Tree Adjoining Grammar, Semantic Calculi and Labelling Invariants

Claire Gardent
CNRS/LORIA
Nancy (France)

November 24, 2006

Abstract

Recently, proposals have been made to combine Tree Adjoining Grammar (TAG) with either Glue or Flat Semantic Representation Languages. In this paper, we additionally specify the combination of TAG with λ-based semantics and compare the three approaches. We observe several invariants and suggest that semantic construction in TAG is governed by a set of general principles that can be used to facilitate the design and development of TAGs integrating a semantic dimension.

1 A brief introduction to TAG

We use a unification based version of Lexicalised TAG namely, Feature-based TAG. A Feature-based TAG (FTAG, [**?**]) consists of a set of (auxiliary or initial) elementary trees and of two tree composition operations: substitution and adjunction. Substitution inserts a tree onto the leaf node of another tree[1] while adjunction inserts an auxiliary tree into a derived tree (i.e., either an elementary tree or a tree resulting from the combination of a derived tree with an elementary tree by means either of adjunction or of substitution).

In an FTAG, each tree node is associated with two feature structures called `top` and `bottom`. The `top` feature structure encodes information that needs to be percolated up the tree should an adjunction take place whilst the `bottom` feature structure encodes information that remains local to the node at which adjunction takes place. During derivation, the unifications listed in Figure 1 take place.

[1]These leaf nodes must be marked for substitution and are graphically distinguished by a down-arrow.

- The adjunction at some node X with `top` features t_X and `bottom` features b_X, of an auxiliary tree with root `top` features r and foot `bottom` features f entails the unification of t_X with r and of b_X with f.
- The substitution at some node X with `top` features t_X and `bottom` features b_X, of a tree with root `top` features t and root `bottom` features b entails the unification of t_X with t and of b_X with b.
- At the end of a derivation, the `top` and `bottom` features of all nodes in the derived tree are unified.

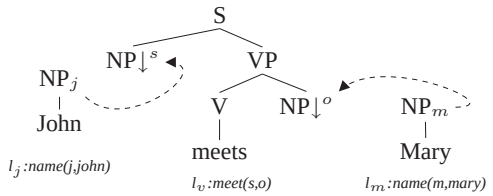
Figure 1: Unifications in FTAG

2 Combining TAG with three distinct semantic calculi

We now show how TAG can be combined with the three types of semantic calculi mentioned aboved: flat semantics, glue semantics and λ semantics. The first two cases have already been discussed in the literature, the third (λ semantics) is a new proposal.

2.1 Flat semantics

The flat semantics approach to semantic construction in TAG was first presented in [**?**] and further elaborated in [**?**]. It works as follows.



$\Rightarrow$ l_j*:name(j,john),* l_m*:name(m,mary),* l_v*:meet(j,m)*

Figure 2: Flat Semantics for "John meets Mary"

Each elementary tree is associated with a flat semantic representation. For instance, in Figure 2[2], the trees for *John*, *meets* and *Mary* are associated with the semantics l_j*:name(j,john),* l_m*:name(m,mary)* and l_v*:meet(s,o)* respectively.

[2]Here and in what follows, a downarrow (↓) indicates a substitution node and C^x/C_x abbreviate a node with category C and a top/bottom feature structure including the feature-value pair { **index :** x}.

Importantly, the arguments of a semantic functor are represented by unification variables which occur both in the semantic representation of this functor and on some nodes of the associated syntactic tree. For instance in Figure 2, the two semantic indices s and o occurring in the semantic representation of *meet* also occur on the subject and the object substitution nodes of the associated elementary tree.

