Large scale semantic construction for Tree Adjoining Grammars*

Claire Gardent[1] and Yannick Parmentier[2]

[1] CNRS, LORIA, Campus Scientifique BP 239, 54 Nancy
Claire.Gardent@loria.fr,
http://www.loria.fr/~gardent
[2] INRIA, LORIA, Campus Scientifique BP 239, 54 Nancy
Yannick.Parmentier@loria.fr

Résumé Although Tree Adjoining Grammars (TAG) are widely used for syntactic processing, there is to date no large scale TAG available which also supports semantic construction. In this paper, we present a highly factorised way of implementing a syntax/semantic interface in TAG. We then show how the resulting resource can be used to perform semantic construction either during or after derivation.

1 Introduction

Developing a Tree Adjoining Grammar which contains the information necessary for building the basic compositional semantics of sentences is a highly complex engineering task. To ensure consistency, ease of writing, of maintainance and of debugging, it is therefore important that this information be described at the appropriate level of abstraction. In the first part of this paper (sections 2 and 3), we show how to achieve a highly factorised integration of semantic information into a Tree Adjoining Grammar (TAG, [1]) using a particularly expressive grammar formalism recently developed by [2].

The second part of the paper shows how the resulting TAG can be used to support semantic construction that is, to associate the sentences generated by the grammar with a semantic representation. Contrary to other linguistic frameworks such as Lexical Functional Grammar, Head Driven Phrase Structure Grammar or Categorial Grammar, there is no clear consensus in TAG on how to perform semantic construction. This is because Tree Adjoining Grammar associates a derivation not with one, but with two structures namely, a derivation tree and a derived tree; and because it is unclear which of these two structures best supports semantic construction. As TAG elementary trees localise predicate-argument dependencies so that derivation trees resemble semantic dependency trees, the TAG derivation tree has long been taken to provide an appropriate basis for semantic construction. Nevertheless, it has repeatedly been shown that the derivation tree alone does not provide all the information needed to perform

* We would like to thank Benoit Crabbé, Denys Duchier, Djamé Seddah and Eric Villemonte de la Clergerie for many useful discussions on the themes discussed in this paper.

semantic construction in all possible cases [3,4,5]; and that information from the derived tree also has to be taken into account.

In the second part of this paper, we show how the semantic TAG described in the first part can be used to support two types of semantic construction processes both of them being based on the information contained in the derived tree. The first method follows traditional unification based grammar practice and performs semantic construction during parsing (section 4) while in the second, semantic construction is done after parsing on the basis of a derivation forest thereby benefiting from the structure sharing supported by such packed representations (section 5). The resulting framework lays the basis for a systematic exploration of the relative efficiency of these two semantic construction methods for TAGs .

2 A TAG with a unification-based syntax/semantics interface

In this section, we present the semantic TAG we use for semantic construction. Section 3 shows how to produce such a TAG on a large scale for a core fragment of French. Sections 4 and 5 show how to use it to perform semantic construction in two different ways.

2.1 Feature-based TAG

In the approach we present here, semantic representations are combined using unification. To this end, we use a unification based version of LTAG namely, Feature-based TAG. A Feature-based TAG (FTAG, [6]) 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[3] while adjunction (sketched in Fig. 1) 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` one encodes information that remains local to the node at which adjunction takes place. During derivation, the unifications listed in Figure 2 take place.

2.2 Semantic representation language and glue mechanism

When doing semantic construction, two main questions arise: the choice of the semantic representation language and that of the "glueing" mechanism used for putting semantic representations together. Mainly, semantic representations

3. These leaf nodes must be marked for substitution and are graphically distinguished by a downarrow.

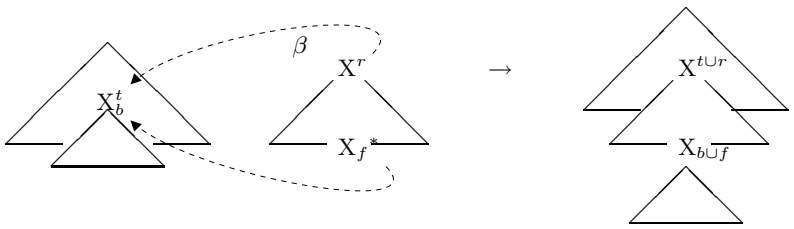


Fig. 1 –. *Adjunction in FTAG*

- 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.

