VP}_2\ (\mathsf{V}_1\ wants) \\
&\qquad\qquad (\mathsf{S}_2\ (\mathsf{NP}_1\ (\mathsf{PRO}_1\ \epsilon))\ (\mathsf{VP}_2\ (\mathsf{V}_1\ to)\ (\mathsf{VP}_1\ sleep)))) \\
&:=_{sem.} \textbf{want john}\ (\textbf{sleep john})
\end{aligned} \tag{30}$$

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.

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

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

$$
\begin{aligned}
c_{\textit{wants}} &:=_{\textit{derived trees}} \lambda^{\circ} \textit{adv}_{\mathsf{S}}\ \textit{adv}_{\mathsf{VP}}\ \textit{subj}\ \textit{pred}.\\
&\qquad \textit{adv}_{\mathsf{S}}\ (\mathsf{S}_2\ \textit{subj}\ (\textit{adv}_{\mathsf{VP}}\ (\mathsf{VP}_2\ (\mathsf{V}_1\ \textit{wants})\ (\textit{pred}\ (\mathsf{PRO}_1\ \epsilon)))))\\
c_{\textit{to sleep}} &:=_{\textit{derived trees}} \lambda^{\circ} \textit{cont}.\textit{cont}(\lambda^{\circ} \textit{subj}.\mathsf{S}_2\ (\mathsf{NP}_1\ \textit{subj})\ (\mathsf{VP}_2\ (\mathsf{V}_1\ \textit{to})\ (\mathsf{VP}_1\ \textit{sleep})))
\end{aligned}
$$

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

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 " ... 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.

$$
\begin{array}{ll}
\textbf{want} & : e \multimap t \multimap t \\
c_{\textit{wants}} & \coloneqq_{sem.} \lambda^{o} adv_{\mathsf{S}}\ adv_{\mathsf{VP}}\ subj\ pred.adv_{\mathsf{S}}\ (subj\ (adv_{\mathsf{VP}} \\
 & \qquad\qquad (\lambda x.\textbf{want}\ x\ (pred\ (\lambda^{o} P.P\ x))))) \\
c_{\textit{to sleep}} & \coloneqq_{sem.} \lambda^{o} cont.cont(\lambda^{o} subj.subj\ (\lambda x.\textbf{sleep}\ x))
\end{array}
$$

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

8 Derivation Trees and Semantic Interpretations

Looking at Fig. **??** (p. **??**), 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 $\Lambda(\Sigma_{trees})$.

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.", although 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.

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 (8). Figure 15 presents the initial trees that allow us to analyze (8) as the literal expression, with a compositional meaning built out of the composition of the initial trees $\alpha_{\textit{John}}$, $\alpha_{\textit{kicked}}$, $\alpha_{\textit{the}}$, and $\alpha_{\textit{bucket}}$. We actually syntactically analyze the idiomatic expression the same way, except that the combination of $\alpha_{\textit{kicked}}$, $\alpha_{\textit{the}}$, and $\alpha_{\textit{bucket}}$ is also considered as the interpretation of the constant $c_{\textit{kicked the bucket}}$. We use $c_{\textit{kicked the bucket}}$ in the idiomatic derivation in Fig. 16(c) to stress that there is no corresponding elementary tree. The ACG abstract term we get really corresponds to this derivation, as the term $C_{16(c)}$ of Equation (9) shows. In both cases, the derived tree is the same (Fig. 16(a)). However, the derivation trees differ, as Fig. 16(b) and Fig. 16(c) show.

(31) John kicked the bucket

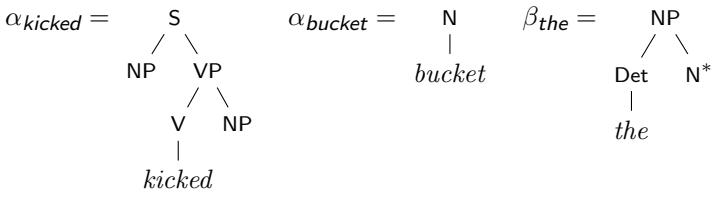
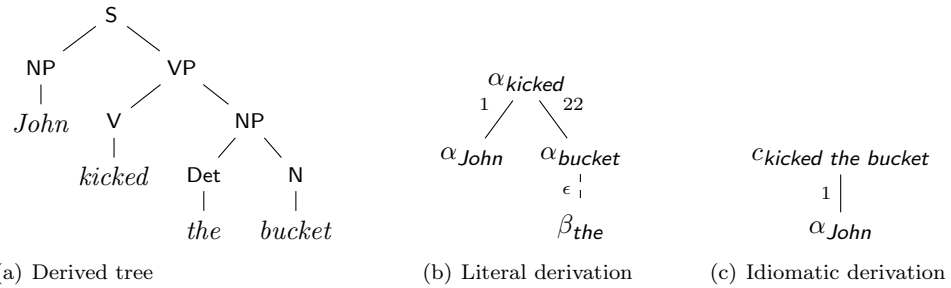


Figure 29: Elementary trees to analyze "kick the bucket"



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

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

Table 8 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 7, 8, and 9, (9), (10), (11), and (12) hold. They show that the two terms $C_{16(b)}$ and $C_{16(c)}$ have the same interpretations as derived tree by $\mathscr{G}_{derived\ trees}$. But they have two different interpretations as logical formulas by the ACG $\mathscr{G}_{sem.}$.

