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

Sylvain Pogodalla

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

Sylvain Pogodalla*

December 11, 2015

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 modeling 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 modeling requires. (Joshi 1985) proposes to call the class of languages (resp. grammars) necessary for describing natural language 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 modeling lies in the capability of its elementary trees to locally specify (syntactic and semantic) dependencies between parts that can occur arbitrary 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 locally described in an elementary tree can end as a long-distance dependency in the resulting derived tree. The relevant structure to store 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 where 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. Nevertheless, solutions to the problem strictly relying on derivation trees were then 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).

In this article, we elaborate on (Pogodalla 2004a; Pogodalla 2009) in order to propose a semantic construction framework for TAGs. We make explicit the above mentioned mismatch between the structure of the TAG derivation trees and the expected semantic dependencies (Section 5.3) and show how our approach can cope with it.

1.2 TAGs and Abstract Categorial Grammars

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). Rather than a grammatical formalism on their own, they provide 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.

A key feature of ACGs is to provide the user a 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 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.

Second-order ACGs (2nd ACGs) is a particular class of interest because of its polynomial parsing property (Salvati 2007). 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

[1] See for instance (Vijay-Shanker and Joshi 1985; Vijay-Shanker 1987; D. J. Weir 1988).

grammars (Kanazawa 2009). A further refinement on the ACG hierarchy allows for 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 (see Section 3.2).

Moving to ACGs to encode TAGs and to build TAG semantic representations offers several advantages. First, we benefit from parsing algorithms and optimization techniques grounded on well established field such as type-theory and Datalog. Kanazawa (2007) showed how parsing of 2nd ACGs reduces to Datalog querying, offering a general method for getting efficient tabular parsing algorithms (Kanazawa 2011). This also allows for deriving algorithms with specific properties such as prefix-correctness in a general way.[2]

A second advantage is to offer a *inherently reversible* framework (Dymetman 1994) as Kanazawa's Datalog reduction 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). Despite the similarity between the ACG approach and the synchronous approaches to semantic construction (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. Processing the semantic trees of a synchronous grammar is straightforward but the inverse process is of interest for generation but is not covered by synchronous TAG. It actually corresponds to the morphism inversion found in ACG parsing. (Koller and Kuhlmann 2012) nevertheless proposes a first step towards such a mechanism using interpreted regular tree grammars. It is however more oriented towards parsing (in the usual sense). On the other hand the computational properties of the lexicon inversion for ACGs has been studied for different classes of λ-terms.[3] It is worth noting that as far as 2nd order ACGs are concerned, whatever the form of the semantic λ-term, generation 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 allows for removing any kind of semantic monotonicity requirement (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. They can be run for parsing and generation using the ACG toolkit.[4]

Semantic representation with λ-terms, and more generally the ACG type-theoretic settings, 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),[5] and phenomena at the syntax-semantics interface where approaches based on underspecification (Pogodalla 2004a; Pogodalla 2004b) or based on type theory and higher-order (Pogodalla 2007b; Pogodalla 2007a; Kobele and Michaelis 2012) can be expressed.

The latter approaches in particular naturally extend the ACG encoding of TAG to deal with scope ambiguities, recasting Barker's (2010) cosubstitution into the standard type-logical approach

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

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

[4] The toolkit is available at `http://www.loria.fr/equipes/calligramme/acg/#Software`. We also provide files implementing the example grammars at `http://www.loria.fr/~pogodall/publications/acg-examples.tgz`. The script file illustrates the terms we use in this article and refers in comment to the relevant sections, equations, and term names.

[5] 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.

```
signature strings =
    o:type;
    string  = o->o:type;
    infix  + = lambda a b z.a (b z):string -> string -> string;
    John,loves,Mary:string;
end
```

ACG example 1: Declaration example

to quantifier raising. This shows how looking at TAG and its variant as fragments of a larger class of grammars allows one to share analysis between grammatical formalisms, and to relate TAG to its variants. It is then possible to add operations to the substitution and adjunction ones that would otherwise be difficult to express as TAG (or even multiple component TAG, MCTAG) operations. Such an operation 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), 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).

Finally, because they consider abstract and object languages as sets of linear λ-terms, the ACG framework provides different ways to *compose* grammars. When two ACGs share a same abstract language, they offer a transduction way of encoding of relations. This is the basic architecture we use to encode the syntax-semantics interface for TAG. On the other hand, an ACG adds further control on a second ACG when the abstract level of the latter is the object level of the former. We extensively use this composition to present a modular perspective on TAG and its extensions. For instance, we show how the semantic interpretation may be defined on structures that are more general than TAG derivation trees (using the sharing of the abstract level between two ACGs) and then how to discard those that do not correspond to TAG derivation trees (using the control from one ACG on another one). (Kobele 2012) shows how to use the modularity to relate both the lexical and the syntactic compositional nature of idioms.

1.3 Experimenting with ACGs

We have been developing the ACG toolkit in order to provide facilities to write grammars and to parse or realize terms of these grammars.[4] We rely on this software to implement the example grammars of this article and the related commands to show the results of parsing or realization of some terms.

An ACG declaration looks like the ACG example 1. Files containing such declaration are meant to be compiled into an object file. The latter can be loaded by the `acg` program that offers for instance type checking or parsing commands as ACG example 2 shows. In such examples, the lines starting with a # correspond to the command to be run and the other ones to the system output.

Large scale tests of the software is however ongoing work, and quantitative evaluation is beyond the scope of this article.

2 Background

2.1 Adjunction and Substitution

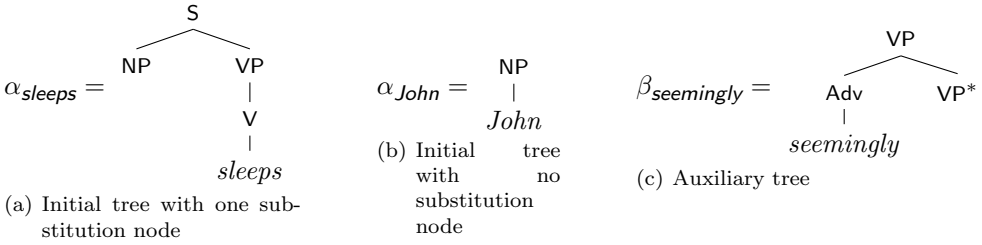
A TAG consists of a finite set of elementary trees whose nodes are labeled by terminal and non-terminal symbols. Nodes labeled with terminals can only be leaves. Figure 1 exemplifies such

```
# load o cl-strings.acgo;
load o cl-strings.acgo;
Loading object file "acg-examples/cl-strings.acgo"...
Done.
# list;
list ;
Available data:
  Signature strings

# strings check John + loves + Mary:string;
strings check John + loves + Mary:string;

In strings:
  John + (loves + Mary) : string
#
```

ACG example 2: Command example

trees. Substituting α_{John} in α_{sleeps} consists in replacing a leaf of α_{sleeps} labeled with a non-terminal symbol NP with the tree α_{John} whose root node is labeled by NP as well.[6] Figure 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 $\beta_{seemingly}$ 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

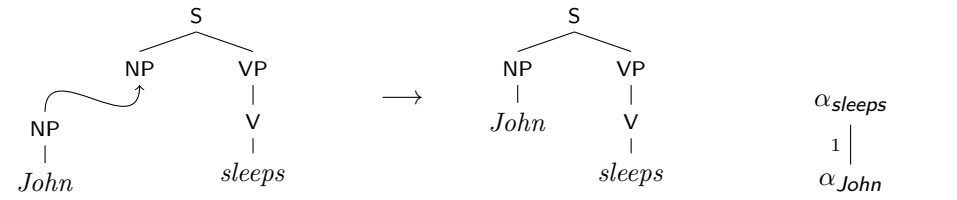
Figure 1: TAG elementary trees

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

2.2 TAG Elementary Trees as Functions

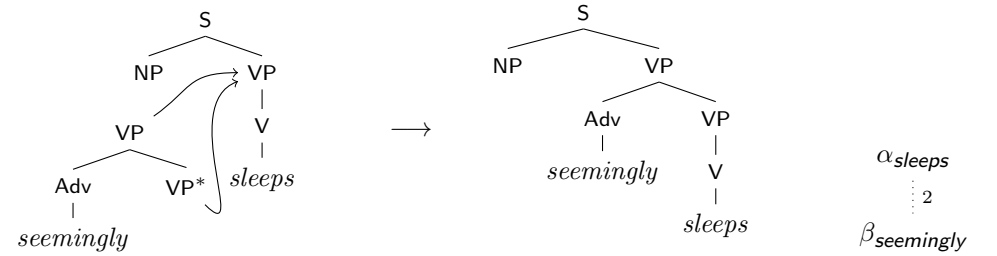
We now present the two operations of adjunction and substitution using a functional interpretation of the elementary trees. We use the standard notations of the typed λ-calculus. As most of the

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



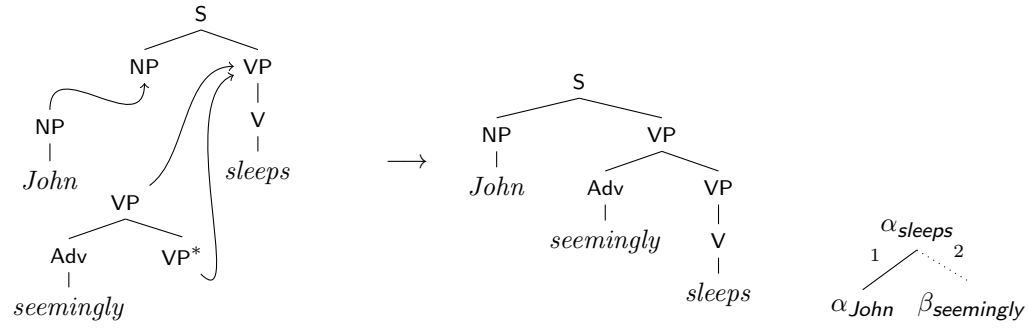
(a) Substitution (b) Derivation tree

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*

examples we present rely on λ-terms and their types, and that running them with the ACG toolkit basically use 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:

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

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 $t, u \in \Lambda(\Sigma)$ and the set of free variables of u and t are disjoint 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 notion of α, β, and η conversions (Barendregt 1984).

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 declaration;
- Δ is a finite set of linear variable typing declaration.

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.

Definition 5 (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 τ of trees and constants of this type (in Section 2.3 we explicit how to systematically encode trees into λ-terms).

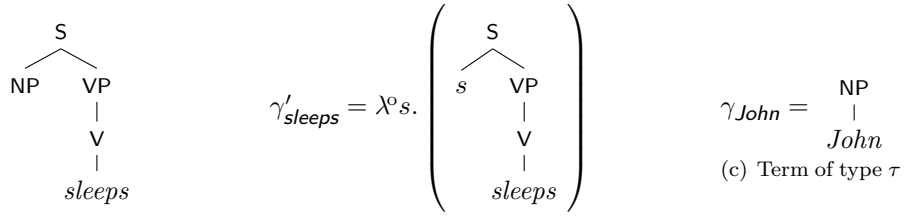
$$\frac{}{\Gamma;\vdash_\Sigma c : \tau(c)}\ (\text{const.})$$

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

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

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

Table 1: Typing rules for deriving typing judgments



(a) TAG initial tree

(b) Function from trees to trees

(c) Term of type τ

Figure 5: Functional interpretation of the substitution operation

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 for considering it as a function that takes a tree as argument and replace the NP node by this argument. Hence we can represent it as the function γ'_{sleeps} shown in Fig. 5(b) with $\gamma'_{sleeps} : \tau \multimap \tau$. A tree where no substitution can occur can be represented as $\gamma_{John} : \tau$ (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.

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

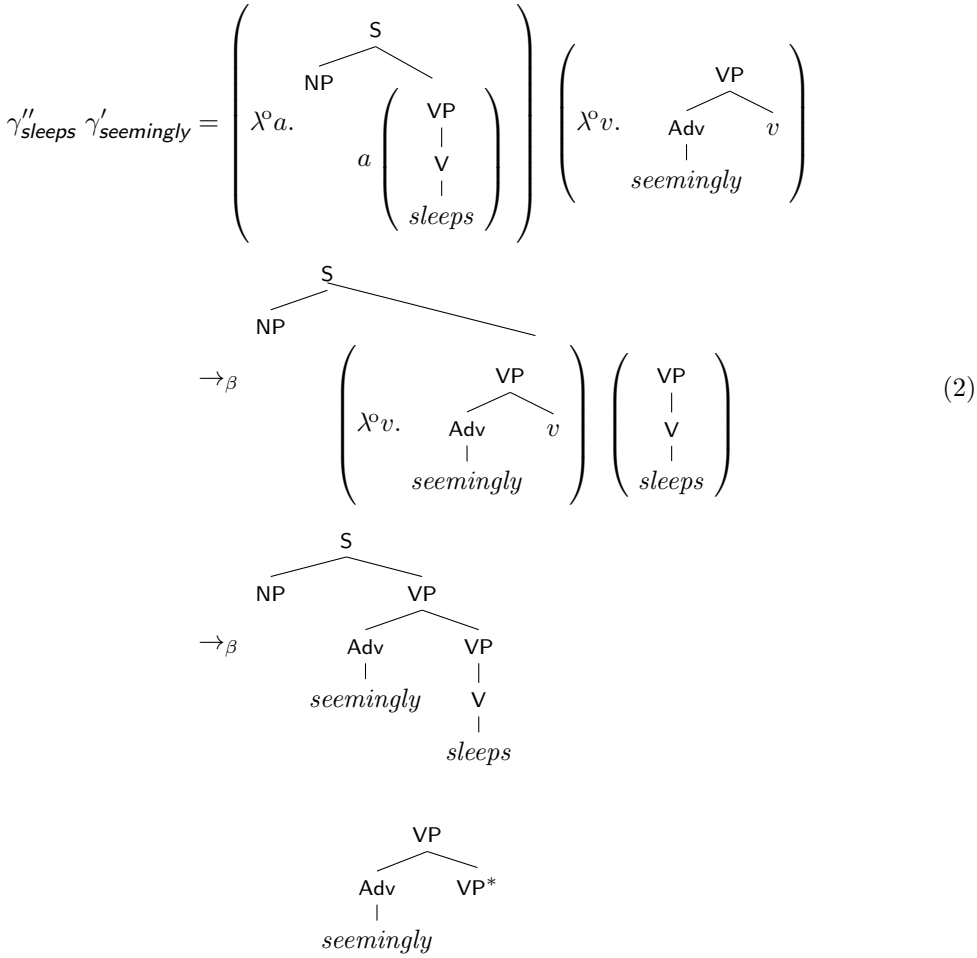
$$\rightarrow_{\beta} \begin{array}{ccc} & \mathsf{S} & \\ \mathsf{NP} & & \mathsf{VP} \\ \vert & & \vert \\ John & & \mathsf{V} \\ & & \vert \\ & & sleeps \end{array}$$

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 $\gamma'_{seemingly} : \tau \multimap \tau$. Then, a tree with an adjunction site can be represented by a term such as $\gamma''_{sleeps} : (\tau \multimap \tau) \multimap \tau$ in Fig. 7(b). Note the higher-order type of γ''_{sleeps}.

