A New Metagrammar Compiler

B. Gaiffe, B. Crabbé, and A. Roussanaly
Loria

1. Why a new metagrammar compiler ?

Writing a TAG grammar manually is known to be a non trivial task (Abeillé 00; Doran et al. 94; Doran et al. 00; VS et al. 92). For that purpose, (Candito 96) has suggested a grammatical framework that allows linguists to describe the syntactic properties of a language at a higher level of abstraction . Given a set of classes, each of these containing a partial tree description, the compiler outputs a set of tree schemata.

In order to work on the organization of a syntactic lexicon, we needed an almost equivalent tool that would produce feature structures together with the tree schematas. Before developping a new tool, we criticized Candito's work (Candito 99) in considering the two following drawbacks:

1. the algorithm is closely linked to the specific linguistic description. As her analysis focusses on the study of verbal trees, the structuration is badly adapted to the description of non verbal units ;
2. the current implementation of the compiler is not flexible enough to be easily adapted to other input/output formats.

In the remainder of the paper we describe our tool and discuss two possible ways to implement a metagrammar for french verbs.

2. Design of our tool

In this section, we present the characteristics of the tool we implemented. Our compiler ressembles the one described in (Candito 96) and (Candito 99). We thus present the latter first in order to emphasize the differences between her proposition and ours.

2.1. Topological factorisation

In (VS et al. 92), the authors propose to use a logic (precisely defined in (Rogers et al. 94)) which describes elementary trees of a TAG grammar so that the topological informations shared by trees is factorized in an inheritance hierarchy.

In practice, information concerning trees may be factorized according to different points of view quite independant from each other. For instance, subcategorization information leads to a rather natural hierarchy while realizations of syntactic functions lead to another hierarchy altogether. Therefore, attempts to describe both hierarchies in a unique inheritance lattice either leads to having to make a copy of one hierarchy at each leaf of the other, or if multiple inheritance is allowed, multiplying links between leaves of the hierarchies.

2.2. Marie-Hélène Candito's Compiler

(Candito 96) and (Candito 99) precisely explains the preceding point and advocates three independant, linguistically motivated hierarchies which she calls dimensions:

1. subcategorization (which she represents in terms of initial syntactic functions)
2. syntactic function redistributions (which lead to final syntactic functions)
3. final functions realisations

The underlying idea is that a linguist only describes these three hierarchies, and an automatic tool completes the inheritance graph by crossing final classes of dimension 1 with the final classes of dimension 2 and further crossing the result with the final classes of dimension 3.

However, not all final classes of dimension 1 are to be crossed with all final classes of dimension 2. For instance, intransitive verbs do not admit passive constructions. In the same way, the resulting crossed classes of dimensions 1 plus 2 have to be crossed only with those final classes of dimension 3 that realize a final function actually occuring in the crossed class.

The linguist has thus to give crossing conditions together with his hierarchies in order to constrain the crossing process.

The algorithm implemented by (Candito 99) is thus the following:

```
dim12 = empty set
for each final class c1 of dimension 1
   for each final class c2 of dimension 2 (compatible with c1)
        create c12 that inherits c1 and c2 and add it to dim12
   end for
end for
res = empty set
for each c12 in dim12
   resOfC12 = {c12}
   for each final function ff appearing in c12
      for each class c3 of dimension 3 that realizes ff
         build new classes with each element of resOfC12 and c3
         resOfC12 = these new classes
      end for
   end for
   res = res U resOfC12
end for
compute minimal referents for each element of res.
```

As this pseudo-algorithm makes clear, some constants that denote tree nodes are labelled by a final syntactic function. They are actually also labelled by an initial syntactic function, probably in order to keep track of the process. In dimension 2, most of the job done by classes consists in modifying the functional assignment[1]. Moreover, final functions are maintained unique in any description by an additional mechanism that equates constants bearing the same final function.