$$\begin{aligned} C_{16(b)} &= C_{kicked}\ I_{\mathsf{S}}\ I_{\mathsf{VP}}\ C_{John}\ (C_{bucket}\ C_{the}) \\ C_{16(c)} &= C_{kicked\ the\ bucket}\ I_{\mathsf{S}}\ I_{\mathsf{VP}}\ C_{John} \end{aligned} \tag{32}$$

$$\begin{aligned} \mathscr{G}_{TAG}(C_{16(b)}) &= c_{kicked}\ I_{\mathsf{S}}\ I_{\mathsf{VP}}\ c_{John}\ (c_{bucket}\ c_{the}) \\ \mathscr{G}_{TAG}(C_{16(c)}) &= c_{kicked\ the\ bucket}\ I_{\mathsf{S}}\ I_{\mathsf{VP}}\ c_{John} \end{aligned} \tag{33}$$

$$\begin{aligned} \mathscr{G}_{derived\ trees} \circ \mathscr{G}_{TAG}(C_{16(b)}) &= \mathsf{S}_2\ (\mathsf{NP}_1\ John)\ (\mathsf{VP}_2\ (\mathsf{V}_1\ kicked) \\ &\qquad (\mathsf{NP}_2\ (\mathsf{Det}_1\ the)\ (\mathsf{N}_1\ bucket))) \\ &= \mathscr{G}_{derived\ trees} \circ \mathscr{G}_{TAG}(C_{16(c)}) \end{aligned} \tag{34}$$

$$\begin{aligned} \mathscr{G}_{sem.}(C_{16(b)}) &= \exists!\, x.(\mathbf{bucket}\ x) \wedge (\mathbf{kick}\ \mathbf{john}\ x) \\ \mathscr{G}_{sem.}(C_{16(c)}) &= \mathbf{die}\ \mathbf{john} \end{aligned} \tag{35}$$

8.2 Subordinate Conjunctions

We saw in Section 4.2 that infinitive clauses behave like clauses missing a subject. In this case, the matrix clause (control verbs) adjoins on the infinitive clause. As the latter is argument of the

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

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

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

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

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 (13), it is the subordinate clause that adjoins on the matrix clause and uses it as argument, as Fig. 17 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 of the subordinate clause.

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

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.

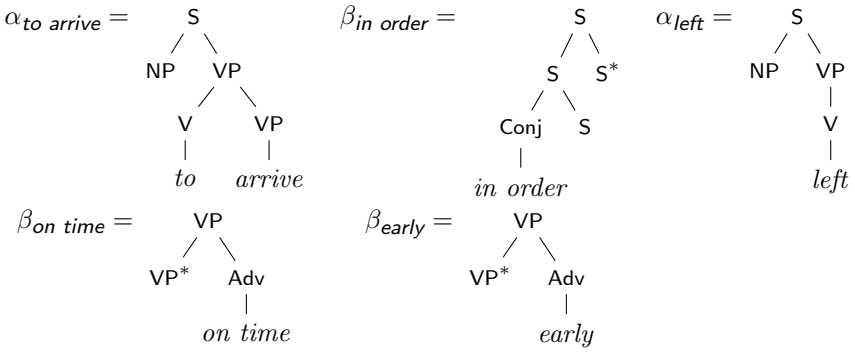
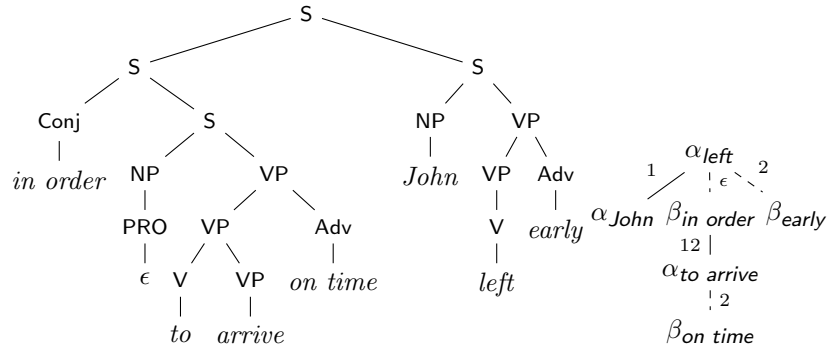


Figure 31: Auxiliary trees for subordinate conjuncts



(a) Derived tree (b) Derivation tree

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

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

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

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

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

In order to implement the solution, terms for verbs such as c_{left} in Table 10 have an additional parameter 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 subordinate conjunction such as $c_{\mathit{in\ order}}$. We can consider this additional parameter 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 $\mathscr{G}_{\mathit{derived\ trees}}(c_{\mathit{left}}) = \gamma_{\mathit{left}}$ how the subordinate clause is inserted. The latter correspond to the *sub* parameter in γ_{left} in Table 12. It takes as argument the whole S rooted 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 $\gamma_{\mathit{in\ order}}$: the *sub* argument corresponds to the infinitive subordinate clause to be substituted in $\beta_{\mathit{in\ order}}$, 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 $\Sigma_{\mathit{derivations}}$ are interpretations of atomic types

$$
\begin{array}{ll}
\mathsf{S}''_A & :=_{TAG} (\mathsf{NP} \multimap \mathsf{S}) \multimap (\mathsf{NP} \multimap \mathsf{S}) \\
\mathsf{S}_{\mathsf{ws}} & :=_{TAG} \mathsf{NP} \multimap \mathsf{S} \\
C_{\mathit{left}} & : \mathsf{S}_A \multimap \mathsf{S}''_A \multimap \mathsf{VP}_A \multimap \mathsf{NP} \multimap \mathsf{S} \\
 & :=_{TAG} c_{\mathit{left}} \\
C_{\mathit{to\ arrive}} & : \mathsf{VP}_A \multimap \mathsf{S}_{\mathsf{ws}} \\
 & :=_{TAG} c_{\mathit{to\ arrive}} \\
C_{\mathit{in\ order}} & : \mathsf{S}_{\mathsf{ws}} \multimap \mathsf{S}''_A \\
 & :=_{TAG} c_{\mathit{in\ order}} \\
I_{\mathsf{S}''} & : \mathsf{S}''_A \\
 & :=_{TAG} I_{\mathsf{NP} \multimap \mathsf{S}} \\
C_{\mathit{early}} & : \mathsf{VP}_A \\
 & :=_{TAG} c_{\mathit{early}} \\
C_{\mathit{on\ time}} & : \mathsf{VP}_A \\
 & :=_{TAG} c_{\mathit{on\ time}}
\end{array}
$$

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

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

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

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 11 shows.