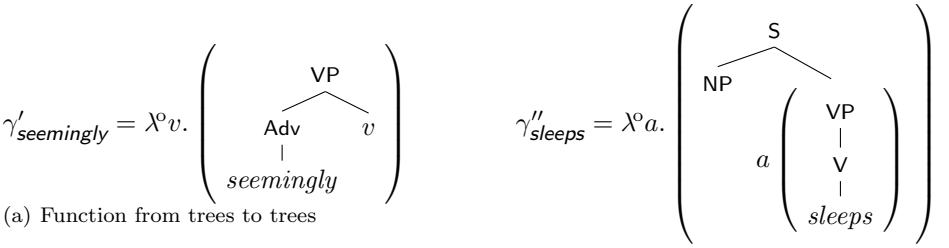
In order to model the adjunction, we then apply γ''_{sleeps} to $\gamma'_{seemingly}$ and perform β-reductions

as (2) shows.



(2)

Figure 6: TAG auxiliary tree



(a) Function from trees to trees

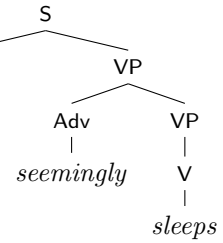
(b) Elementary tree ready to accept an adjunction

Figure 7: Functional interpretation of the adjunction operation

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 $\gamma'''_{\textsf{sleeps}} : (\tau \multimap \tau) \multimap \tau \multimap \tau$.

Such a term can be used to represent the TAG analysis of *John seemingly sleeps* shown in Fig. 4 with the β-reduction of $\gamma'''_{\mathit{sleeps}}\ \gamma'_{\mathit{seemingly}}\ \gamma_{\mathit{John}}$ shown in 3.

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



(3)

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

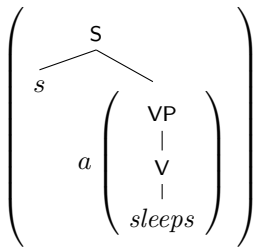
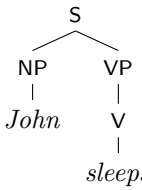


Figure 8: Elementary tree representation available to substitution and adjunction operations

Remark 1 (No adjunction)**.** Typing $\gamma'''_{\mathit{sleeps}}$ with $(\tau \multimap \tau) \multimap \tau \multimap \tau$ makes it require an adjunction (the first $(\tau \multimap \tau)$ parameter) to return a plain tree term of type τ. 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 $\gamma'''_{\mathit{sleeps}}$ to the identity function $I = \lambda^\circ x.x : \tau \multimap \tau$. Then (4) holds.

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



(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 $(\tau \multimap \tau) \multimap (\tau \multimap \tau) \multimap \tau \multimap \tau$ 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.[7]

Remark 2 (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. 5. Accordingly, an auxiliary tree typically also should allow for adjunction at its root. So instead of using $\gamma'_{\mathit{seemingly}} : \tau \multimap \tau$, 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 $\gamma_{\mathit{sleeps}}\ I\ (\gamma_{\mathit{seemingly}}\ (\gamma_{\mathit{usually}}\ I))\ \gamma_{\mathit{John}}$.

[7] 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.

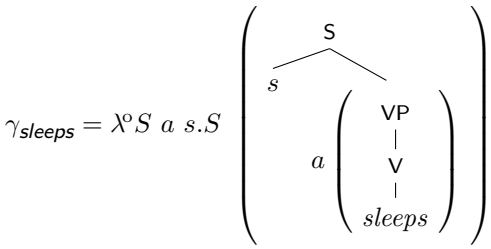


Figure 9: Encoding of α_{sleeps} available to substitution and adjunctions both at the VP and at the S nodes

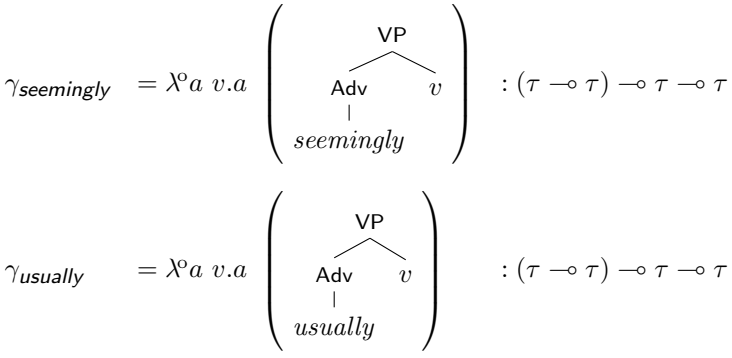


Figure 10: Auxiliary tree representation available to adjunction operations

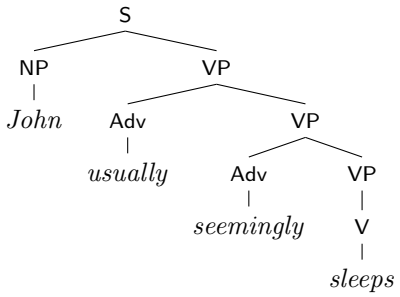


Figure 11: A TAG analysis of *John usually seemingly sleeps*

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 $\sigma \triangleq (o \multimap o)$ the string type. Note it is not an atomic type.

We also define two other terms:

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

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

2.3.2 Encoding Trees

In order to encode the trees defined over a ranked alphabet $\mathcal{F}_a = (\mathcal{F}, arity)$,[8] we use the following higher-order signature $\Sigma^{\mathcal{F}_a}_{trees} = \langle A_\tau, \mathcal{F}, \tau^{\mathcal{F}_a}_\tau \rangle$ where:

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

Trees were primarily defined in TAG using a mapping from positions to labels, elements of a vocabulary. The vocabulary is partitioned into terminal and non-terminal symbols. Hence, a same node label may be used to represent symbols 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 use the notation X_n with n the arity to differentiate between the different usages of an X label. As non-terminal symbols can only be mapped by terminal nodes (leaves), they always have arity 0, so we do not use any subscript for them.

For instance, the TAG elementary trees δ_{anchor}[9] of our running examples can be modeled as the functions (or terms) γ_{anchor} built on the signature Σ_{trees} as Table 2 shows.[10] Then (5) shows

[8] *arity* being a mapping from $\mathcal{F}$ to $\mathbb{N}$.

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

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

that $\gamma_4 = \gamma_{\mathit{sleeps}}\ I\ (\gamma_{\mathit{seemingly}}\ I)\ \gamma_{\mathit{John}}$ corresponds (modulo β) to the tree of Fig. 4.

$$
\begin{aligned}
\gamma_4 &= \gamma_{\mathit{sleeps}}\ I\ (\gamma_{\mathit{seemingly}}\ I)\ \gamma_{\mathit{John}} \\
&= (\lambda^{\circ} S\ a\ s.S\ (\mathsf{S}_2\ s\ (a\ (\mathsf{VP}_1\ (\mathsf{V}_1\ \mathit{sleeps})))))\ I\ (\gamma_{\mathit{seemingly}}\ I)\ \gamma_{\mathit{John}} \\
&\to_\beta (\lambda^{\circ} a\ s.I\ (\mathsf{S}_2\ s\ (a\ (\mathsf{VP}_1\ (\mathsf{V}_1\ \mathit{sleeps})))))\ (\gamma_{\mathit{seemingly}}\ I)\ \gamma_{\mathit{John}} \\
&= (\lambda^{\circ} a\ s.(\lambda^{\circ} x.x)\ (\mathsf{S}_2\ s\ (a\ (\mathsf{VP}_1\ (\mathsf{V}_1\ \mathit{sleeps})))))\ (\gamma_{\mathit{seemingly}}\ I)\ \gamma_{\mathit{John}} \\
&\to_\beta (\lambda^{\circ} a\ s.\mathsf{S}_2\ s\ (a\ (\mathsf{VP}_1\ (\mathsf{V}_1\ \mathit{sleeps}))))\ (\gamma_{\mathit{seemingly}}\ I)\ \gamma_{\mathit{John}} \\
&\to_\beta (\lambda^{\circ} s.\mathsf{S}_2\ s\ ((\gamma_{\mathit{seemingly}}\ I)\ (\mathsf{VP}_1\ (\mathsf{V}_1\ \mathit{sleeps}))))\ \gamma_{\mathit{John}} \\
&\to_\beta \mathsf{S}_2\ \gamma_{\mathit{John}}\ ((\gamma_{\mathit{seemingly}}\ I)\ (\mathsf{VP}_1\ (\mathsf{V}_1\ \mathit{sleeps}))) \\
&\to_\beta \mathsf{S}_2\ \gamma_{\mathit{John}}\ (((\lambda^{\circ} a\ v.a\ (\mathsf{VP}_2\ (\mathsf{Adv}_1\ \mathit{seemingly})\ v))\ I)\ (\mathsf{VP}_1\ (\mathsf{V}_1\ \mathit{sleeps}))) \\
&\to_\beta \mathsf{S}_2\ \gamma_{\mathit{John}}\ ((\lambda^{\circ} v.I\ (\mathsf{VP}_2\ (\mathsf{Adv}_1\ \mathit{seemingly})\ v))\ (\mathsf{VP}_1\ (\mathsf{V}_1\ \mathit{sleeps}))) \\
&\to_\beta \mathsf{S}_2\ \gamma_{\mathit{John}}\ ((\lambda^{\circ} v.(\lambda^{\circ} x.x)\ (\mathsf{VP}_2\ (\mathsf{Adv}_1\ \mathit{seemingly})\ v))\ (\mathsf{VP}_1\ (\mathsf{V}_1\ \mathit{sleeps}))) \\
&\to_\beta \mathsf{S}_2\ \gamma_{\mathit{John}}\ ((\lambda^{\circ} v.\mathsf{VP}_2\ (\mathsf{Adv}_1\ \mathit{seemingly})\ v)\ (\mathsf{VP}_1\ (\mathsf{V}_1\ \mathit{sleeps}))) \\
&\to_\beta \mathsf{S}_2\ \gamma_{\mathit{John}}\ (\mathsf{VP}_2\ (\mathsf{Adv}_1\ \mathit{seemingly})\ (\mathsf{VP}_1\ (\mathsf{V}_1\ \mathit{sleeps}))) \\
&= \mathsf{S}_2\ (\mathsf{NP}_1\ \mathit{John})\ (\mathsf{VP}_2\ (\mathsf{Adv}_1\ \mathit{seemingly})\ (\mathsf{VP}_1\ (\mathsf{V}_1\ \mathit{sleeps})))
\end{aligned}
\tag{5}
$$

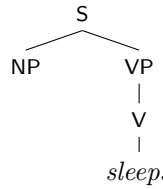
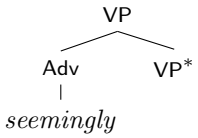
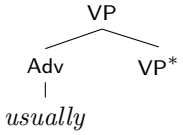
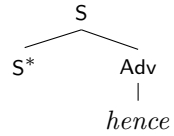
	Terms of $\Lambda(\Sigma_{trees})$	Corresponding TAG tree
γ_{John}	$= \mathsf{NP}_1\ \mathit{John}$ $: \tau$	
γ_{sleeps}	$= \lambda^{\circ} S\ a\ s.S\ (\mathsf{S}_2\ s\ (a\ (\mathsf{VP}_1\ (\mathsf{V}_1\ \mathit{sleeps}))))$ $: (\tau \multimap \tau) \multimap (\tau \multimap \tau) \multimap \tau \multimap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

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

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

3 Abstract Categorial Grammars

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

Definition 6 (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[11];
- $G : C_1 \to \Lambda(\Sigma_2)$. We also note $G : \Lambda(\Sigma_1) \to \Lambda(\Sigma_2)$ its homomorphic extension;
- F and G are such that for all $c \in C_1$, $\vdash_{\Sigma_2} G(c) : F(\tau_1(c))$ is provable.

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

The lexicon is the interpreting device of ACGs.

Definition 7 (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 for the interpretation of the type α (resp. the term t): $\mathscr{L}_{name}(\alpha) = \beta$, $\mathscr{G}_{name}(\alpha) = \beta$, $\alpha :=_{name} \beta$, 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 8 (Abstract and Object Languages)**.** Given an ACG $\mathscr{G}$, the *abstract language* is defined by

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

The *object language* is defined by

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

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 $\mathscr{G}_{\textit{derived trees}}$ and $\mathscr{G}_{\textit{sem.}}$ sharing the $\Sigma_{\textit{derivations}}$ vocabulary in Figure 12. It allows for the semantic interpretation of the derivation trees and we use this in Sect. 5.

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 $\mathscr{G}_{\textit{yield}} \circ \mathscr{G}_{\textit{derived trees}}$ composition in Figure 12, allows for modularity. It also allows for admissible intermediate structures. For instance, the abstract language of $\mathscr{G}_{\textit{yield}}$

[11] such that $F(\alpha \multimap \beta) = F(\alpha) \multimap F(\beta)$ and $F(\alpha \to \beta) = F(\alpha) \to F(\beta)$

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. 4.2 to enforce the matching between node labels in substitution and adjunction operations, and to further restrict the set of derivations to only TAG derivations in Sect. 4.3.

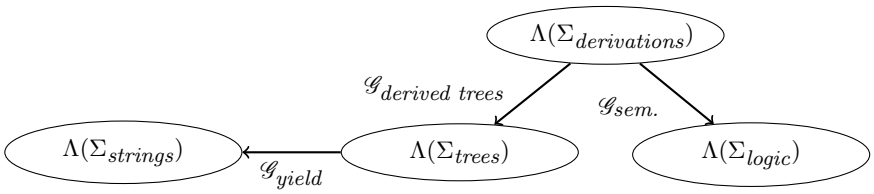


Figure 12: ACG composition for control and syntax-semantic interface

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 9 (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.

