

# Semantic Annotation of the French Treebank with Modular Graph Rewriting

Bruno Guillaume<sup>1,2</sup> Guy Perrier<sup>1,3</sup>

(1) LORIA - Campus Scientifique - BP 239 - 54506 Vandœuvre-lès-Nancy cedex

(2) INRIA Nancy Grand Est - 615, rue du Jardin Botanique - 54600 Villers-lès-Nancy

(3) Université de Lorraine - 34, cours Léopold - CS 25233 - 54502 Nancy cedex

bruno.guillaume@loria.fr, guy.perrier@loria.fr

## Abstract

We propose to annotate the French Treebank with semantic dependencies in the framework of DMRS starting from an annotation with surface syntactic dependencies and using modular graph rewriting. This system has been experimented on the whole French Treebank with the prototype which implements the rewriting calculus.

## 1. Introduction

We propose to produce a semantic annotation of large corpora from an annotation with syntactic dependencies. For the semantic annotation, we choose the framework of DMRS (Dependency Minimal Recursion Semantics). DMRS was introduced by (Copestake, 2009) as a compact and easily readable equivalent to Robust Minimal Recursion Semantics (RMRS), which was defined by (Copestake, 2007). This underspecified semantic formalism was designed for large scale experiments without committing to fine-grained semantic choices. DMRS graphs contain the predicate-argument relations, the restriction of generalized quantifiers and the mode of combination between predicates. Predicate-argument relations are labelled  $arg_i$ , where  $i$  is an integer following a fixed order of obliqueness *subj*, *obj*, *ats*, *ato*, *a-obj*, *de-obj*. . .

We have chosen DMRS because we aim at an annotation which is readable and minimal. We want to avoid any commitment to questionable linguistic choices. Moreover, the DMRS structures are based on dependencies like our initial syntactic structures, which makes the transformation easier. Regarding the input corpora, there are very few French resources syntactically annotated; the largest one is the *French Treebank*. The French Treebank is a corpus of sentences extracted from the newspaper “Le Monde”, which are annotated with phrase structures (Abeillé et al., 2003). These annotations have been converted into syntactic dependencies (Candito et al., 2009) following a format described in the guide designed for this task<sup>1</sup>. In our work, we use this last corpus under the abbreviation FTB.

To compute the semantic structures, we use the framework of *graph rewriting* (Bonfante et al., 2011), which is motivated by the shape of the manipulated objects. The output DMRS structures are graphs of semantic dependencies. The input syntactic structures are often dependency trees but to compute semantics, we need to complete these trees, for instance with some syntactic arguments of infinitives or some antecedents of pronouns determined by the syntax. We obtain wholly general graphs, with cycles and vertices that have several antecedents.

Term rewriting and tree rewriting can benefit from a canon-

ical definition, whereas no such definition exists for graph rewriting. For our application, we have chosen an operational view of graph rewriting. A graph rewriting system is defined as a set of *rewrite rules*. Given a rule, modification of the graph (which correspond to the part usually called “right-hand side” of the rule) is defined by means of a set of commands; the control of the way the rule is applied (usually called the “left hand-side”) uses pattern matching as is done in a traditional graph rewriting setting.

In our modeling of the syntax-semantics interface, every linguistic principle is translated into a few simple and readable rules. Since our ambition is to process large corpora, a complete system can have several hundred rules. The more rules, the more interaction between rules, and the consistency of the whole rule system becomes difficult to maintain. This impinges on our ambition of a large coverage for the grammar. To solve this problem, we propose to organize rules in *modules*.

A module is a set of rules that is linguistically consistent and represents a particular step of the transformation. There is no order between rules inside a module but we fix the order between modules according to linguistic criteria. Modules play a decisive role in the construction of a rewriting system but they are also crucial for the efficiency of computations. The transformation of syntax into semantics is non confluent by essence because the correspondence between syntactic and semantic structures is relational and not functional. With the use of modules, it is possible to restrict non confluence to some particular modules.

Another original feature of our calculus is the way that it makes the link between rules and lexicons. During the syntax-to-semantics transformation, many rules have to be controlled by lexical information; these rules are called *lexical rules*. Of course, many rules differ only by lexical information they contain; that’s why lexical rules can be parametrized. Each lexical rule is associated with a lexicon, which provides a list of possible values for the parameters. Every time such a rule is applied, it is verified that the values of input parameters given by the graph in which the rule matches are present in the associated lexicon.

In our application, we use lexical rules for the detection of syntactic subcategorization frames of verbs and the transformation of syntactic arguments into semantic arguments.

<sup>1</sup>[http://alpage.inria.fr/statgram/frdep/fr\\_stat\\_dep\\_parsing.html](http://alpage.inria.fr/statgram/frdep/fr_stat_dep_parsing.html)

To build the lexicons associated with lexical rules, we appeal to an external resource: Dicovalence (Van den Eynde and Mertens, 2003). Dicovalence is a lexicon of French verbs with more than 8000 entries. Each entry corresponds to a particular meaning of a verb and it gives rich information about its subcategorization frame. All entries have been validated manually by linguists.