Fig. 2 –. *Unifications in FTAG*

can be feature structures, lambda terms or terms of some underspecified logic whereas the available glueing mechanisms include unification, beta-reduction and linear logic.

The approach described here assumes a unification-based semantic construction process where semantic representations are flat semantic representations allowing for scope underspecification [7]. Importantly, the `semantic parameters` (that is, the semantic indices representing the missing arguments of the semantic functors) are represented by unification variables. As we shall see in the following section, the syntax/semantics interface is specified by the grammar in such a way that, as functors and arguments combine, semantic parameters are unified by the semantic construction process with the appropriate `semantic indices`.

For instance, the semantic representation for the semantic functor *every* and for its potential argument *cat* are as given in Example 1 and Example 2 where atoms starting with a capital letter are unification variables.

Example 1. $l_0 : \forall(X,h_1,h_2), h_1 \geq L_{restr}, h_2 \geq L_{scope}$
Example 2. $l_c : cat(Y)$

Combining these two representations using the grammar described in the sequel will yield the representation for *every cat* given in Example 3 where in particular, the restriction handle L_{restr} in the representation of *every* is unified with the label l_c in the representation for *cat* and the individual variable X in the representation of *every* with the variable Y in that of *cat*.

Example 3. $l_0 : \forall(X,h_1,h_2),h_1 \geq l_c,h_2 \geq L_{scope},l_c : cat(X)$

For details about the semantic representation language used, we refer the reader to [5]. Note however that the choice of a particular semantic representation language and of a particular glueing mechanism is not particularly important here. Indeed the proposed approach could be applied to other semantic representation languages using some other glueing mechanism.

2.3 Modelling the relation between syntax and semantics

Syntax specifies which syntactic constituent provides the semantic argument for which semantic functor. To specify this mapping between syntax and semantics, (i) each elementary tree in the grammar is associated with a semantic representation of the type sketched above and (ii) the appropriate nodes of the elementary trees are decorated with semantic indices or parameters.

More precisely, the substitution nodes of the tree associated with a semantic functor will be associated with semantic parameters whilst root nodes and certain adjunction nodes will be labelled with semantic indices. As trees are combined, semantic parameters and semantic indices are unified by the FTAG unification mechanism thus specifying which semantic index provides the value for which semantic parameter. So for instance, the trees for *John, loves* and *Mary* will be as given in Figure 3. The tree for *loves* is associated with a semantic representation including the two semantic parameters x and y. These parameters also label the subject and the object substitution nodes of this tree. Conversely, the root node of the tree for *John* is labelled with the semantic index j. If the string parsed is *John loves Mary*, this tree will be substituted at the subject substitution node of the *loves* tree thus instantiating the semantic parameter x to j. And similarly, for the *Mary* tree.

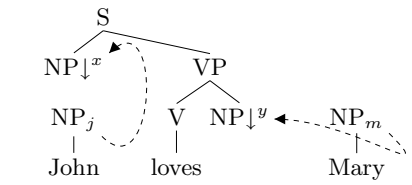


Fig. 3 –. *John loves Mary*

As we shall see in sections 4 and 5, a TAG equipped with the syntax/semantic interface just described can be used to construct semantic representations either during or after derivation. In the first case, the unification variables present both

on the tree nodes and in the semantic representations become instantiated as substitution and adjunction take place and the overall semantics of a sentence is the union of the semantic representations of the elementary trees entering in its derivation modulo unification. In the second case, a semantic lexicon is extracted from the grammar and used to do semantic construction on the basis of the derivation forest.

3 Grammar and Metagrammar: factorising the information

We now show how the metagrammar framework presented in [2] allows for a highly factorised specification of the mapping between syntax and semantics described in the preceding section. We start by presenting the grammar formalism used. We then show how it can be exploited to specify the syntax/semantics interface.

3.1 The metagrammar formalism

The metagrammar formalism presented in [2] can be seen as a generalisation of Shieber's PATR 2 language [8] which is expressive enough to encode (among others) Tree Adjoining Grammars. The goal of such languages is to provide a formalism that will allow a linguist to express her grammatical knowledge both directly and economically: the language must be expressive; it must also allow for the factorisation of redundant information.

As space restrictions do not allow for a complete specification of this formalism, we restrict ourselves here to an informal presentation of the concepts needed for the rest of this paper. The interested reader is invited to refer to [2,9,10] for more details.