With the lexicon of Tables 10, 11, and 12, we can build terms that correspond to the derivation and derived trees of Fig. 18 as (14), (15), (16) show.[20] We compute the semantic interpretation as in (17).

$$C_{18(b)} = C_{\mathit{left}}\ I_{\mathsf{S}}\ (C_{\mathit{in\ order}}\ (C_{\mathit{to\ arrive}}\ C_{\mathit{on\ time}}))\ C_{\mathit{early}}\ C_{\mathit{John}} \tag{37}$$

$$\mathscr{G}_{TAG}(C_{18(b)}) = c_{\mathit{left}}\ I_{\mathsf{S}}\ (c_{\mathit{in\ order}}\ (c_{\mathit{to\ arrive}}\ c_{\mathit{on\ time}}))\ c_{\mathit{early}}\ c_{\mathit{John}} \tag{38}$$

$$
\begin{aligned}
\mathscr{G}_{\mathit{derived\ trees}} \circ \mathscr{G}_{TAG}(C_{18(b)}) = \\
&\mathsf{S}_2\ (\mathsf{S}_2\ (\mathsf{Conj}_1\ \mathit{in\ order})\ (\mathsf{S}_2\ (\mathsf{NP}_1\ (\mathsf{PRO}_1\ \epsilon))(\mathsf{VP}_2 \\
&\quad (\mathsf{VP}_2\ (\mathsf{V}_1 \mathit{to})\ (\mathsf{VP}_1\ \mathit{arrive}))(\mathsf{Adv}_1\ \mathit{on\ time})))) \\
&\qquad (\mathsf{S}_2\ (\mathsf{NP}_1\ \mathit{John})\ (\mathsf{VP}_2\ (\mathsf{VP}_1\ (\mathsf{V}_1\ \mathit{left}))\ (\mathsf{Adv}_1\ \mathit{early})))
\end{aligned} \tag{39}
$$

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

$$\mathscr{G}_{sem.} \circ \mathscr{G}_{TAG}(C_{18(b)}) = \textbf{goal}\ (\textbf{on_time}\ (\textbf{arrive john}))\ (\textbf{early}\ (\textbf{leave john})) \tag{40}$$

Because $\mathscr{G}_{TAG}$ still is second-order, parsing is available. Parsing the logical term t_{20}^{logic} (see (18)) results in the term $t_{20} : \mathsf{S}$ of $\mathcal{A}(\mathscr{G}_{sem.} \circ \mathscr{G}_{TAG})$. This is the same term of $\mathcal{A}(\mathscr{G}_{yield} \circ \mathscr{G}_{derived\ trees} \circ \mathscr{G}_{TAG})$ that we get when parsing t_{20}^{string}.

$$t_{20}^{\text{logic}} = \exists x.(\textbf{man}\ x) \wedge (\textbf{goal}(\textbf{on_time}(\textbf{arrive}\ x))(\textbf{early}(\textbf{leave}\ x))) \tag{41}$$

$$t_{20}^{\text{string}} = in + order + to + arrive + on + time + a + man + left + early \tag{42}$$

$$t_{20} = C_{\mathit{left}}\ I_{\mathsf{S}}(C_{\mathit{in\ order}}\ (C_{\mathit{to\ arrive}}\ C_{\mathit{on\ time}}))\ C_{\mathit{early}}\ (C_{\mathit{man}}\ C_{\mathit{a}}) \tag{43}$$

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 Fig. **??** (p. **??**), the relation between TAG derivations as terms of $\Lambda(\Sigma_{TAG})$ and terms of $\Lambda(\Sigma_{logic})$ was functional (encoded by the function $\mathscr{G}_{sem.} \circ \mathscr{G}_{TAG}$). The non-functional relation was 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 were two derivation trees for *John kicked the bucket*, there were two possible semantic interpretations. But with only one derivation tree for *every man loves some woman*, there was 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*. We do so still using homomorphisms. We introduce an abstract vocabulary Σ_{CoTAG} and two ACGs. The first one, $\mathscr{G}_{CoTAG}$, maps terms of $\Lambda(\Sigma_{CoTAG})$ to terms of $\Lambda(\Sigma_{TAG})$, i.e., TAG derivation trees. The second one, $\mathscr{G}_{co\text{-}sem.}$, 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 $\mathscr{G}_{co\text{-}sem.} \circ \mathscr{G}^{-1}_{CoTAG}$ as Fig. 19 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.

The idea is to use the type-raising methods of categorial and type-logical grammars. So corresponding to a term $C_{\mathit{everyone}} : \mathsf{NP}$ in Σ_{TAG}, we have a term $L_{\mathit{everyone}} : (\mathsf{NP} \multimap \mathsf{S}) \multimap \mathsf{S}$ in Σ_{CoTAG} such that $\mathscr{G}_{CoTAG}(L_{\mathit{everyone}}) = \lambda^{\circ} P.P\ C_{\mathit{everyone}}$. More generally, whenever a term of type A occurring within a constituent of type B can take scope over this term, we associate to $C_{\mathit{scoping}} : A_1 \multimap \ldots \multimap A_n \multimap A$ in Σ_{TAG} a term $L_{\mathit{scoping}} : A_1 \multimap \ldots \multimap A_n \multimap (A \multimap B) \multimap B$ in Σ_{CoTAG} such that $\mathscr{G}_{CoTAG}(L_{\mathit{scoping}}) = \lambda^{\circ} x_1 \ldots x_n . \lambda^{\circ} P.P\ (C_{\mathit{scoping}}\ x_1 \cdots x_n)$. For other lexical items $C_{\mathit{lex.\ item}} : \alpha$, we have $L_{\mathit{lex.item}} : \alpha$ such that $\mathscr{G}_{CoTAG}(L_{\mathit{lex.item}}) = C_{\mathit{lex.\ item}}$. And for any atomic type A, $\mathscr{G}_{CoTAG}(A) = A$.