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

Table 3 sums up some of the formal properties of 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 contex-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)	hyperedge replacement gram.
$\text{ACG}_{(2,4+n)}$	$\text{ACG}_{(2,4)}$	$\text{ACG}_{(2,4)}$

Table 3: The ACG hierarchy

4 Tree-Adjoining Grammars as 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 $\mathcal{G}_{yield} = \langle \Sigma_{trees}, \Sigma_{strings}, \mathcal{L}_{yield}, \tau \rangle$ by specifying $\mathcal{L}_{yield}$ as follows:

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

For instance, (8) and ACG example 3 show that the yield of the tree represented by $\gamma_{\mathit{sleeps}}\ I\ (\gamma_{\mathit{seemingly}}\ I)\ \gamma_{\mathit{John}}$ actually is $\mathit{John} + \mathit{seemingly} + \mathit{sleeps}$ (which can be rephrased as $\gamma_{\mathit{sleeps}}\ I\ (\gamma_{\mathit{seemingly}}\ I)\ \gamma_{\mathit{John}} \coloneqq_{yield} \mathit{John} + \mathit{seemingly} + \mathit{sleeps}$).

$$
\begin{aligned}
\mathcal{G}_{yield}(\mathsf{NP}_1\ \mathit{John}) &= (\mathcal{G}_{yield}(\mathsf{NP}_1))\ (\mathcal{G}_{yield}(\mathit{John})) \quad \text{because } \mathcal{G}_{yield} \text{ is a morphism} \\
&= (\lambda^{o} x_1 . x_1)\ \mathit{John} \quad \text{by definition of } \mathcal{G}_{yield} \text{ on constants} \\
&\to_{\beta} \mathit{John}
\end{aligned}
\tag{6}
$$

$$
\begin{aligned}
&\mathcal{G}_{yield}(\mathsf{VP}_2\ (\mathsf{Adv}_1\ \mathit{seemingly})\ (\mathsf{VP}_1\ (\mathsf{V}_1\ \mathit{sleeps}))) \\
&\quad = \mathcal{G}_{yield}(\mathsf{VP}_2)\ (\mathcal{G}_{yield}(\mathsf{Adv}_1\ \mathit{seemingly}))\ (\mathcal{G}_{yield}((\mathsf{VP}_1\ (\mathsf{V}_1\ \mathit{sleeps})))) \\
&\quad = (\lambda^{o} x_1\ x_2 . x_1 + x_2)\ (\mathcal{G}_{yield}(\mathsf{Adv}_1\ \mathit{seemingly}))\ (\mathcal{G}_{yield}(\mathsf{VP}_1\ (\mathsf{V}_1\ \mathit{sleeps}))) \\
&\quad \to_{\beta} (\mathcal{G}_{yield}(\mathsf{Adv}_1\ \mathit{seemingly})) + (\mathcal{G}_{yield}(\mathsf{VP}_1\ (\mathsf{V}_1\ \mathit{sleeps}))) \\
&\quad \to_{\beta} (\mathcal{G}_{yield}(\mathsf{Adv}_1)\ (\mathcal{G}_{yield}(\mathit{seemingly}))) + ((\mathcal{G}_{yield}(\mathsf{VP}_1))\ (\mathcal{G}_{yield}(\mathsf{V}_1\ \mathit{sleeps}))) \\
&\quad = ((\lambda^{o} x_1 . x_1)\ \mathit{seemingly}) + ((\lambda^{o} x_1 . x_1)\ (\mathcal{G}_{yield}(\mathsf{V}_1\ \mathit{sleeps}))) \\
&\quad \to_{\beta} \mathit{seemingly} + ((\lambda^{o} x_1 . x_1)\ (\mathcal{G}_{yield}(\mathsf{V}_1\ \mathit{sleeps}))) \\
&\quad \to_{\beta} \mathit{seemingly} + (\mathcal{G}_{yield}(\mathsf{V}_1\ \mathit{sleeps})) \\
&\quad = \mathit{seemingly} + ((\mathcal{G}_{yield}(\mathsf{V}_1))\ (\mathcal{G}_{yield}(\mathit{sleeps}))) \\
&\quad = \mathit{seemingly} + ((\lambda^{o} x_1 . x_1)\ \mathit{sleeps}) \\
&\quad \to_{\beta} \mathit{seemingly} + \mathit{sleeps}
\end{aligned}
\tag{7}
$$

$$
\begin{aligned}
\mathcal{G}_{yield}(\gamma_4) &= \mathcal{G}_{yield}(\gamma_{\mathit{sleeps}}\ I\ (\gamma_{\mathit{seemingly}}\ I)\ \gamma_{\mathit{John}}) \\
&= \mathcal{G}_{yield}(\mathsf{S}_2\ (\mathsf{NP}_1\ \mathit{John})\ (\mathsf{VP}_2\ (\mathsf{Adv}_1\ \mathit{seemingly})\ (\mathsf{VP}_1\ (\mathsf{V}_1\ \mathit{sleeps})))) \quad \text{by (5)} \\
&= (\mathcal{G}_{yield}(\mathsf{S}_2))\ (\mathcal{G}_{yield}(\mathsf{NP}_1\ \mathit{John}))\ (\mathcal{G}_{yield}(\mathsf{VP}_2\ (\mathsf{Adv}_1\ \mathit{seemingly})\ (\mathsf{VP}_1\ (\mathsf{V}_1\ \mathit{sleeps})))) \\
&\quad \text{because } \mathcal{G}_{yield} \text{ is a morphism} \\
&= (\lambda^{o} x_1\ x_2 . x_1 + x_2)\ \mathit{John}\ (\mathit{seemingly} + \mathit{sleeps}) \\
&\quad \text{by definition of } \mathcal{G}_{yield} \text{ on constants and by (6) and (7)} \\
&\to_{\beta} \mathit{John} + (\mathit{seemingly} + \mathit{sleeps})
\end{aligned}
\tag{8}
$$

4.2 Derivation Trees and Derived Trees

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

```
# yield realize g_sleeps I (g_seemingly I) g_John : tree;
yield realize g_sleeps I (g_seemingly I) g_John : tree;

In trees:
   g_sleeps I (g_seemingly I) g_John : tree
   = S2 (NP1 John) (VP2 (Adv1 seemingly) (VP1 (V1 sleeps)))
Interpreted by yield in strings as:
   lambda z. John (seemingly (sleeps z)) : o -> o
#
```

ACG example 3: Yield computation with the "realize" command

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

$$\begin{aligned}\gamma_{13} = \gamma_{\textit{seemingly}}\ I\ \gamma_{\textit{John}} \\ &= (\lambda^o a\ v.a\ (\mathsf{VP}_2\ (\mathsf{Adv}_1\ \textit{seemingly})\ v))\ (\lambda^o x.x)\ (\mathsf{NP}_1\ \textit{John}) \\ &\rightarrow_\beta (\lambda^o v.(\lambda^o x.x)\ (\mathsf{VP}_2\ (\mathsf{Adv}_1\ \textit{seemingly})\ v))\ (\mathsf{NP}_1\ \textit{John}) \\ &= \mathsf{VP}_2\ (\mathsf{Adv}_1\ \textit{seemingly})\ (\mathsf{NP}_1\ \textit{John})\end{aligned} \tag{9}$$

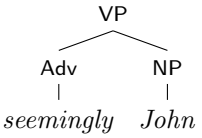


Figure 13: Tree in the abstract language of $\mathscr{G}_{yield}$

In order to avoid such terms to belong to the language we are interested in, we provide another ACG $\mathscr{G}_{derived\ trees} = \langle \Sigma_{derivations}, \Sigma_{trees}, \mathscr{L}_{derived\ trees}, \mathsf{S}\rangle$ 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 $\mathcal{O}(\mathscr{G}_{yield})$.

4.3 TAG Derivation Trees

4.3.1 A Vocabulary for Derivations: The $\Sigma_{derivations}$ Signature

Adjoining $\gamma_{\textit{seemingly}}$ on $\gamma_{\textit{John}}$ is possible in $\Lambda(\Sigma_{trees})$ because the type τ does not take the node labels into account. Hence, there is no distinction between the trees rooted by VP and trees rooted by NP, for instance. We introduce this distinction in $\Sigma_{derivations} = \langle A_{derivations}, C_{derivations}, \tau_{derivations}\rangle$. $A_{derivations} = V^0$ is the set of non-terminal symbols of the TAG grammar $\mathcal{G}$. Then, for any $\delta_{\textit{anchor}} \in \mathcal{I} \cup \mathcal{A}$ an elementary tree of $\mathcal{G}$, let define $c_{\textit{anchor}}$ 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 $\delta_{\textit{anchor}}$ labeled with a non-terminal where an adjunction is possible (by convention we use the breadth-first traversal); [12]

[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

```
# derivations check c_seemingly I_vp c_John : VP;
derivations check c_seemingly I_vp c_John : VP;

c_seemingly I_vp c_John : VP
Error: File "stdin", line 1, characters 17-23
The type of this expression is "VP" but is used with type "NP"

#
```

ACG example 4: Not well-type term

- the Y^i are the labels of the m leaves of δ_{anchor} labeled with non-terminals, *but the foot node if δ_{anchor} is an auxiliary tree* (by convention, we use the left-right order);
- let Z be the label of the root node of δ_{anchor}. $\alpha = Z$ if $\delta_{anchor} \in \mathcal{I}$ is an initial tree, and $\alpha = Z' \multimap Z$ with Z' the label of the foot node if $\delta_{anchor} \in \mathcal{A}$ is an auxiliary tree.[13] In the latter case, we call $Z' \multimap Z$ the *modifier type* of the constant modeling the auxiliary tree.

We get for instance the constants typed as in (10)[14] from the elementary trees of Fig. 1.

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

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

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

Then the set of typed constants of $\Sigma_{derivations}$ is $C_{derivations} = (\cup_{\delta_{anchor} \in \mathcal{I} \cup \mathcal{A}} c_{anchor}) \cup (\cup_{X \in V^0} I_X)$ and $\tau_{derivations}$ is the associated typing function defined as above. The typing provided by $\Sigma_{derivations}$ now disallows the application of $c_{seemingly}\ I_{\mathsf{VP}} : \mathsf{VP} \multimap \mathsf{VP}$ to $c_{John} : \mathsf{NP}$ as ACG example 4 illustrates.

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

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

In order to define $\mathcal{G}_{derived\ trees} = \langle \Sigma_{derivations}, \Sigma_{trees}, \mathcal{L}_{derived\ trees}, \mathsf{S} \rangle$ we are left with defining $\mathcal{L}_{derived\ trees}$. All the atomic types ($\mathsf{S}$, VP, etc.) are interpreted as trees (i.e., with the τ 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 4.

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; Vijay-Shanker and Joshi 1991). See note 13.

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

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

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

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

In Sect. 4.2, we noticed that $\gamma_4 = \gamma_{\mathit{seemingly}}\ I\ \gamma_{\mathit{John}} : \tau \in \mathcal{A}(\mathscr{G}_{\mathit{yield}})$ (see 9). 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 TAG 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 $\gamma_{14} = \gamma_{\mathit{sleeps}}\ I\ (\gamma_{\mathit{seemingly}}\ I)\ \gamma_{\mathit{John}}$, but it is in some sense an artifact of the definition we gave: γ_{14} β-reduces to a tree that does not show this structure anymore. However, a term such as $c_{14} = c_{\mathit{sleeps}}\ I_{\mathsf{S}}\ (c_{\mathit{seemingly}}\ I_{\mathsf{VP}})\ c_{\mathit{John}}$ 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_{14} 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 $\mathscr{G}_{\mathit{derived\ trees}}(c_{14}) = \gamma_{14}$ expresses the relation between the record of these operations and the resulting derived tree.

We can represent c_{14} as a tree (see Fig. 14(a)): 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. 14(b)). 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 tree Derivation

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.

(12) and (13) illustrate these correspondences for the abstract term $c_{14} = c_{sleeps}\ I_{\mathsf{S}}\ (c_{seemingly}\ I_{\mathsf{VP}})\ c_{John}$ representing the derivation for the analysis of *John seemingly sleeps*.

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

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

Remark 3 ($\mathscr{G}_{derived\ trees}$ 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 4). 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 the second child of S_2 (the variable s being the first). The result of this application corresponds to the resulting VP of modifier type $(\mathsf{VP} \multimap \mathsf{VP})$ (for instance $c_{usually}\ I_{\mathsf{S}}$), while the term $(\mathsf{VP}_1\ (\mathsf{V}_1\ sleeps))$ corresponds to the first VP: the parameter. So, in some sense, $(\mathsf{VP} \multimap \mathsf{VP})$ encodes the *dominance constraint* as in the tree descriptions of (Vijay-Shanker 1992).

Remark 4 ($\mathscr{G}_{derived\ trees}$ terms and TAG derivation trees)**.** It should be noted that $\mathscr{G}_{derived\ trees}$ and $\mathscr{G}_{yield} \circ \mathscr{G}_{derived\ trees}$ are *not* second-order ACGs. It means that the polynomial parsing results do not directly apply. But we know that TAG parsing is polynomial. So what is happening here?

The answer is that while $\mathscr{G}_{derived\ trees}$ constrains the string language more than $\mathscr{G}_{yield}$ does, it does not constrain it enough to generate only the corresponding TAG language. Indeed, let us assume a term $c_{matters} : (\mathsf{VP} \multimap \mathsf{VP}) \multimap \mathsf{S} \multimap \mathsf{S}$ and a term $\gamma_{matters} = \lambda^o S\ a\ s.S\ (\mathsf{S}_2\ s\ (a\ (\mathsf{VP}_1\ (\mathsf{V}_1\ matters))))$ corresponding to the initial tree $\alpha_{matters}$ of Fig. 15(a) (as in *To arrive on time matters considerably*,

[15] With some reservations that Sect. 6 clears up, though. See Rem. 4.

see (XTAG Research Group 2001, Section 6.31)) where the S leaf is a substitution node,[16] the VP node accepts adjunctions, but the S node does not. We define $\mathscr{G}_{derived\ trees}(c_{matters}) = \gamma_{matters}$. Then the term $(c_{matters}\ I_{\mathsf{VP}})$ is typed $\mathsf{S} \multimap \mathsf{S}$, which is the modifier type of terms encoding auxiliary trees adjoining on S nodes. Let us assume there also is a term $c_{that\ sleeps} : (\mathsf{VP} \multimap \mathsf{VP}) \multimap \mathsf{NP} \multimap \mathsf{S}$ such that $\mathscr{G}_{derived\ trees}(c_{that\ sleeps}) = \lambda^{\circ} v\ s.\mathsf{S}_2\ (\mathsf{Comp}\ that)\ (\mathsf{S}_2\ s\ (\mathsf{VP}_1\ (\mathsf{V}_1\ sleeps)))$ corresponding to the initial tree of Fig. 15(b).[17]

Then we can built the term c_{16} of (14) corresponding to the derived tree of Fig. 16 by applying $c_{matters}\ I_{\mathsf{VP}} : \mathsf{S} \multimap \mathsf{S}$ to $(c_{that\ sleeps}\ I_{\mathsf{VP}}\ c_{John}) : \mathsf{S}$. This indeed models a substitution.