The value of these arguments is then determined by the unifications resulting from adjunction and substitution[3]. For instance, the semantic indices s and o in the tree for *meet* are unified by substitution with the semantic indices labelling the root nodes of the trees for *John* and *Mary* respectively. As a result, the semantics of *John meets Mary* is:

$$l_j$$*:name(j,john),*$l_j m$*:name(m,mary),*l_v*:meet(j,m)*

Note that although semantic information is here integrated into TAG elementary trees, nothing hinges on this. Indeed as [**?**, **?**] have shown, the semantic information needed to guide semantic construction can be automatically extracted from the elementary trees and the unifications required by semantic construction can be reconstructed after parsing on the basis of the derivation forest. In other words, although for readibility we integrate here semantic information into TAG elementary trees, the approach remains within TAG as it can be reconstructed using a purely syntactic TAG provided the corresponding semantic information has been stored and the unifications corresponding to the TAG operations are computed on the basis of the derivation forest.XS

2.2 Glue semantics

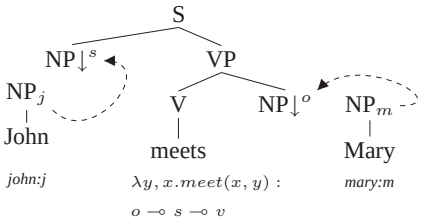
In [**?**], TAG is combined with glue semantics. For lack of space, we present here a slightly simplified version of their proposal where in particular we omit their treatment of what they call "external arguments".

In the Glue Semantics approach, TAG elementary trees are associated with so called *meaning constructors* consisting of a glue- and of a meaning-part. The meaning part is a λ-term whilst the glue part is a Linear Logic expression which specifies how the meaning of the functor arguments combines with that of the functor to determine the meaning of the whole.

As in the flat semantics approach, meanings and trees are related via variables[4] in that the glue part of a meaning constructor contains variables which also occur in the tree. This is illustrated in Figure **??** above where e.g., the index s labelling

[3] As [**?**, **?**] show, these unifications can be performed either during or after parsing.

[4] Whilst [**?**] postulate explicit identifications between tree and meaning constructor variables, we rely here instead on the unifications performed by the substitution and the adjunction operations.



$$\Rightarrow john : j, mary : m, \lambda x, y.meet(x, y) : m \multimap j \multimap v$$

Figure 3: Glue Semantics for *John meets Mary*

the subject node of the tree for *meets* also occur in the glue part of the associated meaning constructor (i.e., $\lambda y, x$.*meet(x,y)* : $o \multimap s \multimap v$).

The meaning constructors derived during parsing are then taken as premises to a linear logic derivation. Following the Curry-Howard isomorphism, a meaning is computed in parallel.

$$\dfrac{\dfrac{\textit{mary} : m \qquad \lambda x, y.\textit{meet(x,y)} : m \multimap j \multimap v}{\lambda x.\textit{meet}(x, \textit{mary}) : j \multimap v} \qquad \textit{john} : j}{\textit{meet}(\textit{john}, \textit{mary}) : v}$$

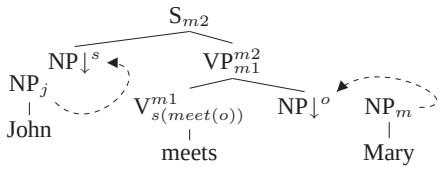
2.3 Lambda semantics

We now show how to combine TAG with a Montague style semantics using a similar process as in the flat and the glue semantic approaches. To start with, each elementary trees is associated with a λ-term and with a proxy (formally, a constant) for that λ-term (cf. Figure **??**). The proxies are then used to specify the way in which the λ-terms associated with the elementary trees should combine. In particular, each elementary tree associated with a semantic functor will be labelled with an *application pattern* indicating how the λ-terms of the arguments should combine with that of the functor. Formally, this application pattern is a feature structure which we abbreviate using a more intuitive linear notation. For instance, in Figure **??** below, the term $s(meet(o))$ abbreviates the feature structure :

$$\begin{bmatrix} \text{functor} & s \\ \text{arg1} & \begin{bmatrix} \text{functor} & \textit{meet} \\ \text{arg1} & o \end{bmatrix} \end{bmatrix}$$

This application pattern indicates that the subject semantics represented by the proxy s applies to the verb semantics with proxy *meet* applied to the object seman-

tics with proxy o[5].