Table 13 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 $L^{sws}_{??}$ and $L^{ows}_{??}$ as defined in (21) and (22) respectively, we indeed have $\mathscr{G}_{CoTAG}(L^{sws}_{??}) = \mathscr{G}_{CoTAG}(L^{ows}_{??})$, i.e., two different abstract terms of $\Lambda(\Sigma_{CoTAG})$ that are mapped to the same term of $\Lambda(\Sigma_{TAG})$

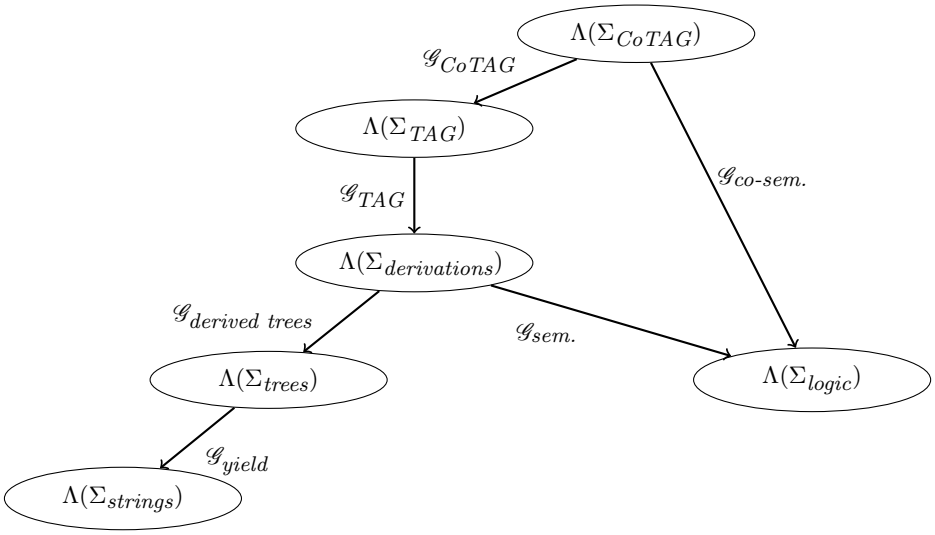


Figure 33: ACG composition for TAG and CoTAG

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

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

(TAG derivation tree).

$$\begin{aligned} L^{sws}_{??} &= (L_{man}\ L_{every}) \\ &\qquad (\lambda^{o}x.(L_{woman}\ L_{some})\ (\lambda^{o}y.L_{loves}\ I_{\mathsf{S}}\ I_{\mathsf{VP}}\ x\ y)) \\ \mathscr{G}_{CoTAG}(L^{sws}_{??}) &= (\lambda^{o}P.P\ (C_{man}\ C_{every}))\ (\lambda^{o}x.(\lambda^{o}P.P\ (C_{woman}\ C_{some}))\ (\lambda^{o}y.C_{loves}\ I_{\mathsf{S}}\ I_{\mathsf{VP}}\ x\ y)) \\ &= C_{loves}\ I_{\mathsf{S}}\ I_{\mathsf{VP}}\ (C_{man}\ C_{every})\ (C_{woman}\ C_{some}) \end{aligned} \tag{44}$$

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

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 $\Sigma_{derivations}$, because the two terms $L^{sws}_{??}$ and $L^{ows}_{??}$ are interpreted as a single term in

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

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

$\Lambda(\Sigma_{TAG})$, we still would get a single interpretation. In other words, we do not want the diagram of Fig. 19 to commute.

Table 14 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.

We can now compute the semantic interpretation of $L_{??}^{sws}$ and $L_{??}^{ows}$ by $\mathscr{G}_{co\text{-}sem.}$. (23) shows 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.

$$\mathscr{G}_{co\text{-}sem.}(L_{??}^{sws}) = \forall x.(\mathbf{man}\ x) \Rightarrow (\exists x'.(\mathbf{woman}\ x') \wedge (\mathbf{love}\ x\ x')) \tag{46}$$

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

This approach to scope ambiguity, first proposed in (Pogodalla 2007b; Pogodalla 2007a) 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 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 $\mathcal{A}(\mathscr{G}_{derived\ trees})$) 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 formula. 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 morphism 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, or even simply with graphs. 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,[21] 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 (Blom et al. 2012) or dynamic semantics (de Groote 2006; Martin and Pollard 2014),[22] and phenomena at the syntax-semantics interface where approaches based on underspecification (Pogodalla 2004a; Pogodalla 2004b) or based on type theory and higher-order logic (Pogodalla 2007b; Pogodalla 2007a; Kobele and Michaelis 2012) can be expressed.

Because they only use unification[23], the approaches to TAG semantics by unification (Gardent and Kallmeyer 2003; Kallmeyer and Romero 2004; Kallmeyer and Romero 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 parameters with unification and to compute the semantics 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

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

[22] 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.

[23] Actually, it's more matching because one of the feature is always a variable.