But we can also build the term c_{17} of (14) corresponding to the syntactic tree of Fig. 17. However, the latter is not a TAG tree as it would correspond to the *adjunction* of the tree for *matters* on the tree for *John sleeps*. The term $(\gamma_{matters}\ I_{\mathsf{VP}}) : \mathsf{S} \multimap \mathsf{S}$ indeed now appears as *argument* of c_{sleeps}, simulating an adjunction on c_{sleeps}.[18] But the tree for *matters* is an initial tree and should not be adjoined.

$$
\begin{aligned}
c_{16} &= c_{matters}\ I_{\mathsf{VP}}\ (c_{that\ sleeps}\ I_{\mathsf{VP}}\ c_{John}) \\
\mathscr{G}_{derived\ trees}(c_{16}) &= \mathsf{S}_2\ (\mathsf{Comp}_1\ that) \\
&\qquad (\mathsf{S}_2\ (\mathsf{NP}_1\ John)\ (\mathsf{VP}_1\ (\mathsf{V}_1\ sleeps)))\ (\mathsf{VP}_1\ (\mathsf{V}_1\ matters)) \\
\mathscr{G}_{yield} \circ \mathscr{G}_{derived\ trees}(c_{16}) &= that + John + sleeps + matters \\
c_{17} &= c_{sleeps}\ (c_{matters}\ I_{\mathsf{VP}})\ I_{\mathsf{VP}}\ c_{John} \\
\mathscr{G}_{derived\ trees}(c_{17}) &= \mathsf{S}_2\ (\mathsf{S}_2(\mathsf{NP}_1\ John)\ (\mathsf{VP}_1\ (\mathsf{V}_1 sleeps)))\ (\mathsf{VP}_1\ (\mathsf{V}_1\ matters)) \\
\mathscr{G}_{yield} \circ \mathscr{G}_{derived\ trees}(c_{17}) &= John + sleeps + matters
\end{aligned}
\tag{14}
$$

The solution we develop in Section 6 is to further control $\mathcal{A}(\mathscr{G}_{derived\ trees})$ with another ACG $\mathscr{G}_{TAG}$ such that $\mathcal{O}(\mathscr{G}_{TAG}) \subset \mathcal{A}(\mathscr{G}_{derived\ trees})$ just as $\mathscr{G}_{derived\ trees}$ allows us to control $\mathcal{A}(\mathscr{G}_{yield})$. The general architecture is then the one Fig. 18 describes. We delay the definition of $\mathscr{G}_{TAG}$ to Sect. 6. It is very similar to the definition of $\mathscr{G}_{derived\ trees}$ except that the adjunction sites are not given a second-order type $(X \multimap X)$ but an atomic type X_A, making $\mathscr{G}_{TAG}$ second-order. Then, $\mathscr{G}_{derived\ trees} \circ \mathscr{G}_{TAG}$ corresponds to the TAG encoding of (de Groote 2002). However, while useful to faithfully encode TAGs and to allow for applying the polynomiality results, this latter encoding is not necessary to provide the semantic interpretation of the derivation trees (and may somewhat obfuscate it, as the interpretation of a $(\mathsf{S} \multimap \mathsf{S})$ type may be more straightforward as the interpretation of a S_A type).

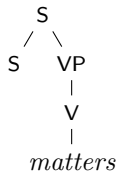
5 Semantic Construction

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

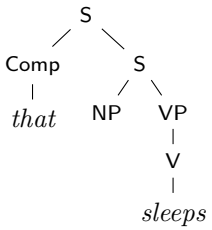
[16] In this example, we do not model the requirement for the sentential subject to be infinite or to be finite and introduced by a complementizer.

[17] For sake of simplicity, we make the complementizer part of the initial tree.

[18] Except the lack of the complementizer, the surface forms are almost the same. This is due to adjoining at the root node.

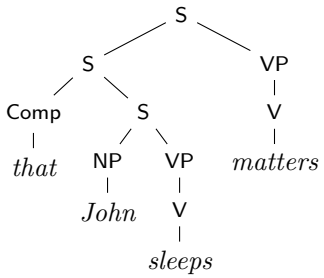


(a) Initial tree for *matters* (the S leaf is a substitution node)

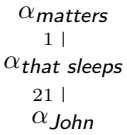


(b) Initial tree for *that . . . sleeps*

Figure 15: Initial trees for verbs with sentential subjects and complementized clause

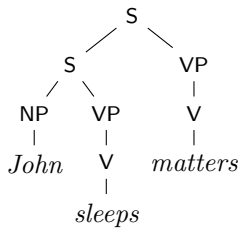


(a) Derived tree for *that John sleeps matters*

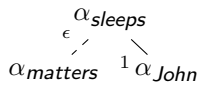


(b) Derivation tree for *that John sleeps matters*

Figure 16: TAG derived and derivation trees for *that John sleeps matters*



(a) Syntactic tree which is not a TAG derived tree



(b) Forbidden corresponding derivation tree

Figure 17: Non-TAG derived and derivation trees for *John sleeps matters*

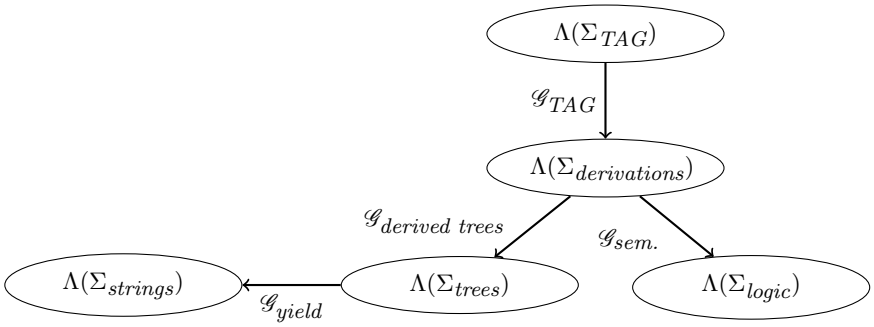


Figure 18: ACG composition for control and syntax-semantics interface

languages, etc. are possible as well. But we want to focus on *how* to build semantic representations rather than on the semantic modeling 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 5 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 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.

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}

5.2 Derivation Tree Based Interpretation

The first step in defining $\mathscr{G}_{sem.}$ to interpret the abstract vocabulary $\Sigma_{derivations}$ into types and terms build on the object vocabulary Σ_{logic} is to define the interpretation of the atomic types (S, VP...). We simply follow the standard interpretation of these syntactic types into the semantic types as proposed in (Montague 1973). This results in the interpretation described in Table 6(a). The interpretation of the constants follows as in Table 6(b). 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 $\mathscr{G}_{sem.}(c_{\mathsf{anchor}})$ (e.g., the type of c_{John} is NP, hence $[\![c_{\mathsf{John}}]\!]_{sem.} : [\![\mathsf{NP}]\!]_{sem.} = (e \multimap t) \multimap t$). But the reader can check this proviso holds.

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

(a) Interpretation of the atomic types

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

(b) Interpretation of the constants

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

Using our favorite examples, (15) shows that $c_{14} :=_{sem.} \mathbf{seemingly}\ (\mathbf{sleep}\ \mathbf{john})$.

$$
\begin{aligned}
& c_{14} = c_{\mathit{sleeps}}\ I_{\mathsf{S}}\ (c_{\mathit{seemingly}}\ I_{\mathsf{VP}})\ c_{\mathit{John}} \\
& :=_{sem.} (\lambda^{\circ} adv_{\mathsf{S}}\ adv_{\mathsf{VP}}\ subj.adv_{\mathsf{S}}\ (subj\ (adv_{\mathsf{VP}}\ (\lambda^{\circ} x.\mathbf{sleep}\ x))))\ [\![I_{\mathsf{S}}]\!]_{sem.}\ [\![c_{\mathit{seemingly}}\ I_{\mathsf{VP}}]\!]_{sem.}\ [\![c_{\mathit{John}}]\!]_{sem.} \\
& \to_{\beta} (\lambda^{\circ} adv_{\mathsf{VP}}\ subj.[\![I_{\mathsf{S}}]\!]_{sem.}\ (subj\ (adv_{\mathsf{VP}}\ (\lambda^{\circ} x.\mathbf{sleep}\ x))))\ [\![c_{\mathit{seemingly}}\ I_{\mathsf{VP}}]\!]_{sem.}\ [\![c_{\mathit{John}}]\!]_{sem.} \\
& = (\lambda^{\circ} adv_{\mathsf{VP}}\ subj.(\lambda^{\circ} x.x)\ (subj\ (adv_{\mathsf{VP}}\ (\lambda^{\circ} x.\mathbf{sleep}\ x))))\ [\![c_{\mathit{seemingly}}\ I_{\mathsf{VP}}]\!]_{sem.}\ [\![c_{\mathit{John}}]\!]_{sem.} \\
& \to_{\beta} (\lambda^{\circ} adv_{\mathsf{VP}}\ subj.subj\ (adv_{\mathsf{VP}}\ (\lambda^{\circ} x.\mathbf{sleep}\ x)))\ [\![c_{\mathit{seemingly}}\ I_{\mathsf{VP}}]\!]_{sem.}\ [\![c_{\mathit{John}}]\!]_{sem.} \\
& \to_{\beta} [\![c_{\mathit{John}}]\!]_{sem.}\ ([\![c_{\mathit{seemingly}}\ I_{\mathsf{VP}}]\!]_{sem.}\ (\lambda^{\circ} x.\mathbf{sleep}\ x)) \\
& = [\![c_{\mathit{John}}]\!]_{sem.}\ (([\![c_{\mathit{seemingly}}]\!]_{sem.}\ [\![I_{\mathsf{VP}}]\!]_{sem.})\ (\lambda^{\circ} x.\mathbf{sleep}\ x)) \\
& = [\![c_{\mathit{John}}]\!]_{sem.}\ (((\lambda^{\circ} adv_{mod}\ pred.\lambda^{\circ} x.adv_{mod}\ (\mathbf{seemingly}\ (pred\ x)))\ [\![I_{\mathsf{VP}}]\!]_{sem.})\ (\lambda^{\circ} x.\mathbf{sleep}\ x)) \\
& = [\![c_{\mathit{John}}]\!]_{sem.}\ ((\lambda^{\circ} pred.\lambda^{\circ} x.[\![I_{\mathsf{VP}}]\!]_{sem.}\ (\mathbf{seemingly}\ (pred\ x)))\ (\lambda^{\circ} x.\mathbf{sleep}\ x)) \\
& = [\![c_{\mathit{John}}]\!]_{sem.}\ ((\lambda^{\circ} pred.\lambda^{\circ} x.(\lambda^{\circ} x.x)\ (\mathbf{seemingly}\ (pred\ x)))\ (\lambda^{\circ} x.\mathbf{sleep}\ x)) \\
& \to_{\beta} [\![c_{\mathit{John}}]\!]_{sem.}\ ((\lambda^{\circ} pred.\lambda^{\circ} x.\mathbf{seemingly}\ (pred\ x))\ (\lambda^{\circ} x.\mathbf{sleep}\ x)) \\
& \to_{\beta} [\![c_{\mathit{John}}]\!]_{sem.}\ (\lambda^{\circ} x.\mathbf{seemingly}\ ((\lambda^{\circ} x.\mathbf{sleep}\ x)\ x)) \\
& \to_{\beta} [\![c_{\mathit{John}}]\!]_{sem.}\ (\lambda^{\circ} x.\mathbf{seemingly}\ (\mathbf{sleep}\ x)) \\
& = (\lambda^{\circ} P.P\ \mathbf{john})\ (\lambda^{\circ} x.\mathbf{seemingly}\ (\mathbf{sleep}\ x)) \\
& \to_{\beta} (\lambda^{\circ} x.\mathbf{seemingly}\ (\mathbf{sleep}\ x))\ \mathbf{john} \\
& \to_{\beta} \mathbf{seemingly}\ (\mathbf{sleep}\ \mathbf{john})
\end{aligned}
\tag{15}
$$

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 (16–18).

(16) Paul claims John loves Mary

(17) Mary Paul claims John seems to love

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

The TAG analysis relies on the elementary trees of Fig. 19 and results in the derived tree and derivation tree of Fig. 20 for (16). 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 (17) as the derivation tree, shown Fig. 21(b), makes $\alpha_{to\ love}$ scope over both β_{claims} and β_{seems}, while semantically both should scope over the **love** predicate. Moreover, the derivation tree does not specify any scoping relation between the two auxiliary trees, whereas we expect **claim** to semantically scope over **seem**.

Finally, (18) and the derivation tree of Fig. 22(b) illustrate how an element such as a *wh*-word can scope over a whole sentence and all its predicates while providing a semantic argument to the semantically "lowest" predicate (**love**).

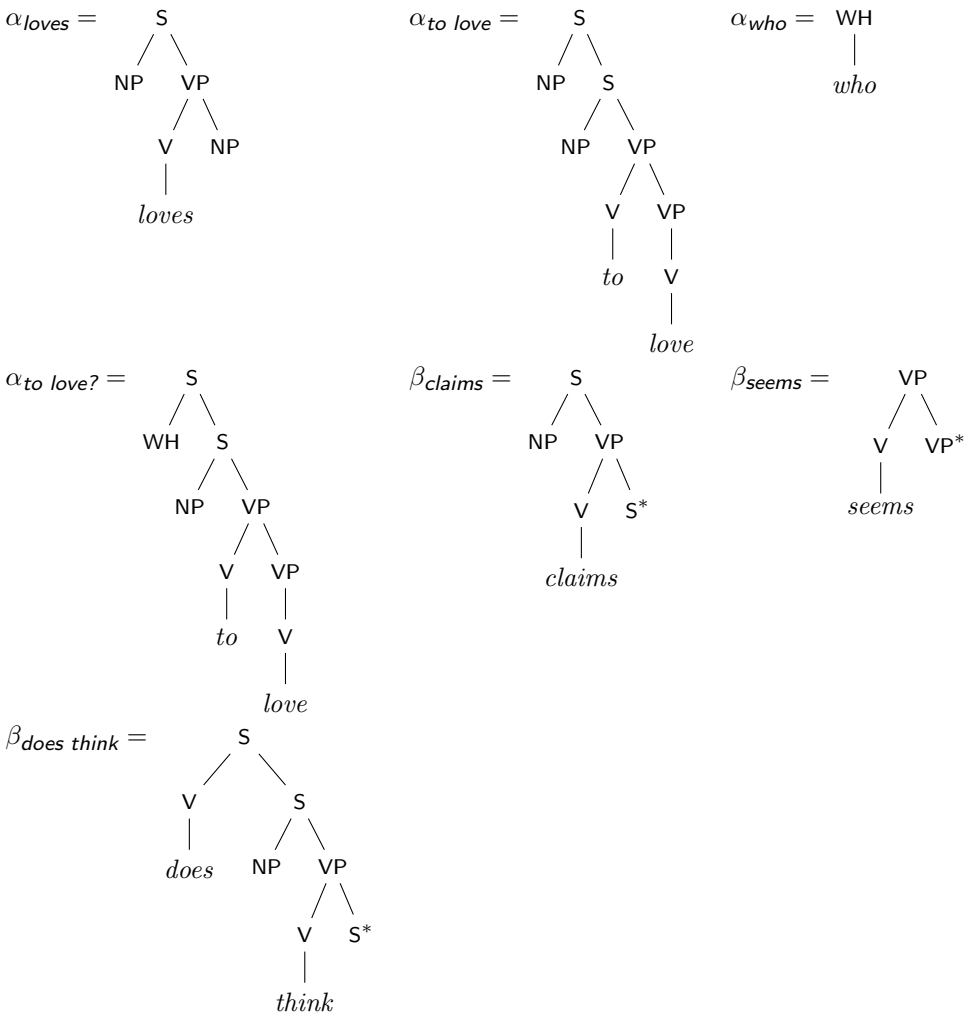
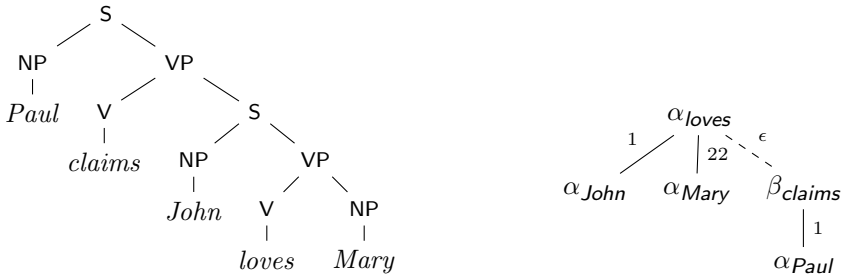
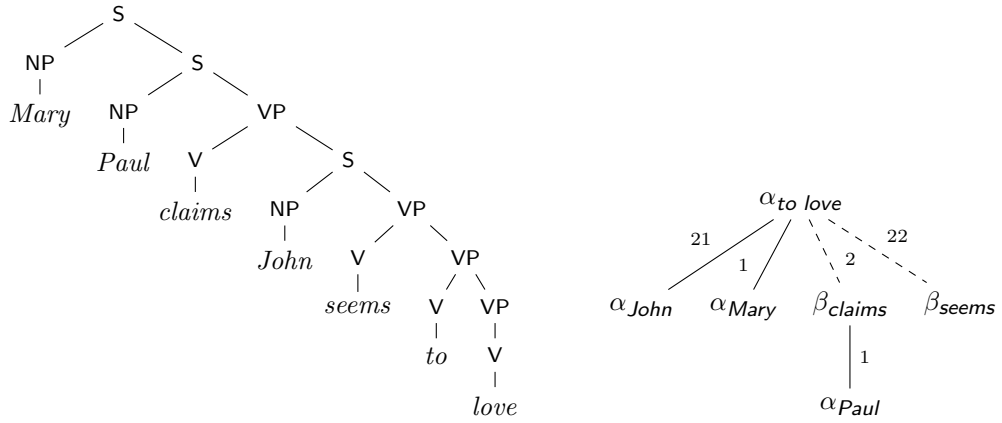