Proxy	*λ-term*
j	$\rightsquigarrow \lambda P.P(j)$
m	$\rightsquigarrow \lambda P.P(m)$
meet	$\rightsquigarrow \lambda O \lambda x.O(\lambda y.meet(x,y))$

Figure 4: λ-Semantics for *John meets Mary*

As in the other approaches, the value of the application pattern is determined by the unifications triggered by the substitution and adjunction operations. Thus in the derivation sketched in Figure **??**, $s(meet(o))$ is instantiated by substitution to $j(meet(m))$ while the post derivation top and bottom unifications unifies m_2 with m_1 and $j(meet(m))$.

The application pattern labelling the root node is then used to retrieve the correct λ-terms and build the complex λ-term representing the sentence meaning:

$$\begin{array}{ll} & m_2 = m_1 = \textit{j(meet(m))} \\ \equiv & \lambda P.P(j)(\lambda O \lambda x.O(\lambda y.meet(x,y))(\lambda P.P(m))) \\ \equiv & \textit{meet(j,m)} \end{array}$$

3 AV-Principles

The three approaches to TAG-based semantic construction just described share one characteristic namely:

Semantic construction is guided by unification variables.

In the flat semantic approach, these unification variables are the semantic indices of semantic functors; In the glue approach, they are part of the types that occur in the meaning constructors and guide the linear logic derivation; And in the λ semantics, they stand proxy for the λ-term to be combined.

Thus in all three cases, semantic construction is guided by unification variables which occur both in the elementary trees and in the semantic representations (flat

[5]Although the feature structures used to represent application terms are recursive, the remark made at the end of section 2.1 still holds so that the approach remains within TAG provided semantic information is extracted from the trees and handled in a post-parsing phase.

formula, meaning constructor or applicative pattern) associated with these trees. From now on, we refer to these unification variables as *assembler variables* (AV).

Formally and computationally, it would be nice if a second invariant was true namely if

> *the distribution of these assembler variables were the same across all three approaches.*

In what follows, we consider additional data and demonstrate that this is almost the case. Specifically, we show that the emerging differences are due on the one hand, to the nature of the assembler variables used (recursive for the λ-approach, atomic for the other two) and on the other hand, to the diverging treatment of scope ambiguity.

3.1 Predicate/Argument

Consider again the simple example we started with, namely *John meets Mary*. Suppose that we postulate the following Assembler Variable Principles (AV-principles) to account for the tree labelling necessary to capture Predicate/Argument relationships[6]

Let AV_b(n) (resp. AV_t(n)) denote the value of the assembler variable feature AV in the bottom (resp. top) feature structure of node n. Then each elementary tree τ must conform to the following principles:

Anchor Projection: Given the sequence of nodes $a, n_1 \ldots, n_m, ap$ from the anchor a to its maximal projection node ap, then: AV_t(a) = AV_b(n1), AV_t(n1) = AV_b(n2), ..., AV_t(n_m) = AV_b(ap) with AV_t(a) a new variable

Argument Labelling: For all argument node arg_i in τ, AV_t(arg_i) = x, x a new variable

ANCHOR PROJECTION requires that the anchor node projects its index upwards to its maximal projection. ARGUMENT LABELLING associates with each argument node a fresh assembler variable. An argument node is a node representing a syntactic argument e.g., a subject, an object, etc.

If we now decorate the elementary trees entering in the derivation of *John meets Mary* according to these principles, we obtain the trees given in Figure **??**.

The verb ($m_\perp, m_1, m_2$) and the NP labels (j, m) are imposed by the ANCHOR PROJECTION Principle. The argument labels (s, o) are fixed by the ARGUMENT LABELLING Principle.

[6]These principles as well as the idea to make them explicit were already present in [**?**]. We are interested here in identifying the basic differences and similarities between the AV-principles governing the three semantic calculi.