2.3. Our proposition

Our initial motivation for developping a new meta-grammar compiler had to do with the lexicon: the grammar compiled with Candito's tool is organised in tree families (as is XTAG (Doran et al. 00)) and lemmas are associated to families. The anchoring of a tree thus consists in computing the lemma and the morpho syntactic features associated to a word form, getting the families associated with the lemma and finally attempting to substitute the lemma with the associated morpho syntactic features in the tree.

Such a process may of course fail, either because the morpho-syntactic features do not match (consider a tree dedicated to an imperative form, together with an infinitive word form) , or because the features associated to the lemma do not match (some transitive verbs, for instance, do not accept a passive form).

Our starting idea was then to generate trees together with a feature structure that globally describes each tree, and Candito's tool did not seem to permit that.

Since we were implementing a new tool anyway, we gave ourselves some additional constraints:

- avoiding non monotonous mechanisms such as the modification of the final functions
- not limiting *a priori* the number of dimensions:
  - the third dimension is *de facto* a collection of dimensions dedicated to realizing each of the possible functions

1. Final functions are initialized to the initial function

– three dimensions is perhaps not a good choice for other categories than verbs, or for other languages than French, English or Italian. (See for instance (Gerdes 02))

As we intend to produce trees together with a feature structure, classes of the meta-grammar contain a feature structure that describe its content. It then seems natural that these feature structures get combined through unification along the inheritance lattice. This feature structure is then a good mechanism to avoid unwanted class crossings. The example we mention of intransitive verbs that do not accepts passive forms may simply be taken into account by means of an attribute **transitive** with values **minus** for an intransitive verb and **plus** for all passive classes.

The remaining problem is to find a mechanism that allowes classes to cross. In (Candito 99) compiler, this mechanism relies on the three dimensions, but we do not want to rely on a fixed number of dimensions.

We thus decided to make explicit the reason why classes are to be crossed, typically a class of dimension 1 has to be crossed with a class of dimension 2 because it **needs** redistribution of syntactic functions. Classes of dimension 2 have to be crossed with classes of ex-dimension 3 because they **need** that their final functions be realized. Conversely, a class of ex-dimension 3 may **provide** the realization of a subject, or an object, or whatever other function.

A class in our system is then described by:

- a name
- a set of super-classes
- a description (which is a feature structure)
- a set of needs (atomic symbols)
- a set of providings (atomic symbols)
- a formula describing trees

When a class c12 inherits two classes c1 and c2, the descriptions are unified (in case of failure, c12 is not created), the set of needs is the union of the two set of needs minus the union of the providings, the set of providings is also the union of the providings minus the union of the needs and the formula is the conjunction of the two formulas.

The crossing mechanism then consists in computing all balanced final classes, that is classes whose set of needs and providings are empty[2].

Finally, the formulas corresponding to balanced final classes are used to compute minimal referents (Rogers et al. 94) together with the description associated with the corresponding class.

3. A survey on the linguistic applications

We made some experiments on french verbs, conforming as much as possible to the analysis of (Abeillé 91; Candito 99). Therefore, we give an overview of the way we describe verbs using three 'dimensions'[3].

Two ways to generate a grammar are given in the following sections. The first approach puts the focus on the topology of the trees. While it allows to identify the nodes representing the predicate and its arguments in the tree, it actually suffers from a major drawback due to the monotonicity of the system. That is, once a node is asserted in the process of generating a tree, it cannot be removed. This is problematic if, for instance, one wants to describe an agentless passive as the ellipsis of the predicate's agent[4].