Figure 19: TAG elementary trees for long distance dependencies



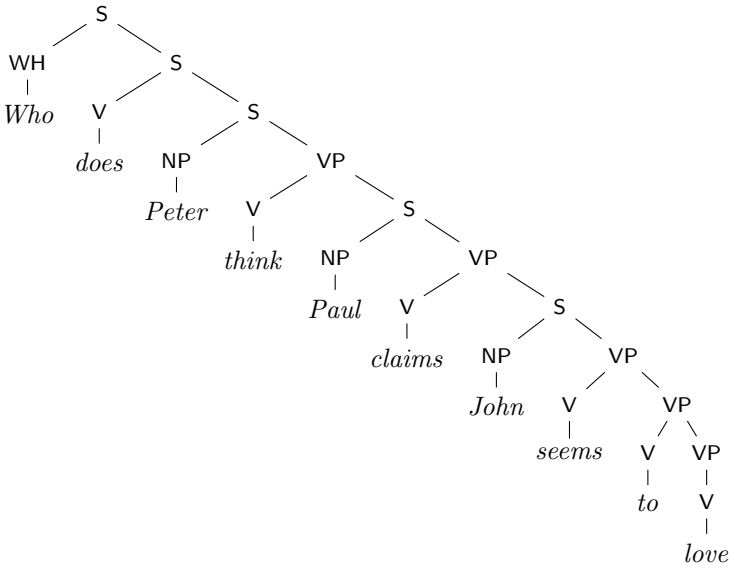
(a) Derived tree (b) Derivation tree

Figure 20: TAG analysis of *Paul claims John loves Mary*

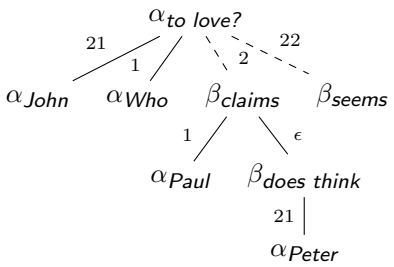


(a) Derived tree (b) Derivation tree

Figure 21: TAG analysis of *Mary Paul claims John seems to love*



(a) Derived tree



(b) Derivation tree

Figure 22: TAG analysis of *Who does Peter think Paul claims John seems to love*

To account for these phenomena, we first extend $\Sigma_{derivations}$ and $\mathscr{G}_{derived\ trees}$ to represent the trees of Fig. 19. Table 7 shows the new constants and their interpretations. The terms c_{20}, c_{21}, and c_{22} in (19) represent the derivation trees of Fig. 20(b), 21(b), and 22(b) respectively. We leave it to the reader to check that the $\mathscr{G}_{derived\ trees}$ interpretations of theses terms are the derived trees of the corresponding figures 20(a), 21(a), and 22(a) respectively.

$$\begin{aligned} c_{20} &= c_{\mathit{loves}}\ (c_{\mathit{claims}}\ I_{\mathsf{S}}\ I_{\mathsf{VP}}\ c_{\mathit{Paul}})\ I_{\mathsf{VP}}\ c_{\mathit{John}}\ c_{\mathit{Mary}} \\ c_{21} &= c_{\mathit{to\ love}}\ (c_{\mathit{claims}}\ I_{\mathsf{S}}\ I_{\mathsf{VP}}\ c_{\mathit{Paul}})\ (c_{\mathit{seems}}\ I_{\mathsf{VP}})\ c_{\mathit{Mary}}\ c_{\mathit{John}} \\ c_{22} &= c_{\mathit{to\ love?}}\ (c_{\mathit{claims}}\ (c_{\mathit{does\ think}}\ I_{\mathsf{S}}\ I_{\mathsf{VP}}\ c_{\mathit{Peter}})\ I_{\mathsf{VP}}\ c_{\mathit{Paul}})\ (c_{\mathit{seems}}\ I_{\mathsf{VP}})\ c_{\mathit{Mary}}\ c_{\mathit{John}} \end{aligned} \tag{19}$$

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

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

We now need to define the $\mathscr{G}_{sem.}$ interpretation that provides the expected semantic dependencies. Table 8 shows the lexical semantics fulfilling the requirements. In the term $c_{20} = c_{\mathit{loves}}\ (c_{\mathit{claims}}\ I_{\mathsf{S}}\ I_{\mathsf{VP}}\ c_{\mathit{Paul}})\ I_{\mathsf{VP}}\ c_{\mathit{John}}\ c_{\mathit{Mary}}$, the constant c_{loves} scopes over c_{claims} as in the derivation tree of Fig. 20(b). However, looking at $[\![c_{\mathit{loves}}]\!]_{sem.}$ in Table 8, we observe that its first argument adv_{S} scopes over the **love** predicate. This argument actually corresponds to the meaning of the auxiliary tree adjoined at the S node of α_{loves}. When it is replaced by some actual value, for instance $(c_{\mathit{claims}}\ I_{\mathsf{S}}\ I_{\mathsf{VP}}\ c_{\mathit{Paul}})$, 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 $[\![c_{\textit{to love?}}]\!]_{sem.}$, 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 $\alpha_{\textit{to love?}}$ (see Fig. 19), typically $\lambda^{\circ}P.\mathbf{WHO}\ P : (e \multimap t) \multimap t$, making **WHO** eventually take scope over all the other predicates.

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

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

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

$$
\begin{array}{l}
c_{20} = c_{\textit{loves}}\ (c_{\textit{claims}}\ I_{\mathsf{S}}\ I_{\mathsf{VP}}\ c_{\textit{Paul}})\ I_{\mathsf{VP}}\ c_{\textit{John}}\ c_{\textit{Mary}} \\
\quad \coloneqq_{sem.} \mathbf{claim\ paul\ (love\ john\ mary)} \\
c_{21} = c_{\textit{to love}}\ (c_{\textit{claims}}\ I_{\mathsf{S}}\ I_{\mathsf{VP}}\ c_{\textit{Paul}})\ (c_{\textit{seems}}\ I_{\mathsf{VP}})\ c_{\textit{Mary}}\ c_{\textit{John}} \\
\quad \coloneqq_{sem.} \mathbf{claim\ paul\ (seem\ john\ (}\lambda x.\mathbf{love}\ x\ \mathbf{mary))} \\
c_{22} = c_{\textit{to love?}}\ (c_{\textit{claims}}\ (c_{\textit{does think}}\ I_{\mathsf{S}}\ I_{\mathsf{VP}}\ c_{\textit{Peter}})\ I_{\mathsf{VP}}\ c_{\textit{Paul}})\ (c_{\textit{seems}}\ I_{\mathsf{VP}})\ c_{\textit{Mary}}\ c_{\textit{John}} \\
\quad \coloneqq_{sem.} \mathbf{WHO\ (}\lambda y.\mathbf{think\ peter\ (claim\ paul\ (seem\ john\ (}\lambda x.\mathbf{love}\ x\ y))))
\end{array}
\tag{20}
$$

5.3.2 Quantification

A similar effect of scope inversion between the derivation tree and the semantic representation occurs with quantified noun phrases such as *everyone*, *someone*, *every man*, etc. The trees of Fig. 23 provide the TAG elementary trees for the TAG analysis for (21) (resp. for (22)) shown in Fig. 24 (resp. in Fig. 25).

(21) everyone loves someone

(22) every man loves some woman

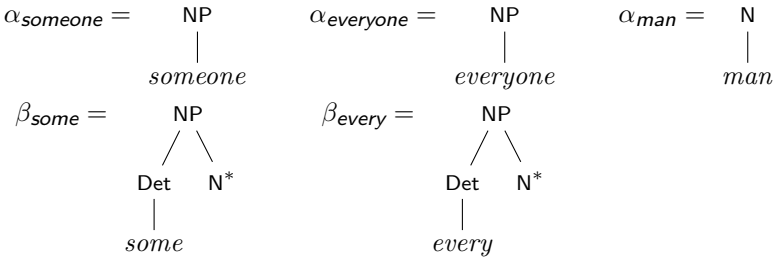


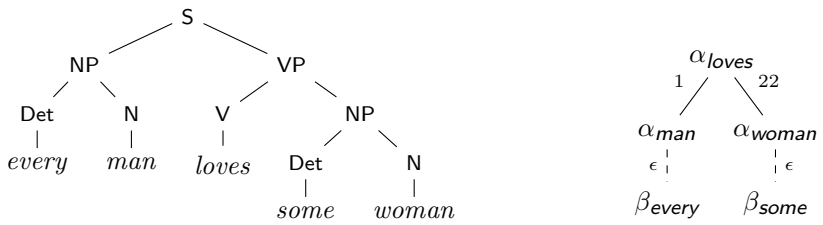
Figure 23: Determiners and quantifiers

Remark 5. 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. 23 look unusual because the root node and the foot node have not 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, Kiefer, and Netter (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 for auxiliary trees with different labels as root and foot nodes. While we discuss the modeling of features in TAG more generally in Section 7, the NP and N notations allow us to model the auxiliary trees of determiners with constants of the usual $\mathsf{N} \multimap \mathsf{NP}$ type (to be compared with a $\mathsf{NP}[-] \multimap \mathsf{NP}[+]$ type). While we could avoid introducing this distinction on the syntactic part of the TAG modeling, 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 6(a)).

The type of the constants modeling 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}) modeling initial trees anchored by nouns (e.g., α_{man}) is: $(\mathsf{N} \multimap \mathsf{NP}) \multimap \mathsf{NP}$.

(a) Derived tree (b) Derivation tree

Figure 24: TAG analysis of *everyone loves someone*



(a) Derived tree (b) Derivation tree

Figure 25: TAG analysis of *everyone loves someone*

The derivation trees of Fig. 24 and 25 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 $\Sigma_{\mathit{derivations}}$ with the constants of Table 9. This table also provides the interpretation of these constants by $\mathscr{G}_{\mathit{derived\ trees}}$ modeling the elementary trees of Fig. 23. The terms of (23) belong to $\mathcal{A}(\mathscr{G}_{\mathit{derived\ trees}})$ and represent the derivation trees of Fig. 24 and 25. (24) shows they

are interpreted as the derived trees of the same Figures.

$$\begin{aligned} c_{24} &= c_{\mathit{loves}}\ I_{\mathsf{S}}\ I_{\mathsf{VP}}\ c_{\mathit{everyone}}\ c_{\mathit{someone}} \\ c_{25} &= c_{\mathit{loves}}\ I_{\mathsf{S}}\ I_{\mathsf{VP}}\ (c_{\mathit{man}}\ c_{\mathit{every}})\ (c_{\mathit{woman}}\ c_{\mathit{some}}) \end{aligned} \tag{23}$$

$$c_{24} :=_{\mathit{derived\ trees}} \mathsf{S}_2\ (\mathsf{NP}_1\ \mathit{everyone})\ (\mathsf{VP}_2\ (\mathsf{V}_1\ \mathit{loves})\ (\mathsf{NP}_1\ \mathit{someone}))$$

$$c_{25} :=_{\mathit{derived\ trees}} \mathsf{S}_2\ (\mathsf{NP}_2\ (\mathsf{Det}_1\ \mathit{every})\ (\mathsf{N}_1\ \mathit{man}))\ (\mathsf{VP}_2\ (\mathsf{V}_1\ \mathit{loves})\ (\mathsf{NP}_2\ (\mathsf{Det}_1\ \mathit{some})\ (\mathsf{N}_1\ \mathit{woman}))) \tag{24}$$

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

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

Then we extend $\mathscr{G}_{\mathit{sem.}}$ with the interpretations of these new constants of $\Sigma_{\mathit{derivations}}$ as terms of $\Lambda(\Sigma_{\mathit{logic}})$. The semantic interpretations of the terms c_{24} and c_{25} are then as expected, as (25) shows.

$$\begin{aligned} c_{24} &:=_{\mathit{sem.}} \forall x.(\mathbf{human}\ x) \Rightarrow (\exists y.(\mathbf{human}\ y) \wedge (\mathbf{love}\ x\ y)) \\ c_{25} &:=_{\mathit{sem.}} \forall x.(\mathbf{man}\ x) \Rightarrow (\exists y.(\mathbf{woman}\ y) \wedge (\mathbf{love}\ x\ y)) \end{aligned} \tag{25}$$

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) discusses the relation with the present approach, relying on the encoding of macro-tree transduction within 2nd order ACGs (Yoshinaka 2006).