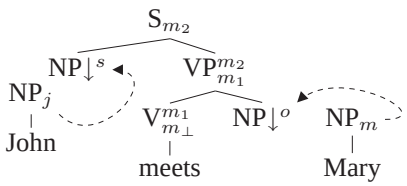


Figure 5: "John meets Mary"

Suppose further that the lexical semantics associated with *Jon, meeets* and *Mary* are as before namely:

FS: *name(j,john), name(m,mary), meet($m_\perp$,s,o)*

GS: *john:j, mary:m,* $\lambda y, x.meet(x, y) : o \multimap (s \multimap v) \equiv meet(john, mary) : v$

LS: : j $\rightsquigarrow \lambda P.P(j)$; m$\rightsquigarrow \lambda P.P(m)$; $m_\perp = j(meet(o))$, *meet* $\rightsquigarrow \lambda O \lambda x.O(\lambda y.meet(x, y))$

Then the AV-labelling shown in Figure **??** appropriately supports all three calculi (cf. section 2). To better illustrate this, we summarise below the unifications that result from substitution and for each approach, the resulting phrasal semantics.

Unifications: { *s = j, o = m,* $m_2 = m_1 = m_\perp$ }

FS: *name(j,john), name(m,mary), meet($m_\perp$,j,m)*

GS: *john:j, mary:m,* $\lambda y, x$*.meet(x,y):m* $\multimap (j \multimap v)$

LS: m_2 = *s(meet(o))* $= \lambda P.P(j)(\lambda O \lambda x.O(\lambda y.meet(x, y))(\lambda P.P(m)))$ = *meet(j,m)*

3.2 Modification

The three approaches to semantic construction can be divided into two classes depending on how the semantics of a derived tree is computed from the semantics of the lexical trees participating in its derivation. In the first case (glue and flat semantics), the semantics of a derived tree is the union of the semantics of the lexical trees entering the derivation modulo the unifications of the assembler variables contained in these semantics. By contrast, in the lambda semantics approach, the semantics of a derived tree is the application pattern of the tree root modulo unifications of AVs.

More generally, the difference is that whilst in the lambda based approach, the semantics of a tree is expressed by a *recursive term* (the application pattern of the tree root), in the other two approaches, the semantics is *flat* and the dependency between semantics parts is expressed either by labels (flat semantics) or by types (glue semantics).

This difference shows in the treatment of modification as follows. In the λ-based approach, ANCHOR PROJECTION suffices to compute the result semantics

as it ensures (together with the top and bottom unifications performed at each node, cf. Figure 1) that the application pattern of the main semantic functor is projected upward to the root node. In contrast, both in the glue and in the flat semantics approach, it is necessary to pass up indices and types so that they are available for eventual further binding.

For instance for the λ-based approach, the tree for *often* and the final derived tree can be as given in Figure **??**.

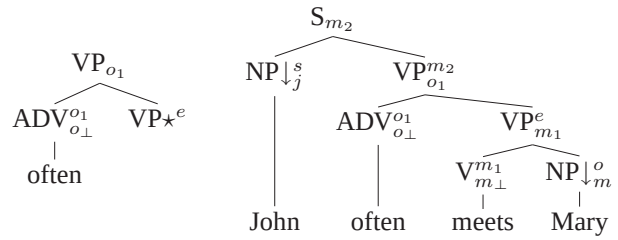


Figure 6: "John often meets Mary" (λ semantics)

The lexical semantics, the unifications resulting from parsing and the resulting phrasal semantics would then be:

Lex Sem: $m_\perp$ = *s(meet(o))*, meet $\rightsquigarrow \lambda O \lambda x.O(\lambda y.meet(x,y))$ $o_\perp$ = *often(e)*, often $\rightsquigarrow \lambda P.often(P)$

Unifications: { *s* = *j*, *o* = *m*, $m_\perp = m_1$ = *e* , $o_\perp = o_1 = m_2$ }

Result Sem: m_2 = *often(j(meet(m)))*

In contrast, both the glue and the flat semantic approach require the following additional principle:

Foot projection: In a modifier-type auxiliary tree with foot node f and root node r, then: L_t(f) = L_b(r)

This principle in essence ensures that the variable bound by the modifier is passed up the tree for eventual binding by another modifier (*John often meets Maria in the street*).

Given this additional principle, the tree for *often* and the final derived tree are as given in Figure **??**.