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 tree, 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, pretty much how statically typed programming language 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, D. J. 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 operations can be used in order to link a TAG phrase grammar with a TAG discourse grammar without requiring an intermediate processing step (Danlos, Maskharashvili, and Pogodalla 2015; Danlos, Maskharashvili, and Pogodalla 2016), contrary to D-LTAG (Webber and Joshi 1998; Forbes et al. 2003; Webber 2004; Forbes-Riley, Webber, and Joshi 2006) or D-STAG (Danlos 2009; Danlos 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 showed how the ACG framework provides a model of the syntax-semantics interface for TAG. To this end, we used the different ways to compose ACGs, resulting in grammars that can be used, with a polynomial complexity, both for parsing and generation. We mainly focused on TAG modelings, but gave examples how this framework naturally allows one to consider modelings that are not part of TAG. A first theoretical perspective consists in studying TAG related extensions in the ACG framework. We may first think of variants related to Multi-Component TAG such as delayed MCTAG (Chiang and Scheffler 2008),[24] or other variants that were designed with a specific purpose (e.g., coordination). Expressing these formalisms in (second-order) ACGs would allow us to directly use their parsing algorithms.

A second more practical perspective consists in improving the ACG parser. This includes implementing optimization techniques, but also experimenting with already available large scale TAG grammars. We will in particular have to address the ambiguity issues and add data-driven techniques, both for parsing (in the usual sense) and for generation.

References

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

Abeillé, Anne (1995). "The Flexibility of French Idioms: A Representation with Lexicalised Tree Adjoining Grammar". In: *Idioms. Structural and Psychological Perspectives*. Ed. by Martin Everaert et al. Psychology Press, Taylor & Francis Group. Chap. 1, pp. 15–42.

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

[24] Expressing MCTAG within ACGs has already been studied in (Pogodalla 2009).

Abeillé, Anne 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)*. Manchester, England: Association for Computational Linguistics, pp. 1–9. ACL anthology: E89-1001.

Bangalore, Srinivas, Robert Frank, and Maribel Romero, eds. (n.d.). *Proceedings of the 10th International Workshop on Tree Adjoining Grammar and Related Frameworks (TAG+10)*. New Haven, CT, US: Linguistics Department, Yale University. ACL anthology: W10-44.

Barendregt, Hendrik Pieter (1984). *The lambda calculus. Its syntax and semantics*. Vol. 103. Studies in Logic and the Foundations of Mathematics. North-Holland.

Barker, Chris (2010). "Cosubstitution, derivational locality, and quantifier scope". In: *Proceedings of the Tenth International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+10)*. Ed. by Srinivas Bangalore, Robert Frank, and Maribel Romero. New Haven, CT, US: Linguistics Department, Yale University, pp. 135–142. ACL anthology: W10-44. URL: http://aclweb.org/anthologyW10-4417.

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

Blom, Chris et al. (2012). "Implicit Arguments: Event Modification or Option Type Categories?" In: *Logic, Language and Meaning*. Ed. by Maria Aloni et al. Vol. 7218. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 240–250. DOI: 10.1007/978-3-642-31482-7_25.

Bos, Johan (1995). "Predicate Logic Unplugged". In: *Proceedings of the Tenth Amsterdam Colloquium*. URL: http://www.let.rug.nl/bos/pubs/Bos1996AmCo.pdf.

Bourreau, Pierre (2012). "Jeux de Typage et Analyse de λ-Grammaires Non-Contextuelles". PhD thesis. Université Bordeaux I. HAL open archive: tel-00733964.

Bourreau, Pierre and Sylvain Salvati (2011). "A Datalog Recognizer for Almost Affine λ-CFGs". In: *The Mathematics of Language*. Ed. by Makoto Kanazawa et al. Vol. 6878. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 21–38. DOI: 10.1007/978-3-642-23211-4_2.

Candito, Marie-Hélène 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)*. Vol. 98-12. IRCS Technical Report Series. University of Pennsylvania. ACL anthology: W98-0106.

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

Chiang, David and Tatjana Scheffler (2008). "Flexible Composition and Delayed Tree-Locality". In: *Proceedings of the Ninth International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+9)*. Ed. by Claire Gardent and Anoop Sarkar. University of Tübingen. Tübingen, Germany. ACL anthology: W08-2303.

Comon, Hubert et al. (2007). *Tree Automata Techniques and Applications*. Available on: http://www.grappa.univ-lille3.fr/tata. release October, 12th 2007.

Culicover, Peter W. and Ray Jackendoff (2005). *Simpler Syntax*. Oxford University Press.

Curry, Haskell Brooks (1961). "Some Logical Aspects of Grammatical Structure". In: *Structure of Language and its Mathematical Aspects: Proceedings of the Twelfth Symposium in Applied Mathematics*. Ed. by Roman Jakobson. American Mathematical Society, pp. 56–68.

Danlos, Laurence (2009). "D-STAG : un formalisme d'analyse automatique de discours basé sur les TAG synchrones". In: *T.A.L.* 50.1, pp. 111–143. HAL open archive: inria-00524743.

Danlos, Laurence (2011). "D-STAG: a Formalism for Discourse Analysis based on SDRT and using Synchronous TAG". In: *14th conference on Formal Grammar - FG 2009*. Ed. by Philippe de Groote, Markus Egg, and Laura Kallmeyer. Vol. 5591. LNCS/LNAI. Springer, pp. 64–84. DOI: 10.1007/978-3-642-20169-1_5.

Danlos, Laurence, 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. Caen, France: Association pour le Traitement Automatique des Langues, pp. 158–169. HAL open archive: hal-01145994.

Danlos, Laurence, 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.

de Groote, Philippe (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.

de Groote, Philippe (2002). "Tree-Adjoining Grammars as Abstract Categorial Grammars". In: *Proceedings of the Sixth International Workshop on Tree Adjoining Grammars and Related Frameworks (TAG+6)*. Ed. by Robert Frank. Università di Venezia. Università di Venezia, pp. 145–150. ACL anthology: W02-2220.

de Groote, Philippe (2006). "Towards a Montagovian account of dynamics". In: *Proceedings of Semantics and Linguistic Theory (SALT) 16*. Ed. by Masayuki Gibson and Jonathan Howell. DOI: 10.3765/salt.v16i0.2952.

de Groote, Philippe 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. URL: http://let.uvt.nl/general/people/rmuskens/ndttg/ndttg2007.pdf.

de Groote, Philippe, Ryo Yoshinaka, and Sarah Maarek (2007). "On Two Extensions of Abstract Categorial Grammars". In: *Logic for Programming, Artificial Intelligence, and Reasoning, 14th International Conference, LPAR 2007, Yerevan, Armenia, October 15-19, 2007, Proceedings*. Ed. by Nachum Dershowitz and Andrei Voronkov. Vol. 4790. Lecture Notes in Computer Science. Springer, pp. 273–287. DOI: 10.1007/978-3-540-75560-9_21.