The metagrammar formalism (called XMG for eXtensible MetaGrammar) used supports both syntactic and semantic information.

In the syntactic dimension, tree fragments can be described that will be combined with other fragments to produce complete trees. These tree fragments can be referred to by means of *abstractions* (also called *classes*). Similarly, in the semantic dimension partial flat semantic formulae can be defined and referred to by means of abstraction thus also allowing for the factorisation of semantic information.

Syntactic and semantic abstractions can be combined using one of three operations namely, *conjunction* (to accumulate information), *disjunction* (to introduce non-determinism, used for instance to express diathesis) and *inheritance*. This last operation is used to specialise a class by incrementally adding pieces of information to a parent class. In our concrete syntax, conjunction, disjunction and inheritance are represented by `;`, `|`, and `import` respectively.

Finally, variables can be shared between classes in two main distinct ways. In the first case, the shared variables belong to classes linked by an inheritance

relation and the scope of these variables can be explicitely managed using import and export declarations. In the second case, the shared variables belong to distinct inheritance chunks and sharing is made possible by a naming mechanism called `interfaces` which allow the global naming of a given value. For instance, in the class `Subj` below, the node `X` is named `sujNode` in the interface `*= [sujNode=X]` .

```
class Subj
declare ?X
{ <syn> { node [cat=s]
             node X [cat=n]
        } *= [sujNode=X]
}
```

The scope of an interface feature is global to its parent branch(es) in the hierarchy. As the next section will illustrate, the value of an interface feature can be shared by any other class by means of explicit variable sharing.

3.2 Specifying the syntax/semantics interface

The main issue when developping a large scale semantic TAG is the correct specification of the mapping between syntax and semantics (cf. section 2.3). in [5], we define this mapping for a serie of syntactico-semantic constructions which are known to be problematic for TAG. Here however, we are concerned with the problem of how to design a *large scale* semantic TAG efficiently and economically. In this respect, verbs or more generally, semantic functors are of particular interest as they represent the bulk of the possible variations. We therefore concentrate on verbs and show how to specify the syntax/semantic interface for their various basic subcategorisation frames (transitive, intransitive, etc.), their various possible argument realisations (e.g., cliticisation, extraction, ommission) and their argument redistributions (active, passive, middle voice, impersonnal passive, etc.). Due to space restrictions, other types of syntactico-semantic constructions, although they can be handled by the grammar formalism used, will not be discussed here.

As was illustrated in section 2.3, the specification of the syntax/semantics interface consists in appropriately defining the mapping between grammatical functions (subject, object, etc.) and thematic roles (e.g., agent, patient or more neutrally, arg1, arg2). For instance, in an active mood sentence with two nominal arguments, the subject NP is mapped to the first semantic argument (arg_1) and the object to the second (arg_2) whereas in a passive mood sentence, the inverse occurs so that the subject NP maps to arg_2 and the object to arg_1.

In a TAG, a word is associated with the set of trees reflecting the range of syntactic configurations this word can occur in. For a verb (and more generally, for any type of syntactic functor), this set can be quite big. For instance, in the grammar for French developed by B. Crabbé [9], a transitive verb with nominal arguments is associated with 153 trees each describing a distinct possible

syntactic environment for such a verb. More generally, Crabbé's core grammar for French totals roughly 3 500 trees for the verb fragment thereby covering 35 basic subcategorisation frames.

Clearly, the specification of the syntax/semantics interface needs to be factorised. We do not want to specify and maintain it for each of the 3 500 trees. To extend Crabbé's grammar with the syntax/semantics interface sketched in the preceding section, we proceed as follows:

1. While the semantic indices labelling the tree nodes are all values of an `idx` feature, they are also assigned a global name reflecting the grammatical function fulfilled by the node they label. For instance, the index x on the subject node of the active tree for *loves* in Figure 3 will be assigned the global name `subjectI`.
2. Similarly, the semantic indices occuring in the semantic representations are assigned a global name reflecting their thematic role. For instance, the first semantic argument of a binary relation is named `arg1`.
3. Finally, the mapping between grammatical functions and thematic roles is specified by coindexing the values of grammatical and thematic indices. For instance, in an active mode sentence tree, the value of `subjectI` will be coindexed with that of `arg1`.

We now show how this works in more detail. We start by showing how the syntactic information is factorised in Crabbé's grammar. We then go on to show how it can be extended with the syntax/semantics interface described in the previous section.