The second approach comes closer to the functional analysis introduced by (Candito 99). We do a topologically free reasoning upon arguments and functions until we have specified a complete final functional subcategorization frame. The main interest of this approach is that the functional component of the grammar is not anymore

2. In order to keep with an associative and commutative mechanism, a cancelled need as well as a cancelled providing is not allowed to appear again.
3. Formally speaking, it should be clear that there are no dimensions anymore.
4. However, there is a trick : one can set a 'kill' attribute to a node. It means that the node will be removed from the tree after its generation. The principle is that the removed node's children (if any) become the children of the removed node's parent (the root of the tree cannot be removed).

mixed with the topological one. We are able to represent the agentless passive quite easily. But here, the relationship between the tree structure and the logical arguments is lost. As a comparison, we do not specify any topological information into the classes that belong to the equivalent of Candito's first and second dimension. The whole analysis is driven by the feature structures describing the classes.

Both approaches share some essentials ideas[5]. (1) Each tree which is generated represents the realization of a logical predicate and its arguments. (2) The focus is put on the functional organization of the grammar. Following these assumptions we specify through three successive layers the trees that represents the syntactic realization of that predicate. Each of these layers performs a mapping as follows :

$$\begin{tikzcd}
\text{Predicate Structure} \arrow[d, Rightarrow, "\text{Dimension 1}"] \\
\text{Initial- Function} \arrow[d, Rightarrow, "\text{Dimension 2}"] \\
\text{Final-Function} \arrow[d, Rightarrow, "\text{Dimension 3}"] \\
\text{Surface-Realization}
\end{tikzcd}$$

- Dimension 1 : maps the predicate arguments to an initial functional subcategorization frame.
- Dimension 2 : maps the initial subcategorization frame to a final functional subcategorization frame.
- Dimension 3 : maps the final subcategorization frame to a tree structure.

3.1. A node driven strategy

Following (Abeillé 91; Candito 99) each tree which belongs to a family represents a predicative structure. The first dimension is dedicated to mapping initial functions to the predicate's argument positions. Here we define a set of classes which represents the arguments and another set which defines the functions that are to be mapped on them. The final classes of this dimension are a list of all the valid mappings in French. For instance the final class *Subj0VObj1* is the class where the first argument is mapped with a subject and the second is mapped with a direct object[6].

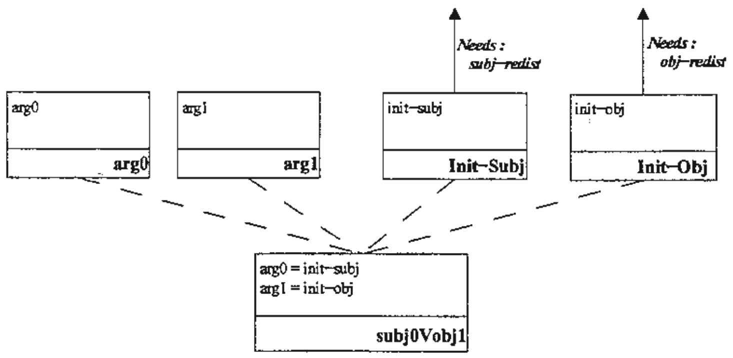


Figure 1: Overview of the first dimension

We express this mapping with the declaration of a constant[7]. Each of these quasi-nodes is equated with the one representing its associated function.

To be linguistically well formed, we impose the requirement that the crossed classes map each initial function to a final function. Then a class where an initial function node is defined contains a corresponding **need** for a final function (see fig. 3.1).

5. These ideas are already central in (Candito 99) work.
6. For the sake of clarity, we do not expand here to the whole (Abeillé 91) analysis in families. We do not consider here sentential arguments and verbal auxiliaries though they are important for the definitions of the families.
7. Following (Rogers et al. 94), a constant is denoting a node.

The second step in the generating process aims at defining a final subcategorization frame. Here we map the initial functions given above with final functions. As a side effect, the verb is given a morphological specification. For instance, through inheritance, the *full personal passive* maps the *initial-subject* to a *by-object*, the *initial object* to a *final-subject* and requests the predicate to be realized as a passive subtree. Furthermore, classes that introduce a final function emit the need that this function is realized.