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

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

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) calls it *dependent* and advocate for an alternative *independent* notion. With dependent derivations, multiple adjunction on the same node is forbidden. So the analysis of (26) requires first the adjunction of β_{big} and β_{black}, and the adjunction of the result on α_{dog}. Figure 28(a) shows the resulting derivation

tree. On the other hand, the independent adjunction shown in Fig. 28(b) only specifies that both adjectives adjoin on the N node of the initial tree, specifying both the derived tree of Fig. 27(a) and the derived tree of Fig. 27(b).

(26) big black dog

(27) black big dog

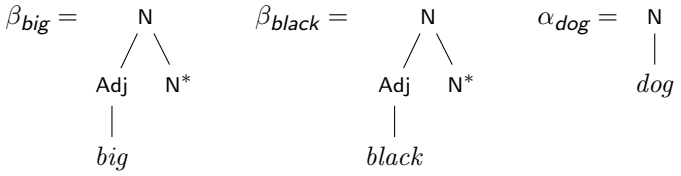
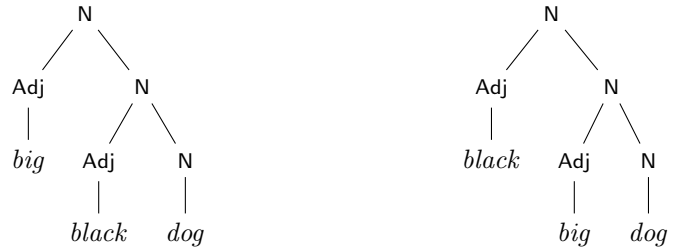


Figure 26: TAG elementary trees for adjectives



(a) Derived tree for (26)

(b) Derived tree for (27)

Figure 27: Alternative notions of derived trees

(a) Dependent derivation tree

(b) Independent derivation tree

Figure 28: Alternative notions of derivation trees

(Schabes and Shieber 1994) presents several arguments in favor 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.[19] The main concern again has to do with the relation between derivation trees and semantic dependencies.

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

The dependent derivation of Fig. 28(a) 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 and their interpretations by $\mathscr{G}_{derived\ trees}$ and $\mathscr{G}_{sem.}$ of Table 11. The types of the the constants modeling adjectives follow the types proposed for constants modeling 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 $(\mathsf{N} \multimap \mathsf{NP}) \multimap \mathsf{N} \multimap \mathsf{NP}$.

(28) shows the interpretations of the term $c_{28} : (\mathsf{N} \multimap \mathsf{NP}) \multimap \mathsf{NP}$ (an expression missing a determiner of type $(\mathsf{N} \multimap \mathsf{NP})$ to provide a NP) that encodes the derivation tree of Fig. 28(a) is interpreted. The interpretation by $\mathscr{G}_{sem.}$ indeed provides a separated modification of the same variable x.

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

	Constants of $\Sigma_{derivations}$	Interpretation by $\mathscr{G}_{derived\ trees}$	Interpretation by $\mathscr{G}_{sem.}$
c_{big}	$: (\mathsf{N} \multimap \mathsf{NP}) \multimap \mathsf{N} \multimap \mathsf{NP}$	$\lambda^{o} a\ n.a\ (\mathsf{N}_2\ (\mathsf{Adj}_1\ \mathit{big})\ n)$	$\lambda^{o} Q\ n.\lambda^{o} q.Q\ (\lambda x.(n\ x) \wedge (\mathbf{big}\ x))\ q$
c_{black}	$: (\mathsf{N} \multimap \mathsf{NP}) \multimap \mathsf{N} \multimap \mathsf{NP}$	$\lambda^{o} a\ n.a\ (\mathsf{N}_2\ (\mathsf{Adj}_1\ \mathit{black})\ n)$	$\lambda^{o} Q\ n.\lambda^{o} q.Q\ (\lambda x.(n\ x) \wedge (\mathbf{black}\ x))\ q$
c_{dog}	$: (\mathsf{N} \multimap \mathsf{NP}) \multimap \mathsf{NP}$	$\lambda^{o} d.d\ (\mathsf{N}_1\ \mathit{dog})$	$\lambda^{o} Q.\lambda^{o} q.Q\ \mathbf{dog}\ q$

Table 11: Interpretation by $\mathscr{G}_{sem.}$ of the $\Sigma_{derivations}$ vocabulary—multiple adjunction

Remark 6. By not introducing a constant $I_{\mathsf{N}} : \mathsf{N} \multimap \mathsf{NP}$ in $\Sigma_{derivations}$, we require actual adjunctions of determiners (of type $(\mathsf{N} \multimap \mathsf{NP})$, e.g., c_{some}) on nouns or on nouns modified by adjectives.

6 Completing the TAG into ACG Encoding

So far, the abstract signature we used, in particular $\Sigma_{derivations}$, introduce constants that are of order strictly greater than 2. This comes in particular from the modeling of auxiliary trees as functions (typically of type $X \multimap X$), hence from having constants of higher-order type modeling 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. From a practical point of view, Remark 4 (p. 22) 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 modeling an auxiliary tree of foot node S from an abstract constant modeling 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 type of the constants modeling auxiliary trees. So in addition to the ACG $\mathscr{G}_{derived\ trees} = \langle \Sigma_{derivations}, \Sigma_{trees}, \mathscr{L}_{derived\ trees}, \mathsf{S} \rangle$, we also define a higher-order signature $\Sigma_{TAG} = \langle A_{TAG}, C_{TAG}, \tau_{TAG} \rangle$ such that $A_{TAG} = A_{derivations} \bigcup_{X \in A_{derivations}} X_A$.

For any $\delta_{anchor} \in \mathcal{I} \cup \mathcal{A}$ 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, *but the foot node if δ_{anchor} is an auxiliary tree* (by convention, we use the left-right order traversal);
- let Z be the label of the root node of δ_{anchor}. $\alpha = Z$ if $\delta_{anchor} \in \mathcal{I}$ is an initial tree, and $\alpha = Z_A$ with Z the label of the foot node if $\delta_{anchor} \in \mathcal{A}$ is an auxiliary tree.

We get for instance the constants typed as (29) shows from the elementary trees of Fig. 1.

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

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

Then the set of typed constants of Σ_{TAG} is $C_{TAG} = (\cup_{\delta_{anchor} \in \mathcal{I} \cup \mathcal{A}} C_{anchor}) \cup (\cup_{X \in V^0} I_X)$ and τ_{TAG} is the associated typing function defined as above. The typing provided by Σ_{TAG} distinguish the typing of the constant $C_{matters} : \mathsf{VP}_A \multimap \mathsf{S} \multimap \mathsf{S}$ and the typing S_A of constants standing for auxiliary tree of root node S. So $C_{matters}\ I_{\mathsf{VP}} : \mathsf{S} \multimap \mathsf{S}$ cannot be argument of $C_{sleeps} : \mathsf{S}_A \multimap \mathsf{VP}_A \multimap \mathsf{NP} \multimap \mathsf{S}$, contrary to what happens in 14 (p. 22).

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

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

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

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

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

Table 12 sums up the constants corresponding to the elementary trees introduced so far as well as their interpretations. Because constants are interpreted as constants, the terms of $\Lambda(\Sigma_{TAG})$ and their interpretations are isomorphic. However, some terms of $\Lambda(\Sigma_{derivations})$ have no antecedent by $\mathcal{L}_{TAG}$ (e.g., $c_{17} = c_{sleeps}\ (c_{matters}\ I_{\mathsf{VP}})\ I_{\mathsf{VP}}\ c_{John} : \mathsf{S}$, see Remark 4 p. 22 and ACG example 5). 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. 18 (p. 24). 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.

Similarly, Σ_{TAG} strictly follows the abstract signature definition proposed in (de Groote 2002) to encode the syntactic part of TAGs. And we get by the function composition $\mathcal{G}_{derived\ trees} \circ \mathcal{G}_{TAG}$ the ACG defined in (de Groote 2002).

```
# derivations check c_sleeps (c_matters l_vp) l_vp c_John : S;
derivations check c_sleeps (c_matters l_vp) l_vp c_John : S;

In derivations:
   c_sleeps (c_matters l_vp) l_vp c_John : S
# TAG_derivations parse c_sleeps (c_matters l_vp) l_vp c_John : S;
TAG_derivations parse c_sleeps (c_matters l_vp) l_vp c_John : S;
No solution.
#
```

ACG example 5: c_{17} is a term of $\Lambda(\Sigma_{derivations})$ which is in $\mathcal{A}(\mathscr{G}_{derived\ trees})$ but not in $\mathcal{O}(\mathscr{G}_{TAG})$

```
# yield_TAG parse every + big + black + dog + usually + barks :S;
yield_TAG parse every + big + black + dog + usually + barks :S;
An antecedent by yield_TAG in TAG is:
C_barks I_s (C_usually I_vp) (C_dog (C_black (C_big C_every))) : S
Do you want to look for another solution?
        y/yes
        n/no
        a/all
(Default: yes):

No other possible value
#
```

ACG example 6: ACG parsing from a string

Remark 7. Because the modeling of adjunction is now controlled by the interpretation of the types X_A from Σ_{TAG}, we see that we can have more freedom in the type that is given in $\Sigma_{derivations}$. For instance, we can set $\mathsf{N}_A :=_{TAG} \mathsf{N} \multimap \mathsf{NP}$. We can use even more complex interpretations if it helps explaining the semantic interpretation. For instance, in Section 7.2 we introduce a type $\mathsf{S}'_A :=_{TAG} (\mathsf{NP} \multimap \mathsf{S}) \multimap \mathsf{S}$ to model control verbs.

Since Σ_{TAG} is 2nd order, we may parse with ACGs that have it as abstract vocabulary. ACG example 6 shows the result of parsing from the string *every+ big+ black+ dog+ usually+ barks*. ACG example 7 shows the result of parsing from 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.

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.

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

Table 12: Σ_{TAG} constants and their interpretation by $\mathscr{L}_{TAG}$

```
# sem_TAG parse All x. (((dog x) & (black x)) & (big x)) > (usually (bark x)) : S;
sem_TAG parse All x. (((dog x) & (black x)) & (big x)) > (usually (bark x)) : S;
An antecedent by sem_TAG in TAG is:
C_barks I_s (C_usually I_vp) (C_dog (C_black (C_big C_every))) : S
Do you want to look for another solution?
        y/yes
        n/no
        a/all
(Default: yes):

No other possible value
#
```

ACG example 7: ACG parsing from a logical formula

7.1 Obligatory Adjoining Constraints

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

7.2 Selective Adjoining Constraints

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

As noticed in Remark 7, the key here is to model auxiliary trees using an atomic type X_A, so that the ACG is second-order, and to interpret this type as a functional type $X \multimap X'$ of $\Sigma_{derivations}$, without the actual requirement that X and X' are atomic or that $X = X'$. We illustrate such an encoding with the TAG analysis of control verbs.

TAG analyzes a sentence such as (30) with an adjunction of the control verb *wants* on the reduced clause *to sleep* as Fig. 30 shows. Figure 29 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 30 shows the derived tree and the derivation trees for (30).

(30) John wants to sleep

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

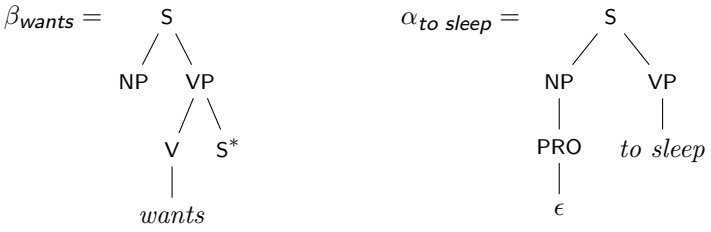
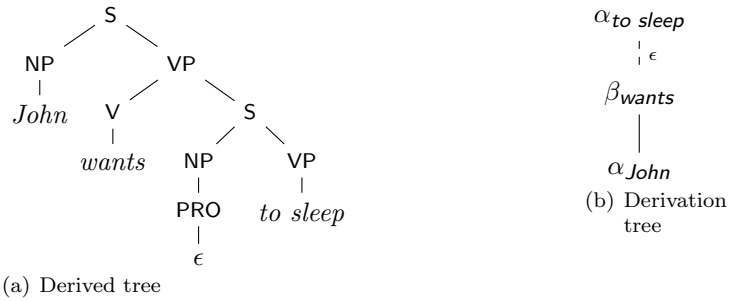


Figure 29: Elementary trees for control verbs



(a) Derived tree

(b) Derivation tree

Figure 30: 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 15 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.

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

Table 13: $\mathscr{G}_{TAG}$ extension—control verbs

Let C_{30} of $\Lambda(\Sigma_{TAG})$ in 31 represent the derivation tree of Fig. 30, and let c_{30} be its interpretation in $\Lambda(\Sigma_{derivations})$. We can further interpret c_{30} in $\Lambda(\Sigma_{trees})$ (resp. in $\Lambda(\Sigma_{logic})$) in order to have a

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

Table 14: $\mathscr{G}_{\textit{derived trees}}$ extension—control verbs

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

Table 15: $\Sigma_{\textit{logic}}$ and $\mathscr{G}_{\textit{sem.}}$ extension—control verbs

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

$$
\begin{aligned}
C_{30} &= C_{\textit{to sleep}}\ (C_{\textit{wants}}\ I_{\mathsf{S}}\ I_{\mathsf{VP}}\ C_{\textit{John}}) \\
&:=_{\textit{derivations}} c_{30} \\
c_{30} &= c_{\textit{to sleep}}\ (c_{\textit{wants}}\ I_{\mathsf{S}}\ I_{\mathsf{VP}}\ c_{\textit{John}}) \\
&:=_{\textit{derived trees}} \mathsf{S}_2\ (\mathsf{NP}_1\ \textit{John})\ (\mathsf{VP}_2\ (\mathsf{V}_1\ \textit{wants})\ (\mathsf{S}_2\ (\mathsf{NP}_1\ (\mathsf{PRO}_1\ \epsilon))\ (\mathsf{VP}_2\ (\mathsf{V}_1\ \textit{to})\ (\mathsf{VP}_1\ \textit{sleep})))) \\
&:=_{\textit{sem.}} \textbf{want john}\ (\textbf{sleep john})
\end{aligned}
\tag{31}
$$

7.3 Feature Sharing and Semantic Computation

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

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

However, (Vijay-Shanker and Joshi 1988, p. 718) already noticed that " ... 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. While the typing exerts control over the admissible derivation structures, the associated computations are managed using interpretations, to compute the derived trees as well as the logical formulas.

8 Derivation Trees and Semantic Interpretations

Looking at Fig. 18 (p. 24), 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 'rue-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. This approach, already described in TAG or Synchronous TAG (Abeillé and Schabes 1989; Shieber and Schabes 1990; Abeillé 1995), also works with the ACG encoding of TAG (Kobele 2012).