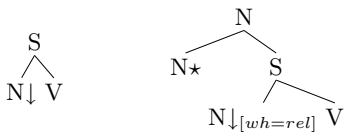
The syntax In Crabbé's metagrammar [9,10], the syntactic information associated with a TAG elementary tree is factorised along the following three dimensions.

First, grammatical function classes are defined which describe their structural properties. For instance, the `Subject` class is defined by the disjunction:

```
class Subject{
  CanonicalSubject | RelativeSubject | whSubject | ...
}
```

where each subclass (`CanonicalSubject`, etc.) is associated with the appropriate structural description [4] e.g.,

`CanonicalSubject` `RelativeSubject`



4. To improve readibility, we represent tree descriptions using graphics rather than logical formula. The precise tree language supported by XMG is described in [2].

Next, alternations are defined in terms of grammatical functions and verbal morphology. For instance, the active and passive alternations for transitive verbs are defined by the following conjunctions of classes:

```
class n0Vn1Active{
  Subject ; Object ; activeVerbMorphology
}
class n0Vn1Passive{
  Subject ; CAgent; passiveVerbMorphology
}
```

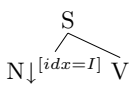
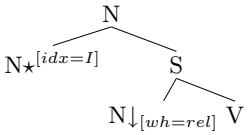
where `passiveVerbMorphology, activeVerbMorphology, Object` and `CAgent` are abstractions over the TAG structural objects associated with these linguistic notions.

Finally, the set of elementary trees associated with a given subcategorisation frame (e.g., `n0Vn1`) is defined by the disjunction of its alternations e.g.,

```
class n0Vn1{
  noVn1Active | noVn1Passive | n0Vn1dePassive | n0Vn1ShortPassive |
  n0Vn1ImpersonalPassive | n0Vn1middle
}
```

Augmenting the metagrammar with semantic information. As mentioned above, augmenting the metagrammar with semantic information involves three steps.

First, the semantic indices labelling the tree nodes are named according to their grammatical function. For instance, the index labelling the subject node of the active tree for a transitive verb will be named `subjectI`. As shown below, this is done using an interface constraint: for each possible realisation of a subject, the value of the semantic index labelling the subject node is named `subjectI` by means of the interface constraint. In practice, the naming is done for a total of 12 grammatical functions (Subject, Object, SententialSubject, SententialCObject, SententialDeObject,SententialAObject, SententialInterrogative, Iobject, CAgent, Oblique, Locative, Genitive) and 56 realisations.

Class Name	`CanonicalSubject`	`RelativeSubject`
Structural description		
Interface constraint	`subjectI` $= I$	`subjectI` $= I$

Second, the semantic indices occuring in the semantic representations are named according to their thematic role. For instance, the first semantic argument of a binary relation is named `arg1`. This naming is again enforced by an interface constraint making the value globally accessible under that name .

```
class binaryRel
```

```
declare !L0 ?Rel ?E ?I1 !L1 ?I2 !L2
      {
        <sem>{
                L0:Rel(E) ;  L1:arg1(E,I1) ;  L2:arg2(E,I2)
              }
              *=[rel=Rel,evt=E,arg1=I1,arg2=I2]
         }
```

Third, the mapping between grammatical functions and thematic roles is specified by coindexing the relevant values.

Consider the class `n0Vn1` for instance, which describes the set of syntactico-semantic configurations associated in a TAG (for French) with verbs taking two nominal arguments. This set covers the configurations possible for the active mode, the long passive mode, the passive in *de*, the short passive, the impersonal passive and the middle form. For each of these modes, there are several possible configurations depending on how the arguments are realised (i.e., whether the subject/object/agent/etc. is canonical, cliticised, extracted, etc.) so that in total the class `n0Vn1` includes 153 trees. The mapping between syntax and semantics for these 153 trees is realised in the metagrammar by the labelling described above and by the following class definition:

```
class n0Vn1{
               binaryRel*=[evt=E,arg1=X,arg2=Y] ;
              { n0Vn1Active*=[subjectI=X,objectI=Y,vbI=E]
                | n0Vn1Passive*=[subjectI=Y,cagentI=X,vbI=E]
                | n0Vn1dePassive*=[subjectI=Y,genitiveI=X,vbI=E]
                | n0Vn1ShortPassive*=[subjectI=Y,vbI=E]
                | n0Vn1ImpersonalPassive*=[objectI=X,vbI=E]
                | n0Vn1middle*=[objectI=Y,vbI=E]
               }
}
```