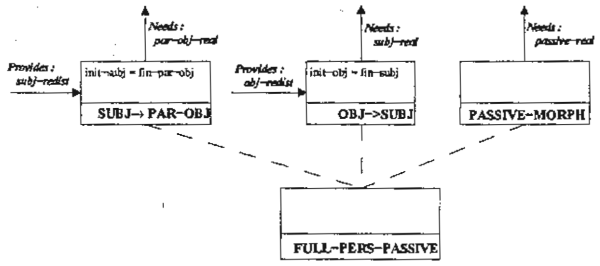


Figure 2: Overview of the second dimension

Our general strategy consists in manipulating functions through constants denoting nodes: instead of function features, we have such constants as *arg0, initial-subject, or final-object.* The redistributions are then performed by means of equalities between such constants. The final classes of dimension 2 express equalities between the nodes carrying the initial functions and the ones carrying the final functions. The interface with the first dimension is done through **providings** that satisfy the final-function **needs** of dimension 1 classes. For instance the *Full-personal-passive* class will inherit classes that provide the *subj-redist* and *obj-redist*. The content of the class reflects the mapping : the initial-subject node equals the final-by-object node and the initial-object node equals the final subject node (see fig. 3.1).

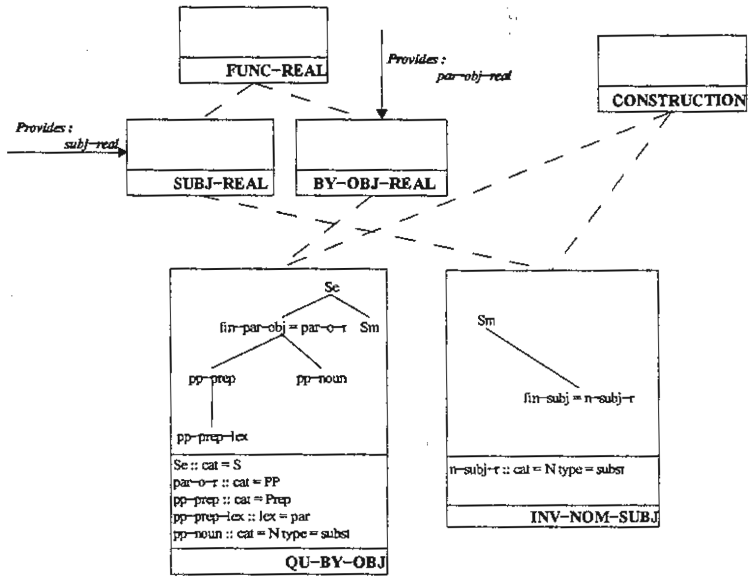


Figure 3: Overview of the third dimension

The functional realization 'dimensions', contains classes that actually yield trees. Basically, we view here a syntactic function as a label for a set of subtrees. Thus, this dimension groups all the subtrees and each of them is labelled as the representation of a function. The label assignment is performed through multiple inheritance as shown in figs. 3.1 and 3.1. Note that contrary to (Candito 99) approach, there are here two 'third dimensions'. One for the predicate's arguments and one for the predicate's head. The motivation is mainly methodological, we want to explicitly separate the functional and the topological part of the grammar.

Writing a metagrammar remains (at least for us) an experimental process. Other formalisms have been proposed that rely extensively either on feature structures (HPSG) or Linguistic functions (LFG). We experimented with a LFG inpired approach which allows us to deal with the topology of trees only in dimension 3. The general sketch is then to build feature structures in dimension 1 and 2 and to assemble the trees according to the specifications given by the feature structure in dimension 3.