The associated AV unification, lexical semantics and the resulting semantics are as follows:

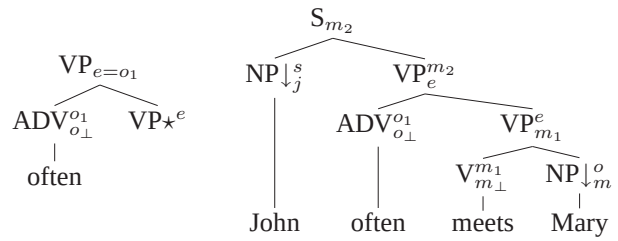


Figure 7: "John often meets Mary" (Glue and Flat Semantics)

Unifications: { *s = j, o = m,* $m_\perp = m_1 = m_2 = e$, $o_\perp = o_1 = m_2 = e$ }

FS: *name(j,john)+name(m,mary)+meet(*$m_\perp$*,s,o) +*
often(e)
⇒ *name(j,john), name(m,mary), meet(*$m_\perp$*,j,m), often(m*$\perp$*)*

GS: *john:j, mary:m,* $\lambda y, x$*.meet(x,y):*$o \multimap (s \multimap m_\perp)$,
λP*.often(P):*$e \multimap e$
⇒ *john:j, mary:m,* $\lambda y, x$*.meet(x,y):* $m \multimap (j \multimap m_\perp)$,
$\lambda P.often(P) : m_\perp \multimap m_\perp$

Summing up: in the flat/glue approach, a modifier needs to pass up the a-variable of its argument for an eventual further binding. This is ensured by the FOOT PROJECTION Principle. In the λ-based approach on the other hand, a modifier must pass up to the root the applicative term resulting from the application of this modifier to the applicative term associated by semantic composition to the constituent it modifies. This in turn is ensured by the ANCHOR PROJECTION Principle.

3.3 Control

To capture control phenomena, the flat semantics approach introduces an additional feature C whose value is constrained to be both the semantic index of the controller and that of the controllee [**?**, **?**]. We generalise this mechanism to all three approaches and posit the following additional principle:

Controller/Controllee: In the elementary tree associated with a control verb, the C feature of the foot node associated with the sentential argument is identified with its controller label.
$C_t(n_s) = L_t(n_c)$, with n_s the sentential argument node and n_c, the controller node.

In Figure **??**, the derivation for *John tries to meet Mary* illustrates this (the C value is given by the second a-variable occurring on the sentential argument node in the tree for *tries* and on the root node of the tree for *meet*) . Note that Foot Node Arguments are treated in the same way as Substitution Node Ones. That is, although in TAG, a sentential argument is usually associated with a foot node, such a node is not subject to the FOOT PROJECTION Principle (because it is not part of a modifier-type auxiliary tree).

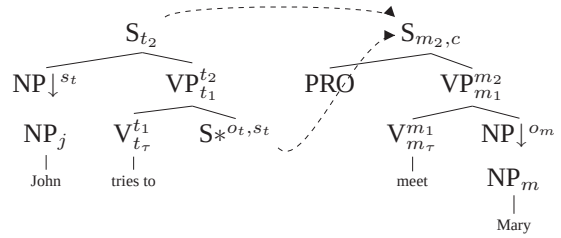


Figure 8: “John tries to meet Mary”

As the associated unifications, the lexical and the phrasal semantics given below illustrates, this AV-labelling supports all three approaches.

Unifications: $\{ s_t = j, o_t = m_2, c = s_t, o_m = m, t_\perp = t_1 = t_2, m_\perp = m_1 = m_2 \}$

FS: *name(j,john) + name(m,mary) + try($t_\perp$,s_t,o_t) + meet($m_\perp$,c,o_m)*
$\Rightarrow$ *name(j,john), name(m,mary), try($t_\perp$,j,$m_\perp$), meet($m_\perp$,j,m)*

GS: *john:j, mary:m,* $\lambda y, x$*.meet(x,y):*$o_m \multimap (c \multimap m_\perp)$, *λP,x.try(x,P):*$(c \multimap o_\perp) \multimap (s_t \multimap t_\perp)$
$\Rightarrow$ *john:j, mary:m,*$\lambda y, x$*.meet(x,y):*$m \multimap (j \multimap m_\perp)$, *λP,x.try(x,P):*$(j \multimap m_\perp) \multimap (j \multimap t_\perp)$

LS: : $m_\perp$ = *c(meet(o_m))+*$t_\perp$ = *s_t(try(o_t))*
$\Rightarrow$ AV(root) = *j(try(j(meet(m))))*

4 Conclusion

We have sketched a method for combining TAG with λ-based semantics and compared the resulting calculus for semantic construction with both a glue- and a flat-semantics approach. In so doing, we have identified the following common semantic principles:

Anchor projection: The anchor node projects its label up to its maximal projection.