That is, the set of syntactico-semantic configurations associated with the `n0Vn1` verbs is defined as consisting of (i) a binary semantic relation, (ii) trees realising the different possible verbal modes and grammatical functions realisations and (iii) a mapping between the semantic parameters of the binary relations and the semantic indices labelling the nodes of the trees. Typically, the first semantic parameter is identified with the subject semantic index in the active mode and with the object in the passive mode. If the verb is impersonnal passive (*il est arrivé trois femmes*), this first parameter is identified with the semantic index of the object, etc.

In sum, the XMG formalism allows for a direct encoding of the linguistic notions necessary to specify the syntax/semantics interface: naming of the semantic indices labelling the tree nodes realising a given grammatical function, naming of the semantic parameters according to their thematic role and coindexing of the two types of indices. This expressivity in turn permits a clear and economical encoding: the factorisation is high in that the relevant notions need

only be encoded once but are used by many distinct classes. For instance, the labelling of the subject index is done once but is used by all of the 35 verb classes defined in our current TAG for French (since all verb classes make use of the subject class for their definition).

Using this encoding, a core TAG for French can be developped which encodes the semantic information necessary to support semantic construction. We now show how this information can be used in two different ways to compute the compositional semantics of a sentence during (or after) parsing.

4 Derived tree and semantic construction

A first, simple way to construct semantic representations based on the semantic TAG described in the preceding section is to build these during derivation. Such an approach can be integrated in a TAG parser by simply associating the semantic representation of an elementary tree with the anchor node of that tree (cf. Figure 4). Since an anchor node never merges with any other node, there can be no conflict and the semantic representation remains untouched modulo the unification its indices can undergo via the coindexing with the tree nodes indices. The semantics of a derived tree is then the union of the values of the **semf** features present in this tree after unifications have taken place. For instance, if the trees of Figure 4 are combined to parse the string *Jean aime beaucoup Marie*, the resulting derived tree will be as given in Figure 5.

As mentioned above, the semantics associated with this tree is the union of the **semf** values after the TAG imposed unifications (cf. section 2) have taken place namely [5]

```
{!e:aime(e,j,m), jean(j), marie(m), beaucoup(e)}
```

5 Derivation forest, semantic lexicon and semantic construction

As [5] shows, the semantic construction process described in the previous section accounts for data which an approach based on the derivation tree does not. However, the approach is open to two potential problems. First as mentioned in [11], the semantic representations included in the elementary and derived tree imply an infinite number of labels and individual variables so that in contrast to standard FTAG, the formalism is theoretically no longer equivalent to TAG. In practice of course, real sentences have finite lengths and so an upper bound could be specified which makes the set of feature values finite. Another difficulty however, is that the semantic information labelling the trees might decrease the amount of sharing in a tabular parsing approach and thereby decrease parsing efficiency.