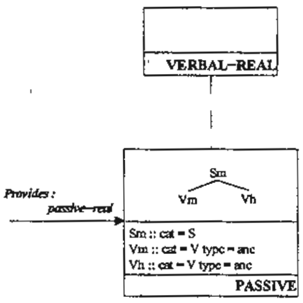


Figure 4: Overview of 'another' third dimension

3.2. A feature driven strategy

The feature structure descriptions, contained in classes and therefore associated to the produced trees at the end of the process, not only concern the anchoring, but may also actually describe the linguistic properties the tree is responsible for. Typically, it enables us to know that a tree is the representation of a two place predicate, that this predicate is a passive predicate, that the first argument is expressed as clitic and so forth. Thus the compiler allows to generate trees but also complex feature structures that are an explicit translation of what each tree 'means' linguistically.

In the previous approach we put the focus on the identification of particular nodes into the trees, and the feature structures associated to the classes only concerns the impossibilities in crossings. In the new approach we build complex feature structures and less complex formulas as lots of constants equated in rorder to represent redistributions simply disappear.

As an example, here are the features inherited by the following final classes:

- SUBJ0VOBJ1 :

$$\left[\text{PRED} \begin{bmatrix} \text{HEAD} & \begin{bmatrix}\text{CAT} & \text{Verb}\end{bmatrix} \\ \text{SUBCAT} & \left\langle \begin{bmatrix}\text{INIT-FUNC} & \text{SUBJECT}\end{bmatrix}, \begin{bmatrix}\text{INIT-FUNC} & \text{object}\end{bmatrix} \right\rangle \end{bmatrix}\right]$$

- FULL-PERS-PASSIVE :

$$\left[\text{PRED} \begin{bmatrix} \text{HEAD} & \begin{bmatrix}\text{VMORPH} & \text{passive}\end{bmatrix} \\ \text{SUBCAT} & \left\langle \begin{bmatrix}\text{INIT-FUNC} & \text{SUBJECT} \\ \text{FIN-FUNC} & \text{BY-OBJ} \\ \text{CONS} & \boxed{2}\end{bmatrix}, \begin{bmatrix}\text{INIT-FUNC} & \text{object} \\ \text{FIN-FUNC} & \text{subject} \\ \text{CONS} & \boxed{1}\end{bmatrix} \right\rangle \\ \text{CSET} & \begin{bmatrix}\text{SUBJ} & \boxed{1} \\ \text{BY-OBJ} & \boxed{2}\end{bmatrix} \end{bmatrix}\right]$$

- QUEST-BY-OBJ :

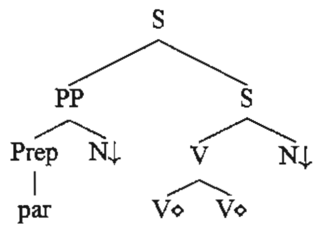
$$\left[\text{PRED} \begin{bmatrix} \text{CSET} & \begin{bmatrix}\text{EXTRACTION} & \text{By-obj} \\ \text{BY-OBJ} & \text{quest}\end{bmatrix}\end{bmatrix}\right]$$

- INV-SUBJ :

$$\left[\text{PRED} \begin{bmatrix} \text{CSET} & \begin{bmatrix}\text{SUBJ} & \text{inverted}\end{bmatrix}\end{bmatrix}\right]$$

In this approach, **needs** and **providings** are dispatched as they were in the previous approach. Classes of dimension 1 and 2 do not contain any formulas anymore. The third dimension realizes arguments as well as the predicate subtrees.

As a sample we generate the tree representing the schema that allows to analyze the sentence *Par qui sera accompagnee Marie ?* (*By whom will be accompanied Mary*) with the combination of the following final classes *subj0Vobj1, full-pers-passive, passive, inverted-subject, questioned-par-obj*(see figs. 3.1, 3.1, 3.1 and the feature structures given above) :