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

de Groote, Philippe and Sylvain Pogodalla (2004). "On the expressive power of Abstract Categorial Grammars: Representing context-free formalisms". In: *Journal of Logic, Language and Information* 13.4. http://www.springerlink.com/content/1572-9583/, pp. 421–438. DOI: 10.1007/s10849-004-2114-x. HAL open archive: inria-00112956.

Dymetman, Marc (1994). "Inherently Reversible Grammars". In: *Reversible Grammars in Natural Language Processing*. Ed. by Tomek Strzalkowski. Kluwer Academic Publishers. Chap. 2, pp. 33–57.

Egg, Markus, Alexander Koller, and Joachim Niehren (2001). "The Constraint Language for Lambda Structures". In: *Journal of Logic, Language, and Information* 10.4, pp. 457–485. DOI: 10.1023/A:1017964622902.

Forbes, Katherine et al. (2003). "D-LTAG System: Discourse Parsing with a Lexicalized Tree-Adjoining Grammar". In: *Journal of Logic, Language and Information* 12.3. Special Issue: Discourse and Information Structure, pp. 261–279. DOI: 10.1023/A:1024137719751.

Forbes-Riley, Katherine, Bonnie Lynn Webber, and Aravind K. Joshi (2006). "Computing Discourse Semantics: The Predicate-Argument Semantics of Discourse Connectives in D-LTAG". In: *Journal of Semantics* 23.1, pp. 55–106. DOI: 10.1093/jos/ffh032.

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

Gardent, Claire (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)*. Manchester, UK, pp. 249–256. ACL anthology: C08-1032.

Gardent, Claire 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.

Gardent, Claire and Yannick Parmentier (2005). "Large Scale Semantic Construction for Tree Adjoining Grammars". In: *Logical Aspects of Computational Linguistics*. Ed. by Philippe Blache et al. Vol. 3492. LNCS. Sptringer, pp. 131–146. DOI: 10.1007/11422532_9.

Gardent, Claire and Anoop Sarkar, eds. (n.d.). *Proceedings of the Ninth International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+9)*. University of Tübingen. ACL anthology: W08-23.

Girard, Jean-Yves (1987). "Linear Logic". In: *Theoretical Computer Science* 50.1, pp. 1–102. DOI: 10.1016/0304-3975(87)90045-4.

Han, Chung-hye and Giorgio Satta, eds. (n.d.). *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.

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

Joshi, Aravind K. (1985). "Tree-adjoining grammars: How much context sensitivity is required to provide reasonable structural descriptions?" In: *Natural Language Parsing*. Ed. by David R. Dowty, Lauri Karttunen, and Arnold M. Zwicky. Cambridge University Press, pp. 206–250.

Joshi, Aravind K. (1994). "Preface". In: *Computational Intelligence* 10.4, pp. VII–XV. DOI: 10.1111/j.1467-8640.1994.tb00002.x.

Joshi, Aravind K., Laura Kallmeyer, and Maribel Romero (2003). "Flexible Composition in LTAG: Quantifier Scope and Inverse Linking". In: *Proceedings of the Fifth International Workshop on Computational Semantics IWCS-5*. Ed. by Harry Bunt, Ielka van der Sluis, and Roser Morante.

Joshi, Aravind K., Leon S. Levy, and Masako Takahashi (1975). "Tree Adjunct Grammars". In: *Journal of Computer and System Sciences* 10.1, pp. 136–163. DOI: 10.1016/S0022-0000(75)80019-5.

Joshi, Aravind K. and Yves Schabes (1997). "Tree-adjoining grammars". In: *Handbook of formal languages*. Ed. by Grzegorz Rozenberg and Arto K. Salomaa. Vol. 3. Springer. Chap. 2.

Kahane, Sylvain, 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.

Kallmeyer, Laura (2002). "Using an Enriched TAG Derivation Structure as Basis for Semantics". In: *Proceedings of the Sixth International Workshop on Tree Adjoining Grammar and Related Frameworks (TAG+6)*. Ed. by Robert Frank. Università di Venezia, pp. 127–136. ACL anthology: W02-2218.

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

Kallmeyer, Laura and Aravind K. Joshi (2003). "Factoring Predicate Argument and Scope Semantics: Underspecified Semantics with LTAG". In: *Research on Language and Computation* 1.1-2, pp. 3–58. DOI: 10.1023/A:1024564228892.

Kallmeyer, Laura and Marco Kuhlmann (2012). "A Formal Model for Plausible Dependencies in Lexicalized Tree Adjoining Grammar". In: *Proceedings of the 11th International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+11)*. Ed. by Chung-hye Han and Giorgio Satta. INRIA Paris Rocquencourt & Université Paris Diderot. Paris, France, pp. 108–116. ACL anthology: W12-4613.

Kallmeyer, Laura and Maribel Romero (2004). "LTAG Semantics with Semantic Unification". In: *Seventh International Workshop on Tree Adjoining Grammar and Related Formalisms - TAG+7*. Ed. by Owen Rambow and Matthew Stone. Simon Fraser University. Vancouver, British Columbia, Canada, pp. 155–162. ACL anthology: W04-3321.

Kallmeyer, Laura and Maribel Romero (2008). "Scope and Situation Binding for LTAG". In: *Research on Language and Computation* 6.1, pp. 3–52. DOI: 10.1007/s11168-008-9046-6.