5. The `!` stands for the existential quantifier.

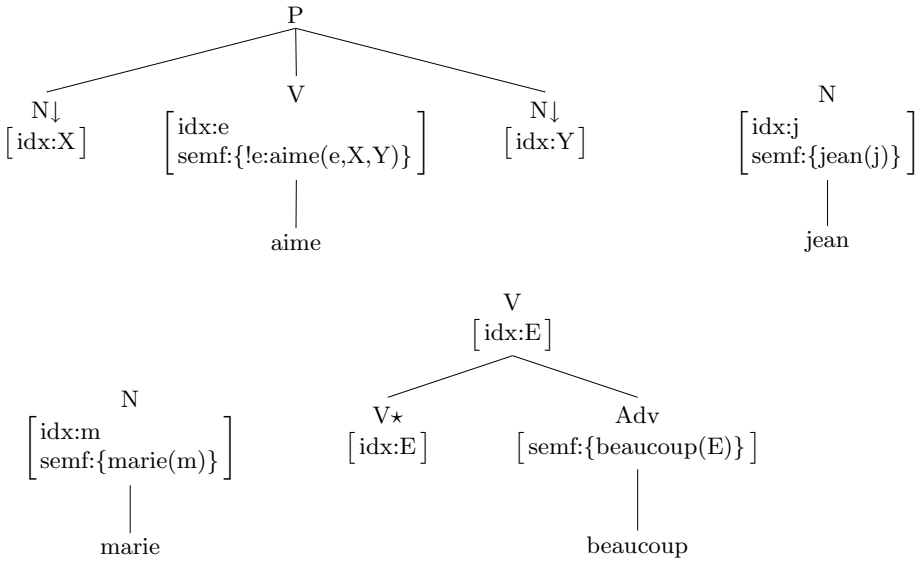


Fig. 4 –. *TAG elementary trees with semantics included*

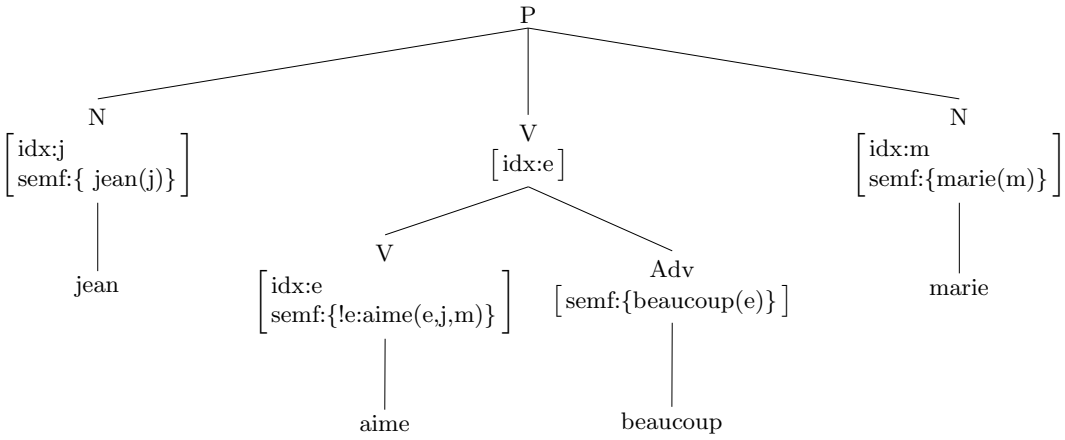


Fig. 5 –. *TAG derived tree with semantics included*

To explore whether the second of these two objections is a real problem, we thus investigate a second way to do semantic construction where in essence, the semantic information is extracted from the TAG and used after parsing to reconstruct on the basis of the derivation tree the semantic representation of the sentence under consideration. This second way of doing semantic construction was first presented in [11]. We show here how it can be implemented on the basis of a standard TAG parser and of the semantic TAG produced by the XMG (cf. section 2). We start by giving a simplified example illustrating the workings of the approach. We then indicate first, how the required semantic lexicon can be automatically extracted from the semantic TAG described in section 2 and second, how semantic construction proceeds.

5.1 A simple example

A TAG derivation tree records how the elementary trees used to build a derived tree are put together using the two combining operations permitted by TAG namely, adjunction and substitution. Formally, the nodes of such a tree are labelled with tree names and its edges with a pair $\langle$ Op, Id $\rangle$ where Op denotes the combining operation used to combine the trees labelling the vertices of the edge and Id identifies the node at which this operation takes place.

Now suppose that parsing the sentence *Jean court* yields the unique derivation tree pictured in Figure 6.

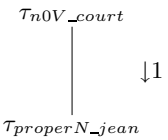


Fig. 6 –. *Derivation tree for Jean court*

And suppose further that the semantic lexicon extracted from the semantic TAG for *Jean* and *court* is as follows:

TreeName	*n0V*	TreeName	*properN*
Lemma	court	Lemma	Jean
SemRepr	!e:court(e,X)	SemRepr	jean(j)
ANodes	2.bot = [idx=e]	ANodes	
SNodes	1.top = [idx=X]	SNodes	
Root	0.bot = [idx=e]	Root	0.bot = [idx=j]

That is, the semantic information extracted from each elementary tree and stored in the semantic lexicon consists of the name of that tree, a record of the lemma anchoring that tree, the semantic representation associated with that tree and a record of the semantic information associated with the nodes (substitution nodes, nodes where adjunction can take place, root and foot nodes) of that tree.