We illustrate this with (32). Figure 31 presents the initial trees that allow us to analyze (32) either as an idiomatic expression, meaning "to die", with the initial tree $\alpha_{\textit{kicked the bucket}}$, or 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 $\alpha_{\textit{bucket}}$. In both cases, the derived tree is the same (Fig. 32(a)). However, the derivation trees differ, as Fig. 32(b) and Fig. 32(c) show.

(32) John kicked the bucket

According to the lexicons of Tables 16 and 17, (33), (34), (35), and (36) hold. They show that the two terms $C_{32(b)}$ and $C_{32(c)}$ have the same interpretations as derived tree by $\mathscr{G}_{derived\ trees}$. ACG example 8 also shows that parsing (32) yields two terms in $\Lambda(\Sigma_{TAG})$. But they have two different interpretations as logical formulas by the ACG $\mathscr{G}_{sem.}$.

$$\begin{aligned} C_{32(b)} &= C_{\textit{kicked}}\ I_{\mathsf{S}}\ I_{\mathsf{VP}}\ C_{\textit{John}}\ (C_{\textit{bucket}}\ C_{\textit{the}}) \\ C_{32(c)} &= C_{\textit{kicked the bucket}}\ I_{\mathsf{S}}\ I_{\mathsf{VP}}\ C_{\textit{John}} \end{aligned} \tag{33}$$

$$\begin{aligned} \mathscr{G}_{TAG}(C_{32(b)}) &= \gamma_{\textit{kicked}}\ I_{\mathsf{S}}\ I_{\mathsf{VP}}\ \gamma_{\textit{John}}\ (\gamma_{\textit{bucket}}\ \gamma_{\textit{the}}) \\ \mathscr{G}_{TAG}(C_{32(c)}) &= \gamma_{\textit{kicked the bucket}}\ I_{\mathsf{S}}\ I_{\mathsf{VP}}\ \gamma_{\textit{John}} \end{aligned} \tag{34}$$

$$\begin{aligned} \mathscr{G}_{derived\ trees} \circ \mathscr{G}_{TAG}(C_{32(b)}) &= \mathsf{S}_2\ (\mathsf{NP}_1\ \textit{John})\ (\mathsf{VP}_2\ (\mathsf{V}_1\ \textit{kicked})\ (\mathsf{NP}_2\ (\mathsf{Det}_1\ \textit{the})\ (\mathsf{N}_1\ \textit{bucket}))) \\ &= \mathscr{G}_{derived\ trees} \circ \mathscr{G}_{TAG}(C_{32(c)}) \end{aligned} \tag{35}$$

$$\begin{aligned} \mathscr{G}_{sem.}(C_{32(b)}) &= \exists!\, x.(\textbf{bucket}\ x) \wedge (\textbf{kick john}\ x) \\ \mathscr{G}_{sem.}(C_{32(c)}) &= \textbf{die john} \end{aligned} \tag{36}$$

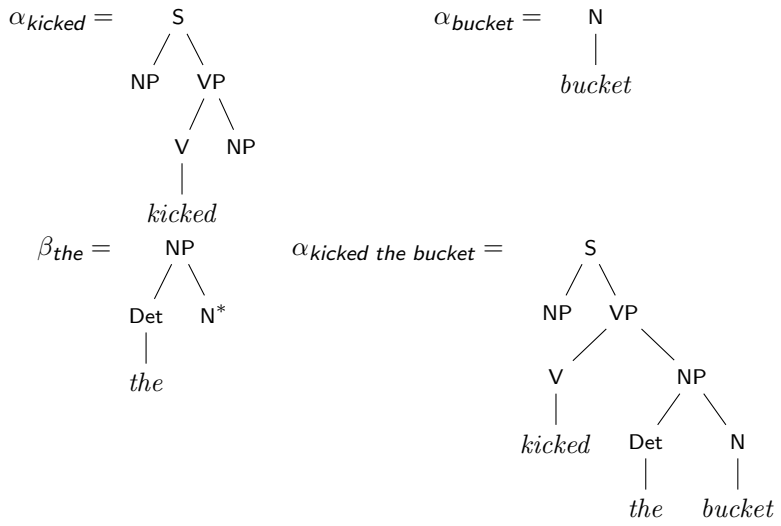
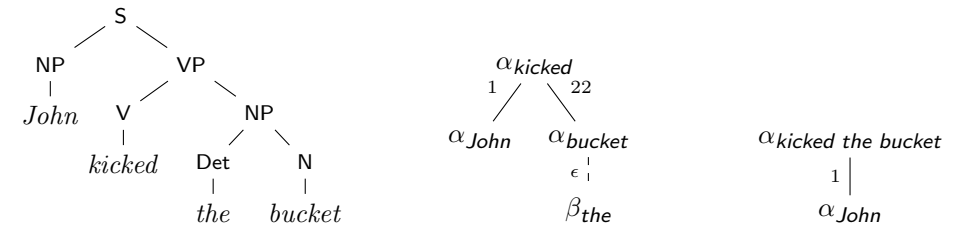


Figure 31: Elementary trees to analyze "kick the bucket"



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

Figure 32: Derived tree and derivation trees for *John kicked the bucket*

	Terms of $\Lambda(\Sigma_{trees})$	Corresponding TAG elementary tree
γ_{kicked}	$= \lambda^{o} S\ a\ s\ o.S\ (\mathsf{S}_2\ s\ (a\ (\mathsf{VP}_2\ (\mathsf{V}_1\ kicked)\ o)))$ $: (\tau \multimap \tau) \multimap (\tau \multimap \tau) \multimap \tau \multimap \tau \multimap \tau$	α_{kicked}
γ_{bucket}	$= \lambda^{o} d.d\ (\mathsf{N}_1\ bucket)$ $: (\tau \multimap \tau) \multimap \tau$	α_{bucket}
γ_{the}	$= \lambda^{o} n.\mathsf{NP}_2\ (\mathsf{Det}_1\ the)\ n$ $: \tau \multimap \tau$	β_{the}
$\gamma_{kicked\ the\ bucket}$	$= \lambda^{o} S\ a\ s.S\ (\mathsf{S}_2\ s\ (a\ (\mathsf{VP}_2\ (\mathsf{V}_1\ kicked)$ $(\mathsf{NP}_2\ (\mathsf{Det}_1\ the)(\mathsf{N}_1\ bucket)))))$ $: (\tau \multimap \tau) \multimap (\tau \multimap \tau) \multimap \tau \multimap \tau$	$\alpha_{kicked\ the\ bucket}$

Table 16: TAG elementary tree encoding as $\Lambda(\Sigma_{trees})$ terms

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

Table 17: Constants of $\Sigma_{\textit{derivations}}$ (resp. Σ_{TAG}) and their interpretation by $\mathscr{G}_{\textit{derived trees}}$ and $\mathscr{G}_{\textit{sem.}}$ (resp. $\mathscr{G}_{TAG}$)

---

yield_TAG **parse** John + kicked + the + bucket : S;
yield_TAG **parse** John + kicked + the + bucket : S;
An antecedent by yield_TAG in TAG is:
C_kicked_the_bucket I_s I_vp C_John : S
Do you want to look for another solution?
y/yes
n/no
a/all
(Default: yes):

An antecedent by yield_TAG in TAG is:
C_kicked I_s I_vp C_John (C_bucket C_the) : S
Do you want to look for another solution?
y/yes
n/no
a/all
(Default: yes):

No other possible value
#

ACG example 8: Parsing idiomatic expressions

8.2 Subordinate Conjunctions

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

In the case of subordinate conjunctions as in (37), it is the subordinate clause that adjoins on the matrix clause and uses it as argument, as Fig. 33 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.

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

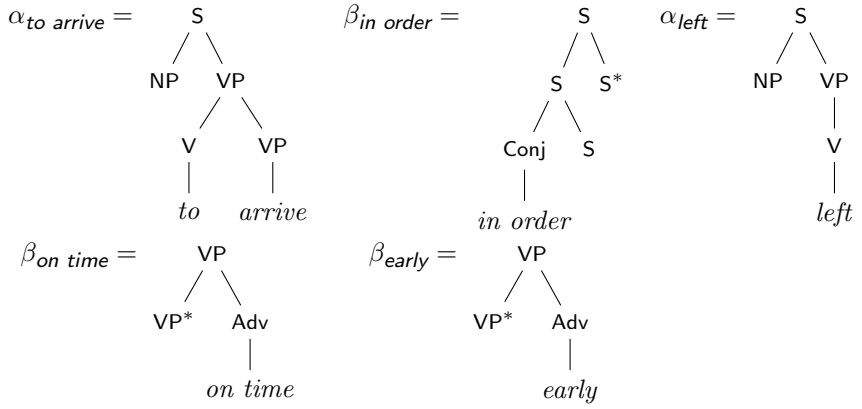


Figure 33: Auxiliary trees for subordinate conjuncts

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

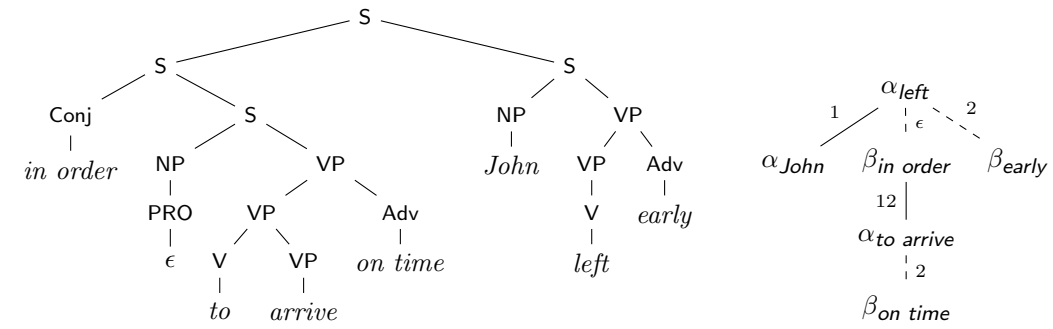
- there is no need for an adjunction on a S_n node of a term in $\Lambda(\Sigma_{trees})$ to be the image of a term (in $\Lambda(\Sigma_{derivations})$) of type S. We already used this feature;
- there is no need for an *actual node* in the derived tree to allow for an adjunction.

In order to implement the solution, terms for verbs such as c_{left} in Table 18 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 $c_{in\ order}$. 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 occur, we use the $I_{\mathsf{NP} \multimap \mathsf{S}}$ constant which is interpreted (syntactically and semantically) as the identity.

We can observe in $\mathscr{G}_{derived\ trees}(c_{left}) = \gamma_{left}$ how the subordinate clause is inserted. The latter correspond to the *sub* parameter in γ_{left} in Table 19. 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

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

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



(a) Derived tree (b) Derivation tree

Figure 34: Derived tree and derivation trees for *John kicked the bucket*

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 $\gamma_{\textit{in order}}$: the *sub* argument corresponds to the infinitive subordinate clause to be substituted in $\beta_{\textit{in order}}$, while the *matrix* argument corresponds to the matrix clause into which it adjoins and to which the *subj* argument is given as the subterm (*matrix subj*) shows.

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

	Terms of $\Lambda(\Sigma_{trees})$	Corresponding TAG elementary tree
$\gamma_{\textit{left}}$	$= \lambda^{o} S\ sub\ a\ subj\ obj.$ $S\ ((sub\ (\lambda^{o} s'.\mathsf{S}_2\ s'\ (a\ (\mathsf{VP}_2\ (\mathsf{V}_1\ kicked)\ obj))))\ subj)$ $: (\tau \multimap \tau) \multimap ((\tau \multimap \tau) \multimap (\tau \multimap \tau)) \multimap (\tau \multimap \tau) \multimap \tau \multimap \tau \multimap \tau$	$\alpha_{\textit{left}}$
$\gamma_{\textit{to arrive}}$	$= \lambda^{o} a\ s.\mathsf{S}_2\ (\mathsf{NP}_1\ s)\ (a\ (\mathsf{VP}_2\ (\mathsf{V}_1\ to)\ (\mathsf{VP}_1\ arrive)))$ $: (\tau \multimap \tau) \multimap \tau \multimap \tau$	$\alpha_{\textit{to arrive}}$
$\gamma_{\textit{in order}}$	$= \lambda^{o} sub\ matrix\ subj.$ $\mathsf{S}_2\ (\mathsf{S}_2\ (\mathsf{Conj}_1\ in\ order)\ (sub\ (\mathsf{PRO}_1\ \epsilon)))\ (matrix\ subj)$ $: (\tau \multimap \tau) \multimap (\tau \multimap \tau) \multimap \tau \multimap \tau$	$\beta_{\textit{in order}}$

Table 19: TAG elementary tree encoding as $\Lambda(\Sigma_{trees})$ terms

With the lexicon of Tables 18 and 19, we can build terms that correspond to the derivation and derived trees of Fig. 34 as (38), (39), (40) show.[20] We compute the semantic interpretation as

[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 $C_{\textit{left}}$ and $c_{\textit{left}}$ are modified here.

in (41).

$$C_{34(b)} = C_{\textit{left}}\ I_{\mathsf{S}}\ (C_{\textit{in order}}\ (C_{\textit{to arrive}}\ C_{\textit{on time}}))\ C_{\textit{early}}\ C_{\textit{John}} \tag{38}$$

$$\mathscr{G}_{TAG}(C_{34(b)}) = c_{\textit{left}}\ I_{\mathsf{S}}\ (c_{\textit{in order}}\ (c_{\textit{to arrive}}\ c_{\textit{on time}}))\ c_{\textit{early}}\ c_{\textit{John}} \tag{39}$$

$$\begin{aligned}\mathscr{G}_{\textit{derived trees}} \circ \mathscr{G}_{TAG}(C_{34(b)}) = {} & \mathsf{S}_2\ (\mathsf{S}_2\ (\mathsf{Conj}_1\ \textit{in order})\ (\mathsf{S}_2\ (\mathsf{NP}_1\ (\mathsf{PRO}_1\ \epsilon)) \\ & (\mathsf{VP}_2\ (\mathsf{VP}_2\ (\mathsf{V}_1 \textit{to})\ (\mathsf{VP}_1\ \textit{arrive}))(\mathsf{Adv}_1\ \textit{on time})))) \\ & (\mathsf{S}_2\ (\mathsf{NP}_1\ \textit{John})\ (\mathsf{VP}_2\ (\mathsf{VP}_1\ (\mathsf{V}_1\ \textit{left}))\ (\mathsf{Adv}_1\ \textit{early})))\end{aligned} \tag{40}$$

$$\mathscr{G}_{sem.} \circ \mathscr{G}_{TAG}(C_{34(b)}) = \mathbf{goal}\ (\mathbf{on\_time}\ (\mathbf{arrive}\ \mathbf{john}))\ (\mathbf{early}\ (\mathbf{leave}\ \mathbf{john})) \tag{41}$$