$$\Downarrow$$

$$\left[\begin{array}{ll}\text{PRED} & \left[\begin{array}{ll} \text{HEAD} & \left[\begin{array}{ll}\text{CAT} & \text{Verb}\\ \text{VMORPH} & \text{passive}\end{array}\right]\\ \text{SUBCAT} & \left\langle \left[\begin{array}{ll} \text{INIT-FUNC} & \text{SUBJECT}\\ \text{FIN-FUNC} & \text{BY-OBJ}\\ \text{CONS} & \boxed{2}\ \text{QUEST}\end{array}\right], \left[\begin{array}{ll} \text{INIT-FUNC} & \text{object}\\ \text{FIN-FUNC} & \text{subject}\\ \text{CONS} & \boxed{1}\ \text{inverted}\end{array}\right]\right\rangle \\ \text{CSET} & \left[\begin{array}{ll} \text{EXTRACTION} & \text{By-obj}\\ \text{SUBJ} & \boxed{1}\\ \text{PAR-OBJ} & \boxed{2}\end{array}\right]\end{array}\right]\end{array}\right]$$

Notice that when we use this second approach (which anyway is an enhancement of the node driven approach), the feature structure keeps track of the successive mapping steps that are performed throughout the process of generation. This approach consists of not declaring any structural constraints in the two first dimensions[8]. This solution has the benefit of clearly splitting the functional from the topological part of the grammar. But at the time of this writing, we are not able to establish a link between the feature structures associated to the classes and the constants of the logical formulas used to generate the trees.

4. Conclusion

The tool developped so far, though rough and buggy[9] enables us to experiment with metagrammar writing. As we just mentioned it also raises interesting questions regarding the precise objects we are dealing with when describing a grammar. One of the main drawbacks of our implementation is the absence of relationship between the descriptive feature structure and the logical formulas. In our opinion, the root of this problem concerns what a TAG grammar really is: a set of trees gathered in families together with indices indexing nodes (cf. **n0Vn1**) are more than just elementary trees.

References

Abeillé, A., *Une grammaire lexicalisée d'arbres adjoints pour le français. Application à l'analyse automatique*, Doctoral dissertation, Université de Paris 7, 1991.

Abeillé, A, Candito, M.-H., "FTAG : A Lexicalized Tree Adjoining Grammar for French", *in* Abeillé, A. and Rambow, O. éd. *Tree Adjoining Grammars. Formalisms, Linguistic Analysis and Processing*, Stanford, CSLI, 2000.

Candito, M.-H., Candito, "A Principle Based Hierarchical Representation of LTAGs", *COLING*, 1996.

8. That also suppress the trouble related to node deletion as mentioned earlier for the analysis of the agentless passive.
9. . . . and available at *http://www.loria.fr/equipes/led/outils/mgc/mgc.html.*

Candito, M.-H.,*Organisation modulaire et paramétrable de grammaires électroniques lexicalisées. Application au français et à l'italien*, Doctoral dissertation, Université de Paris 7, 1999.

Doran, C, Egedi, D, Hockey, A., Srinivas, B., Zaidel, M., "XTAG System - A Wide Coverage Grammar for English", *COLING*, 1994.

Doran, C, Sarkar, A., Srinivas, B., Xia, F., "Evolution of the XTAG System", *in* Abeillé, A. and Rambow, O. éd. *Tree Adjoining Grammars. Formalisms, Linguistic Analysis and Processing*, Stanford, CSLI, 2000.

Gerdes, K., DTAG ? Attempts to generate a useful TAG for German using a metagrammar, TAG+6, 2002.

Joshi, A. K., Levy, L. S., Takahashi, M., "Tree Adjunct Grammars", *Journal of Computer Science*, 1975.

Kinyon, A., "Hypertags", *COLING*, 2000.

Lopez, P., Bonhomme, P., "Resources for Lexicalized Tree Adjoining Grammars and XML encoding : TagML", *LREC*, 2000.

Rogers, J., Vijay-Shanker, K., "Obtaining Trees from Their Descriptions : An Application to Tree-Adjoining Grammars", *Computational Intelligence*, 10, 4, 1994.

Vijay-Shanker, K., Schabes, Y., "Structure Sharing in Lexicalized Tree-Adjoining Grammars", *COLING*, 1992.