Semantic construction then proceeds by traversing the derivation tree, collecting the lexical semantics associated in the semantic lexicon with each tree present in the derivation tree and performing the unifications imposed by a TAG derivation (cf. Figure 2). In this case, collecting the lexical semantics associated with the two trees occurring in the derivation tree yields:

{!e:court(e,X), jean(j)}

Two unification steps are furthermore involved. The first follows from the substitution of $\tau_{properN\_jean}$ at node 1 of τ_{n0V_court} which entails the unification of the feature structures of the root node of $\tau_{properN\_jean}$ with those of node 1 in τ_{n0V_court}:

$$1.\text{top} = 0.\text{top} = [\text{idx=X}]$$
$$1.\text{bot} = 0.\text{bot} = [\text{idx=j}]$$

The second unification step follows from the requirement that at the end of a TAG derivation the top and bottom feature structures of all nodes in the derived tree be unified. This requirement entails in particular the following unification:

$$1.\text{top} = 1.\text{bot}$$
$$0.\text{top} = 0.\text{bot}$$

As a result [idx=X] unifies with [idx=j] and the overall semantics of *Jean court* becomes:

{!e:court(e,j), jean(j)}

5.2 Extracting a semantic lexicon from the semantic TAG

As the above example illustrates, the information required to perform semantic construction on the basis of a derivation forest consists of: a treename, a lemma, a semantic representation and four sets of path equations relating derived tree nodes with the semantic information labelling these nodes. One set of equations pertains to substitution nodes, another to nodes where adjunction can take place, the third to the root node and the fourth to the foot node if any.

That is, for each elementary tree present in the grammar, an entry is added to the semantic lexicon which contains the above information. Note further that the TAG used for parsing does not need to include any semantic information: all the semantic processing is done after parsing has taken place and relies only on the information contained in a (purely syntactic) derivation forest and in the semantic lexicon.

To automatically extract the required semantic lexicon from the semantic TAG G described in section 2, we proceed as follows:

1. For each tree T in G, all the nodes of T are numbered with their gorn addresses so that the nodes of the resulting grammar GG are then labelled both with semantic indices and with a gorn address.

2. For each tree T in GG:
   (a) create a tree ST by erasing on all nodes of T the semantic information (if any) labelling that node. Call the resulting purely syntactic grammar SG
   (b) create an entry in the semantic lexicon which contains: the tree name, the semantic representation associated by the metagrammar with this tree, the gorn addresses and the semantic information labelling the tree nodes

5.3 Computing semantic representations

As [12] shows, computing semantic representations from a parse forest is a natural way to deal with the combinatorial explosion that can result from enumerating all the readings of a given sentence: by doing semantic construction on the basis of the parse forest rather than the derivation trees, these shared syntactic constituents that have a single reading can also be shared during semantic construction. When combined with the use of an underspecified semantic representation language, such an approach allows for a large amount of structure sharing thereby increasing efficiency.

We now show how the semantic lexicon which, as shown in section 5.2 can be automatically extracted from the semantic TAG described in section 2, can be used in conjunction with a derivation forest to construct semantic representations.

A derivation forest is a compact representation of the derivation trees resulting from a sentence parse. It can be represented either by an and-or graph or by a context free grammar and its precise format may vary depending on the degree of sharing required [13]. Here we assume a CFG format where rules are of the form:

$$\begin{array}{ll} \mathit{DTNodeId} :: \mathit{ElTreeId} & \leftarrow (\mathit{DTNode/Op.Node})^{+} \\ \mathit{ElTreeId} :: \mathit{Lemma.TreeName} & \end{array}$$

with *DTNodeId, DTNode* identifying nodes in the derivation tree, *ElTreeId* identifying the elementary tree labelling a derivation tree node, *Op* being either *s* for *substitution* or *a* for *adjunction* and *Node* specifying the node in the elementary tree at which *Op* takes place.

To perform semantic construction, we simply traverse the derivation forest top-down, tabulating the constituents found and checking before constructing an item that it is not already included in the table built so far. For a given derivation tree in the parse forest, semantic construction is performed by a recursive descent through the tree as follows.

To construct the semantics *Sem* of a derivation tree with root *DTNodeId* given the parse forest rule $\mathit{DTNodeId} :: \mathit{ElTreeId} \leftarrow \mathit{Dtrs}$ **do**

Lemma.TreeName ← *terminal(DTNodeId)*
HeadSem ← *lexSem(Lemma.TreeName)*
SemDtrs ← *dtrsSem(HeadSem,Dtrs)*
Sem ← *HeadSem + SemDtrs*

where *terminal* is a procedure mapping each derivation tree node to the terminal node it directly or indirectly rewrites as within the parse forest; *lexSem* is a function retrieving from the semantic lexicon described in the preceding section, the lexical semantics associated with a given ⟨ *Lemma, TreeName* ⟩ pair ; *dtrsSem* is a procedure (described below) constructing the semantic representation of the daughters of a rule given the head semantics of its lhs ; and + denotes the operation accumulating the semantic representations being built. The *dtrsSem* procedure is defined as follows.