Argument labelling: In trees associated with semantic functors, each argument node is labelled with a new a-variable.

Controller/Controllee: In trees associated with control verbs, the av of the controller is identified with the value of the C-feature occuring on the sentential argument node.

On the other hand, we have shown that the λ-approach differs from both the flat- and the glue-semantics approach in that it does not require the additional FOOT PROJECTION Principle. And although space restrictions do not allow us to do so here, it can also be shown that only the flat-semantics approach requires additional AV-principles for the treatment of scope ambiguity.

These observations should not be taken as "imperatives". There are many possible ways to encode semantics in a grammar; there are various ways of encoding scope ambiguity; and different syntactic encoding might result in different semantic encoding. Hence the principles identified in this paper are not necessarily true of all implementations. Nonetheless they suggest an important point namely, that:

> The labelling principles necessary to integrate semantic information into a Tree Adjoining Grammar are (i) limited in numbers and (ii) partially "reusable" across semantic approaches.

These two points have an obvious practical and theoretical impact. First and foremost, they suggest that the AV-labelling that is required to augment a syntactic TAG with semantic information is governed by *general principles* in the sense of Generalised Phrase Structure Grammar (GPSG) or more recently, head-Driven Phrase Structure Grammar (HPSG): an elementary TAG tree τ is well-formed iff (i) τ is a well-formed TAG tree and (ii) τ verifies the AV-Principles governing the specific type of semantics used. This in turn raises the question of how such principles could be enforced and we are currently investigating how such principles could be integrated in XMG, an expressive grammar formalism [**?**] for specifying and semi-automatically generating Tree-Based Grammar.

A second consequence concerns the mapping between different types of "semantic TAGs" (glue-, flat- or λ-based). Given that the three approaches have much in common, it should be relatively easy to develop and empirically compare each of the three approaches within a TAG framework.

References

[CLF01] Ann Copestake, Alex Lascarides, and Dan Flickinger. An algebra for semantic construction in constraint-based grammars. In *Proceedings of the*

39th Annual Meeting of the Association for Computational Linguistics, Toulouse, France, 2001.

[DRP04] Denys Duchier, Joseph Le Roux, and Yannick Parmentier. The metagrammar compiler: An nlp application with a multi-paradigm architecture. In *2nde confrence internationale Oz/Mozart (MOZ'2004)*, Charleroi, 2004.

[FvG01] Anette Frank and Josef van Genabith. Ll-based semantics for ltag - and what it teaches us about lfg and ltag. In *Proceedings of the LFG'01 Conference*, Hong Kong, 2001. CSLI Online Publications.

[GK03] Claire Gardent and Laura Kallmeyer. Semantic construction in ftag. In *Proceedings of the 10th meeting of the European Chapter of the Association for Computational Linguistics*, Budapest, Hungary, 2003.

[GP05] Claire Gardent and Yannick Parmentier. Large scale semantic construction for tree adjoining grammar. In *Proceedings of Logical Aspects in Computational Linguistics*, Bordeaux, France, 2005.

[KR04] Laura Kallmeyer and Maribel Romero. Ltag semantics with semantic unification. In *Proceedings of TAG+7*, pages 155–162, 2004.

[SW98] Matthew Stone and Bonnie Webber. Textual economy through closely coupled syntax and semantics. In *Proceedings of the Ninth International Workshop on Natural Language Generation*, pages 178–187, Niagara-on-the-Lake, Canada, 1998.

[VSJ88] K. Vijay-Shanker and A. K. Joshi. Feature structures based tree adjoining grammar. In *Proceedings of COLING*, pages 714–719, Budapest, 1988.