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

Kanazawa, Makoto (2008a). "A prefix-correct Earley recognizer for multiple context-free grammars". In: *Proceedings of the Ninth International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+9)*. Ed. by Claire Gardent and Anoop Sarkar. University of Tübingen. Tübingen, Germany, pp. 49–56. ACL anthology: W08-2307.

Kanazawa, Makoto (2008b). *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.

Kanazawa, Makoto (2009). "Second-Order Abstract Categorial Grammars as Hyperedge Replacement Grammars". In: *Journal of Logic, Language, and Information* 19.2, pp. 137–161. DOI: 10.1007/s10849-009-9109-6.

Kanazawa, Makoto (2017). "Parsing and Generation as Datalog Query Evaluation". In: *IfCoLog Journal of Logics and their Applications* 4.4. Special Issue Dedicated to the Memory of Grigori Mints, pp. 1103–1211. URL: http://www.collegepublications.co.uk/downloads/ifcolog00013.pdf#page=303.

Kanazawa, Makoto and Sylvain Salvati (2007). "Generating Control Languages with Abstract Categorial Grammars". In: *Proceedings of The 12th conference on Formal Grammar FG 2007*. Ed. by Gerald Penn. CSLI Publications. URL: http://research.nii.ac.jp/~kanazawa/publications/control.pdf.

Kasper, Robert et al. (1995). "Compilation of HPSG to TAG". In: *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics*. Cambridge, Massachusetts, USA: Association for Computational Linguistics, pp. 92–99. DOI: 10.3115/981658.981671. ACL anthology: P95-1013.

Kobele, Gregory M. (2012). "Idioms and Extended Transducers". In: *Proceedings of the 11th International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+)*. Ed. by Chung-hye Han and Giorgio Satta. INRIA Paris Rocquencourt & Université Paris Diderot. Paris, France, pp. 153–161. ACL anthology: W12-4618.

Kobele, Gregory M. and Jens Michaelis (2012). "CoTAGs and ACGs". In: *Logical Aspects of Computational Linguistics*. Ed. by Denis Béchet and Alexander Dikovsky. Vol. 7351. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 119–134. DOI: 10.1007/978-3-642-31262-5_8.

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

Koller, Alexander and Marco Kuhlmann (2012). "Decomposing TAG Parsing Algorithms Using Simple Algebraizations". In: *Proceedings of the 11th International Workshop on Tree Adjoining*

Grammars and Related Formalisms (TAG+). Ed. by Chung-hye Han and Giorgio Satta. INRIA Paris Rocquencourt & Université Paris Diderot. Paris, France, pp. 135–143. ACL anthology: W12-4616.

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

Lambek, Joachim (1958). "The Mathematics of Sentence Structure". In: *American Mathematical Monthly* 65.3, pp. 154–170. DOI: 10.2307/2310058.

Martin, Scott and Carl Pollard (2014). "A Dynamic Categorial Grammar". In: *Formal Grammar*. Ed. by Glyn Morrill et al. Vol. 8612. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 138–154. DOI: 10.1007/978-3-662-44121-3_9.

Maskharashvili, Aleksandre 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*. The Asian Federation of Natural Language Processing. Nagoya, Japan, pp. 1257–1263. HAL open archive: hal-00868363. ACL anthology: I13-1179.

Montague, Richard (1973). "The Proper Treatment of Quantification in Ordinary English". In: *Approaches to Natural Language: Proceedings of the 1970 Stanford Workshop on Grammar and Semantics*. Ed. by Jaakko Hintikka, Julius Moravcsik, and Patrick Suppes. Dordrecht, Holland: D. Reidel Publishing Co., pp. 221–242. Reprinted as (Montague 1974).

Montague, Richard (1974). "The Proper Treatment of Quantification in Ordinary English". In: *Formal Semantics: The Essential Readings*. Ed. by Paul Portner and Barbara H. Partee. Blackwell Publishers. Chap. 1.

Nesson, Rebecca Nancy (2009). "Synchronous and Multicomponent Tree-Adjoining Grammars: Complexity, Algorithms, and Applications". PhD thesis. Harvard University. URL: http://rebeccanesson.herokuapp.com/thesis.pdf.

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

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

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

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

Pogodalla, Sylvain (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.

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

Pompigne, Florent (2013). "Logical modelization of language and Abstract Categorial Grammars". PhD thesis. Université de Lorraine. HAL open archive: tel-00921040.

Portner, Paul and Barbara H. Partee, eds. (2002). *Formal Semantics: The Essential Readings*. Blackwell Publishers.

Proceedings of the Fifth International Workshop on Tree Adjoining Grammar and Related Frameworks (TAG+5) (2000). Université Paris 7, Jussieu. Paris, France. ACL anthology: W00-20.

Proceedings of the Fourth International Workshop on Tree Adjoining Grammars and Related Frameworks (TAG+4) (1998). Vol. 98-12. IRCS Report. University of Pennsylvania. ACL anthology: W98-01.

Rambow, Owen and Matthew Stone, eds. (n.d.). *Proceedings of the 7th International Workshop on Tree Adjoining Grammar and Related Formalisms*. Simon Fraser University. Vancouver, British Columbia, Canada. ACL anthology: W04-3300.

Rambow, Owen, K. Vijay-Shanker, and David Weir (1995). "D-Tree Grammars". In: *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics*. Cambridge, Massachusetts, USA: Association for Computational Linguistics, pp. 151–158. DOI: 10.3115/981658.981679. ACL anthology: P95-1021.

Rambow, Owen, K. Vijay-Shanker, and David Weir (2001). "D-Tree Substitution Grammars". In: *Computational Linguistics* 27.1, pp. 87–121. DOI: 10.1162/089120101300346813. ACL anthology: J01-1004.