To construct the semantic representation *Sem* of the daughters *DTNodeId/Op.NodeId | ODtrs* of a rule given the head semantics *HeadSem* of its lhs, **do**

Lemma.TreeName ← *terminal(DTNodeId)*
HeadSemD1 ← *lexSem(Lemma.TreeName)*
tagUnify(HeadSem,HeadSemD1)
semODtrs ← *dtrsSem(HeadSem,ODtrs)*
Sem ← *HeadSemD1 + semODtrs*

where *tagUnify* performs the unification operations imposed on TAG derivations (cf. Figure 2) on the node labels provided by the semantic lexicon described in the previous section.

6 Conclusion

The proposal described in this paper is partially implemented. A core TAG for French is available which extends the syntactic TAG described in [9,10] to include semantic information as described in sections 3 and 2. Semantic construction during derivation is currently being implemented whilst semantic construction after derivation has been implemented using the above grammar, an XSLT style sheet to extract the semantic lexicon and a prolog module to perform semantic construction on the basis of this semantic lexicon and of the derivation forest produced by Eric de la Clergerie's Dyalog TAG parser.

The resulting framework thus supports the comparative evaluation of the two semantic construction procedures for TAG as well as the development and testing of large scale semantic TAGs for French. Future work will focus on comparing the relative efficiency of these two semantic construction procedures; extending the grammar to include further types of alternations and in particular those described in [14] and the LADL tables; and experimenting with different semantic representation languages and glueing mechanisms.

Références

1. Joshi, A.K., Schabes, Y.: Tree-Adjoning Grammars. In Rozenberg, G., Salomaa, A., eds.: Handbook of Formal Languages. Springer (1997) 69–123
2. Duchier, D., Le Roux, J., Parmentier, Y.: The metagrammar compiler : An nlp application with a multi-paradigm architecture. In: Second International Mozart/Oz Conference - MOZ 2004, Charleroi, Belgique. (2004)
3. Frank, A., van Genabith, J.: GlueTag. Linear Logic based Semantics for LTAG. In Butt, M., King, T.H., eds.: Proceedings of the LFG01 Conference, Hong Kong (2001)
4. Kallmeyer, L.: Using an Enriched TAG Derivation Structure as Basis for Semantics. In: Proceedings of TAG+6 Workshop, Venice (2002) 127 – 136
5. Gardent, C., Kallmeyer, L.: Semantic construction in ftag. In: Proceedings of the 10th meeting of the European Chapter of the Association for Computational Linguistics, Budapest, Hungary (2003)
6. Vijay-Shanker, K., Joshi, A.: Feature structures based tree adjoining grammars. In: Proceedings of COLING, Budapest, Hungary (1988) 714–719
7. Copestake, A., Lascarides, A., Flickinger, D.: 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)
8. Shieber, S.: An Introduction to Unification-based Approaches to Grammar. CSLI Lecture Notes (1986)
9. Crabbé, B., Duchier, D.: Metagrammar redux. In: International Workshop on Constraint Solving and Language Processing - CSLP 2004, Copenhagen. (2004)
10. Crabbé, B.: Grammatical development with XMG. Submitted to LACL05 (2005)
11. Kallmeyer, L., Romero, M.: Ltag semantics with semantic unification. In: Proceedings of the 7th International Workshop on Tree Adjoining Grammar and Related Formalisms, Vancouver, BC, Canada (2004) 155–162
12. Schiehlen, M.: Semantic construction from parse forests. In: Proceedings of the 16th International Conference on Computational Linguistics, Copenhagen (1996)
13. Alonso, M.A., Villemonte de la Clergerie, E., Diaz, V.J., Vilares, M.: 1. In: Relating Tabular Parsing Algorithms for LIG and TAG. Kluwer Academic Publishers (2002) to appear, revised notes of a paper for IWPT2000.
14. Saint-Dizier, P.: Alternation and verb semantic classes for french: Analysis and class formation. In: Predicative forms in natural language and in lexical knowledge bases. Kluwer Academic Publishers (1999)