Because $\mathscr{G}_{TAG}$ still is second-order, parsing is available as ACG example 9 shows. The first command parses the logical term $t_9^{\log}$ (see (42)) to get the term $t_9 : \mathsf{S}$ of $\mathcal{A}(\mathscr{G}_{sem.} \circ \mathscr{G}_{TAG})$.[21] The second command yields the same term as term of $\mathcal{A}(\mathscr{G}_{yield} \circ \mathscr{G}_{\textit{derived trees}} \circ \mathscr{G}_{TAG})$ while parsing the string t_9^{string}.

$$t_9^{\log} = \exists x.(\mathbf{man}\ x) \wedge (\mathbf{goal}(\mathbf{on\_time}(\mathbf{arrive}\ x))(\mathbf{early}(\mathbf{leave}\ x))) \tag{42}$$

$$t_9^{\text{string}} = \textit{in} + \textit{order} + \textit{to} + \textit{arrive} + \textit{on} + \textit{time} + \textit{a} + \textit{man} + \textit{left} + \textit{early} \tag{43}$$

$$t_9 = C_{\textit{left}}\ I_{\mathsf{s}}(C_{\textit{in order}}\ (C_{\textit{to arrive}}\ C_{\textit{on time}))}\ C_{\textit{early}}\ (C_{\textit{man}}\ C_{\textit{a}}) \tag{44}$$

8.3 Scope Ambiguity and Non-Functional Form-Meaning Relation

The phenomena we have modeled so far make use of derivation structures (either in $\Lambda(\Sigma_{\textit{derivations}})$ or in $\Lambda(\Sigma_{TAG})$) that are very close (homomorphic) to TAG derivation trees. As we can see on Fig. 18 (p. 24), the relation between TAG derivations as terms of $\Lambda(\Sigma_{TAG})$ and terms of $\Lambda(\Sigma_{\textit{logic}})$ was functional (encoded by the function $\mathscr{G}_{sem.} \circ \mathscr{G}_{TAG}$). The non-functional relation was between terms of $\Lambda(\Sigma_{\textit{trees}})$ and terms of $\Lambda(\Sigma_{\textit{logic}})$ (encoded by the function $\mathscr{G}_{sem.} \circ \mathscr{G}^{-1}_{\textit{derived trees}}$). So because there is two derivation trees for *John kicked the bucket*, there was 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, $\mathscr{G}_{\textit{co-sem.}}$, maps terms of $\Lambda(\Sigma_{CoTAG})$ to terms of $\Lambda(\Sigma_{\textit{logic}})$. It then provides a relation between $\Lambda(\Sigma_{TAG})$ and $\Lambda(\Sigma_{\textit{logic}})$ as $\mathscr{G}_{\textit{co-sem.}} \circ \mathscr{G}^{-1}_{CoTAG}$ as Fig. 35 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 $C_{\textit{everyone}} : \mathsf{NP}$ in Σ_{TAG}, we have a term $L_{\textit{everyone}} : (\mathsf{NP} \multimap \mathsf{S}) \multimap \mathsf{S}$ in Σ_{CoTAG} such that $\mathscr{G}_{CoTAG}(L_{\textit{everyone}}) = \lambda^{\circ} P.P\ C_{\textit{everyone}}$. More generally, whenever a term of type A occurring within a constituent of type B can take scope over this term, we associate to $C_{\textit{scoping}} : A_1 \multimap \ldots \multimap A_n \multimap A$ in Σ_{TAG} a term $L_{\textit{scoping}} : A_1 \multimap \ldots \multimap A_n \multimap (A \multimap B) \multimap B$ in Σ_{CoTAG} such that $\mathscr{G}_{CoTAG}(L_{\textit{scoping}}) = \lambda^{\circ} x_1 \ldots x_n . \lambda^{\circ} P.P\ (C_{\textit{scoping}}\ x_1 \cdots x_n)$. For other lexical

[21] A second solution is available using the constant $C_{\textit{some}}$ instead of $C_{\textit{a}}$ as these two terms have the same semantic representation. This second solution will of course not be available when parsing from the string as their surface form differ.

```
# sem_TAG parse Ex x. (man x) & (goal (on_time (arrive x)) (early (leave x))) : S;
sem_TAG parse Ex x. (man x) & (goal (on_time (arrive x)) (early (leave x))) : S;
An antecedent by sem_TAG in TAG is:
C_left I_s (C_in_order (C_to_arrive C_on_time)) C_early (C_man C_a) : S
Do you want to look for another solution?
        y/yes
        n/no
        a/all
(Default: yes):

An antecedent by sem_TAG in TAG is:
C_left I_s (C_in_order (C_to_arrive C_on_time)) C_early (C_man C_some) : S
Do you want to look for another solution?
        y/yes
        n/no
        a/all
(Default: yes):

No other possible value
# yield_TAG parse in + order + to + arrive + on + time + a +man + left + early :S;
yield_TAG parse in + order + to + arrive + on + time + a +man + left + early :S;
An antecedent by yield_TAG in TAG is:
C_left I_s (C_in_order (C_to_arrive C_on_time)) C_early (C_man C_a) : S
Do you want to look for another solution?
        y/yes
        n/no
        a/all
(Default: yes):

No other possible value
#
```

ACG example 9: Parsing and generation with subordinate clauses

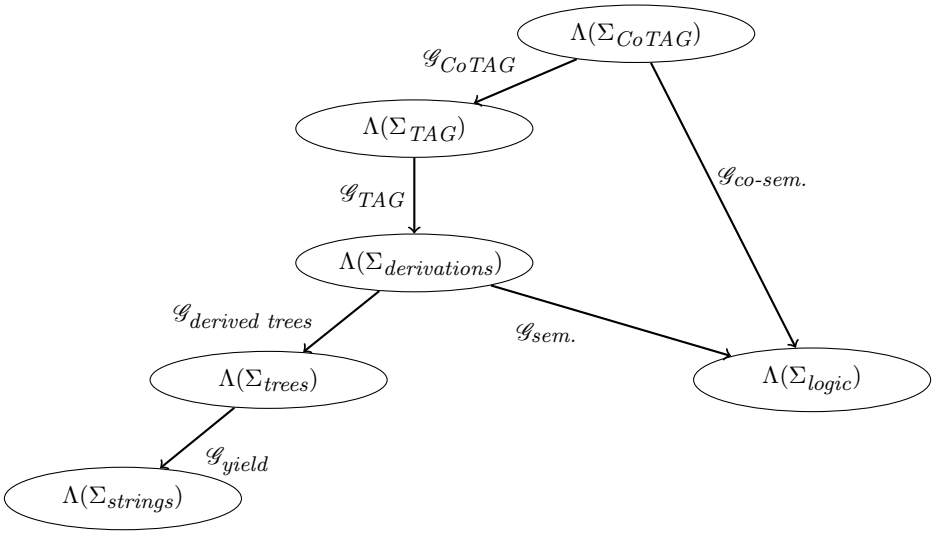


Figure 35: ACG composition for TAG and CoTAG

items $C_{\mathsf{lex.\ item}} : \alpha$, we have $L_{\mathsf{lex.item}} : \alpha$ such that $\mathscr{G}_{CoTAG}(L_{\mathsf{lex.item}}) = C_{\mathsf{lex.\ item}}$. And for any atomic type A, $\mathscr{G}_{CoTAG}(A) = A$.

Table 20 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}_{22} and L^{ows}_{22} as defined in (45) and (46) respectively, we indeed have $\mathscr{G}_{CoTAG}(L^{sws}_{22}) = \mathscr{G}_{CoTAG}(L^{ows}_{22})$, i.e., two different abstract terms of $\Lambda(\Sigma_{CoTAG})$ that are mapped to the same term of $\Lambda(\Sigma_{TAG})$ (TAG derivation tree).

$$\begin{aligned} L^{sws}_{22} &= (L_{\mathsf{man}}\ L_{\mathsf{every}})\ (\lambda^{\circ}x.(L_{\mathsf{woman}}\ L_{\mathsf{some}})\ (\lambda^{\circ}y.L_{\mathsf{loves}}\ I_{\mathsf{S}}\ I_{\mathsf{VP}}\ x\ y)) \\ \mathscr{G}_{CoTAG}(L^{sws}_{22}) &= (\lambda^{\circ}P.P\ (C_{\mathsf{man}}\ C_{\mathsf{every}}))\ (\lambda^{\circ}x.(\lambda^{\circ}P.P\ (C_{\mathsf{woman}}\ C_{\mathsf{some}})) \\ &\qquad\qquad (\lambda^{\circ}y.C_{\mathsf{loves}}\ I_{\mathsf{S}}\ I_{\mathsf{VP}}\ x\ y)) \\ &= C_{\mathsf{loves}}\ I_{\mathsf{S}}\ I_{\mathsf{VP}}\ (C_{\mathsf{man}}\ C_{\mathsf{every}})\ (C_{\mathsf{woman}}\ C_{\mathsf{some}}) \end{aligned} \tag{45}$$

$$\begin{aligned} L^{ows}_{22} &= (L_{\mathsf{woman}}\ L_{\mathsf{some}})\ (\lambda^{\circ}y.(L_{\mathsf{man}}\ L_{\mathsf{every}})\ (\lambda^{\circ}x.L_{\mathsf{loves}}\ I_{\mathsf{S}}\ I_{\mathsf{VP}}\ x\ y)) \\ \mathscr{G}_{CoTAG}(L^{ows}_{22}) &= (\lambda^{\circ}P.P\ (C_{\mathsf{woman}}\ C_{\mathsf{some}}))\ (\lambda^{\circ}y.(\lambda^{\circ}P.P\ (C_{\mathsf{man}}\ C_{\mathsf{every}})) \\ &\qquad\qquad (\lambda^{\circ}x.C_{\mathsf{loves}}\ I_{\mathsf{S}}\ I_{\mathsf{VP}}\ x\ y)) \\ &= C_{\mathsf{loves}}\ I_{\mathsf{S}}\ I_{\mathsf{VP}}\ (C_{\mathsf{man}}\ C_{\mathsf{every}})\ (C_{\mathsf{woman}}\ C_{\mathsf{some}}) \end{aligned} \tag{46}$$

In order to get two semantic interpretations from the two abstract terms of $\Lambda(\Sigma_{CoTAG})$, we need to directly provide them with a semantic lexicon. For if we keep on interpreting them through Σ_{TAG} and $\Sigma_{derivations}$, because the two terms L^{sws}_{22} and L^{ows}_{22} are interpreted as a single term in $\Lambda(\Sigma_{TAG})$, we still would get a single interpretation. In other words, we do not want the diagram of Fig. 35 to commute.

Table 21 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}_{22} and L^{ows}_{22} by $\mathscr{G}_{co\text{-}sem.}$. (47) shows these two terms are mapped to two logical formulas corresponding to the subject wide scope reading

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

Table 20: Constants of Σ_{CoTAG} and their interpretation by $\mathscr{G}_{CoTAG}$

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}	$:=_{co\text{-}sem.} \lambda det.det\ \mathbf{woman}$

Table 21: Constants of Σ_{CoTAG} and their interpretation by $\mathscr{G}_{co\text{-}sem.}$

on the one hand, and to the object wide scope reading on the other hand.

$$\mathscr{G}_{co\text{-}sem.}(L_{22}^{sws}) = \forall x.(\mathbf{man}\ x) \Rightarrow (\exists x'.(\mathbf{woman}\ x') \wedge (\mathbf{love}\ x\ x')) \tag{47}$$

$$\mathscr{G}_{co\text{-}sem.}(L_{22}^{ows}) = \exists x.(\mathbf{woman}\ x) \wedge (\forall x'.(\mathbf{man}\ x') \Rightarrow (\mathbf{love}\ x'\ x)) \tag{48}$$

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 problems 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 Conclusion

We showed that the TAG operations of substitution and adjunction can be represented as function application. This underlies the ACG encoding of TAG. The ACG framework comes with a modularity property that results from the two notions of composition between grammars it provides.

We then used the TAG into ACG encoding and the grammar composition to give a semantic construction process for TAG that relies on the derivations. We also used the grammar composition to restrict the derivations to actual TAG derivations, using a second-order ACG. This allows us to apply the ACG parsing results and make the grammar reversible so that both parsing and syntactic realization are available. Finally, we showed how the representation of derivations as terms of an ACG can be used to syntactically and semantically model phenomena such as idioms or subordinate conjunctions. We also showed that together with ACG composition, we can bring account from type-logical frameworks into TAG, such as the modeling of scope ambiguities.

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.

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.

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)*. New Haven, CT, pp. 135–142. URL: http://semanticsarchive.net/barker/barker-cosubstitution.pdf.

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. URL: http://www.kahane.fr/component/docman/doc_download/65-mtt-tag1998.html?Itemid=126.

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. URL: http://link.springer.com/content/pdf/10.1007/978-3-642-20169-1_5.pdf.

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.

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)*. Università di Venezia, pp. 145–150. URL: http://www.loria.fr/equipes/calligramme/acg/publications/2002-tag+6.pdf.

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. SALT: 16.1/1791. URL: http://elanguage.net/journals/index.php/salt/article/view/16.1/1791.

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, 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%3A1024137719751.

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. URL: http://jos.oxfordjournals.org/content/23/1/55.abstract.

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.

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)*. URL: http://www.kahane.fr/component/docman/doc_download/59-motsqu-tag2000.html?Itemid=126.

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)*. Università di Venezia, pp. 127–136. URL: http://www.sfs.uni-tuebingen.de/~lk/papers/tag+6.ps.gz.

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)*, pp. 108–116. URL: http://alpage.inria.fr/tagplus11/lib/exe/fetch.php?media=tag_1113.pdf.

Kallmeyer, Laura and Maribel Romero (2004). "LTAG Semantics with Semantic Unification". In: *Seventh International Workshop on Tree Adjoining Grammar and Related Formalisms - TAG+7*, pp. 155–162. URL: http://www.sfs.uni-tuebingen.de/~lk/papers/kallmrom-tag+7.pdf.

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 (2008). "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)*. Tuebingen, Germany, pp. 49–56. URL: http://tagplus9.cs.sfu.ca/papers/Kanazawa.pdf.

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. URL: http://www.springerlink.com/content/p644605651088uv6/.

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

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://www.dei.unipd.it/~fgrammar/fg07/fg07preproc/0005/paper_10.pdf.

Kasper, Robert, Bernd Kiefer, and Klaus Netter (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. 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+)*. Paris, France, pp. 153–161. URL: http://home.uchicago.edu/~gkobele/files/Kobele12Idioms.pdf.

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 (2012). "Decomposing TAG Parsing Algorithms Using Simple Algebraizations". In: *Proceedings of the 11th International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+)*. Paris, France, pp. 135–143. URL: http://www.ida.liu.se/~marku61/pdf/koller2012decomposition.pdf.

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

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.

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 (**montague74b**).

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.

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*. HAL open archive: hal-00749297.

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. 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. 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 (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.

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. 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. ACL anthology: P89-1002.

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. 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*.