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

Salvati, Sylvain (2005). "Problèmes de filtrage et problèmes d'analyse pour les grammaires catégorielles abstraites". PhD thesis. Institut National Polytechnique de Lorraine.

Salvati, Sylvain (2006). "Encoding second order string ACG with Deterministic Tree Walking Transducers". In: *Proceedings of The 11th conference on Formal Grammar FG 2006*. Ed. by Shuly Wintner. FG Online Proceedings. Malaga Espagne: CSLI Publications, pp. 143–156. URL: http://cslipublications.stanford.edu/FG/2006/salvati.pdf.

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

Salvati, Sylvain (2010). "On the membership problem for non-linear Abstract Categorial Grammars". In: *Journal of Logic, Language and Information* 19.2, pp. 163–183. DOI: 10.1007/s10849-009-9110-0.

Schabes, Yves and Stuart M. Shieber (1994). "An Alternative Conception of Tree-Adjoining Derivation". In: *Computational Linguistics* 20.1, pp. 91–124. ACL anthology: J94-1004.

Shieber, Stuart M. (1988). "A Uniform Architecture for Parsing and Generation". In: *Proceedings of the 12th International Conference on Computational Linguistics*. Ed. by Dénes Vargha. Vol. 2. Budapest, pp. 614–619. ACL anthology: C88-2128.

Shieber, Stuart M. (1994). "Restricting the Weak-Generative Capacity of Synchronous Tree-Adjoining Grammars". In: *Computational Intelligence* 10.4, pp. 371–385. DOI: 10.1111/j.1467-8640.1994.tb00003.x.

Shieber, Stuart M. (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)*. Trento, Italy, pp. 377–384. ACL anthology: E06-1048.

Shieber, Stuart M. (2014). "Bimorphisms and synchronous grammars". In: *Journal of Language Modelling* 2.1, pp. 51–104. DOI: 10.15398/jlm.v2i1.84.

Shieber, Stuart M. and Yves Schabes (1990). "Synchronous Tree-Adjoining Grammars". In: *Proceedings of the 13th International Conference on Computational Linguistics*. Vol. 3. Helsinki, Finland, pp. 253–258. DOI: 10.3115/991146.991191. ACL anthology: C90-3045.

Shieber, Stuart M., Gertjan van Noord, et al. (1989). "A Semantic-Head-Driven Generation Algorithm for Unification-Based Formalisms". In: *Proceedings of the 27th Annual Meeting of the Association for Computational Linguistics*. Vancouver, British Columbia, Canada: Association for Computational Linguistics, pp. 7–17. DOI: 10.3115/981623.981625. ACL anthology: P89-1002.

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

Steedman, Mark and Jason Baldridge (2011). "Combinatory Categorial Grammar". In: *Non-Transformational Syntax: Formal and Explicit Models of Grammar*. Ed. by Robert Borsley and Kersti Börjars. Wiley-Blackwell. Chap. 5.

van Benthem, Johan (1986). *Essays in Logical Semantics*. Vol. 39. Studies in Linguistics and Philosophy. Springer Netherlands. DOI: 10.1007/978-94-009-4540-1.

Vijay-Shanker, K. (1987). "A Study of Tree Adjoining Grammars". PhD thesis. University of Pennsylvania.

Vijay-Shanker, K. (1992). "Using Descriptions of Trees in a Tree Adjoining Grammar". In: *Computational Linguistics* 18.4, pp. 481–518. ACL anthology: J92-4004.

Vijay-Shanker, K. 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*. Chicago, Illinois, USA: Association for Computational Linguistics, pp. 82–93. DOI: 10.3115/981210.981221. ACL anthology: P85-1011.

Vijay-Shanker, K. and Aravind K. Joshi (1988). "Feature Structures Based Tree Adjoining Grammars". In: *Proceedings of the 12th International Conference on Computational Linguistics (COLING Budapest)*. Ed. by Dénes Vargha. Vol. 2. Budapest, pp. 714–718. ACL anthology: C88-2147.

Vijay-Shanker, K. and Aravind K. Joshi (1991). *Unification-Based Tree Adjoining Grammars*. Tech. rep. MS-CIS-91-25. Paper 762. University of Pennsylvania Department of Computer and Information Science (CIS). URL: http://repository.upenn.edu/cis_reports/762.

Webber, Bonnie Lynn (2004). "D-LTAG: extending lexicalized TAG to discourse". In: *Cognitive Science* 28.5, pp. 751–779. DOI: 10.1207/s15516709cog2805_6.

Webber, Bonnie Lynn and Aravind K. Joshi (1998). "Anchoring a Lexicalized Tree-Adjoining Grammar for Discourse". In: *Proccedings of the ACL/COLING workshop on Discourse Relations and Discourse Markers*. Ed. by Manfred Stede, Leo Wanner, and Eduard Hovy. ACL anthology: W98-0315.

Weir, David J. (1988). "Characterizing Mildly Context-Sensitive Grammar Formalisms". PhD thesis. University of Pennsylvania.

XTAG Research Group, (2001). *A Lexicalized Tree Adjoining Grammar for English*. Tech. rep. IRCS-01-03. IRCS, University of Pennsylvania. URL: ftp://ftp.cis.upenn.edu/pub/xtag/release-2.24.2001/tech-report.pdf.

Yoshinaka, Ryo (2006). "Linearization of Affine Abstract Categorial Grammars". In: *proceedings of Fromal Grammar 2006*.

Yoshinaka, Ryo and Makoto Kanazawa (2005). "The Complexity and Generative Capacity of Lexicalized Abstract Categorial Grammars". In: *Proceedings of Logical Aspects of Computational Linguistics: 5th International Conference, LACL 2005*. Ed. by Philippe Blache et al. Vol. 3492. LNCS. Springer-Verlag GmbH, pp. 330–348. DOI: 10.1007/11422532_22.