Our rewriting system is composed of 562 rules including 402 lexical rules and organized in 34 modules. In the rest of the article, it is called `SYNSEMFTB`. Our calculus is implemented in a prototype which was used to experiment `SYNSEMFTB` on the whole FTB.

The plan of the article is the following: Section 2. presents the main features of our graph rewriting calculus; the system `SYNSEMFTB` of rules used for transforming the syntactic annotation of the FTB into a semantic annotation is described in Section 3. and finally Section 4. gives the results of the experimentation with the FTB.

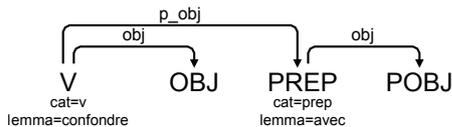
## 2. The main features of the calculus

### 2.1. The rules

Each rule is composed of two parts: the *template* and the *commands*. To explain each one, we consider a rule  $R$  which is applied to a graph  $G$ :

- The template contains one *positive pattern*  $P$  and a possibly empty set of *negative patterns*  $\{N_1, \dots, N_k\}$ . The positive pattern is a graph and each negative pattern  $N_i$  is a graph extension<sup>2</sup> of  $P$ . A rule can be applied if the positive pattern  $P$  match the graph  $G$  and, for all negative graph extension  $N_i$ , the matching of  $P$  into  $G$  fail to extend to a matching of  $P \cup N_i$  in  $G$ .
- The commands part of the rule describes atomic operations that are used to modify the graph  $G$  when the rule  $R$  is applied. A command is defined relatively to the positive pattern of the rule; it can be addition or deletion of a node, an edge or a feature, merging of two nodes .... Commands are executed in order and then this order is relevant.

To illustrate our purpose, we consider the rule that transforms the syntactic arguments into semantic argument for the transitive verb *confondre* with an indirect object introduced with the specific preposition *avec* ("*confondre X avec Y*" translates to "*mistake X for Y*"). We call it `subjvobjpobj`. Its positive pattern is represented with the following diagram:



The complete rule is encoded in the following way:

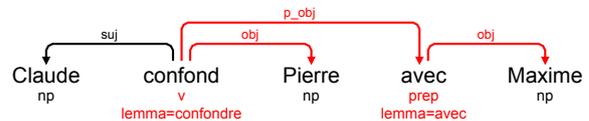
```
1 rule subjvobjpobj {
2   match{
```

<sup>2</sup>A negative pattern can contain edges referring to nodes defined in  $P$ ; formally, in negative conditions, we consider the graph defined by the union of  $P \cup N_i$ .

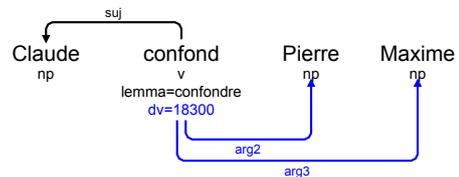
```
3   V [cat=v, lemma="confondre"];
4   OBJ [];
5   objrel: V -[obj]-> OBJ;
6   PREP [cat=prep, lemma="avec"];
7   preprel:V -[p_obj]-> PREP;
8   POBJ [];
9   pobjrel:PREP -[obj]-> POBJ;
10  }
11  without{ V[dv=*] }
12  without{ V-[a_obj|de_obj|ato|ats]->* }
13  commands {
14    del_edge objrel; del_edge preprel;
15    del_edge pobjrel; del_node PREP;
16    add_edge V -[arg2]-> OBJ; add_edge V -[arg3]-> POBJ;
17    V.dv = 18300;
18  }
19 }
```

Lines 2 to 10 describe the positive pattern  $P$  of `subjvobjpobj` rule. Lines 11 and 12 describes two negative patterns  $N_1$  and  $N_2$ . The first one ( $N_1$  on line 11) prevents the rule to loop indefinitely: if the node  $V$  has already a feature named `dv`, the rule application is blocked. The second one ( $N_2$  on line 12) prevents the matched verb to have other complements in the graph than a direct object (OBJ here) and an indirect object (POBJ here). Lines 13 to 18 describe the commands of the rule, which replace the direct object and indirect object dependencies with semantic dependencies, removing the preposition. Line 17 is used to record information about the lexical information used: in our case, we use Dicovalence which gives an integer identifier to each entry; it can be useful to keep track of this information.

Rule `subjvobjpobj` applies to the following graph representing the syntax of the sentence *Claude confond Pierre avec Maxime* (*Claude mistakes Pierre for Maxime*).



The positive pattern of the rule matches with the part of the graph in red and the two negative patterns fail to extend this matching. Therefore, the rule applies and transforms the graph into the following one:



If some rule application on  $G$  produces  $G'$ , we write  $G \rightarrow G'$ . If  $G$  rewrites to  $G'$  with any number of rule applications, we write  $G \rightarrow^* G'$ .

### 2.2. Lexical rules

The rule presented above uses lexical information. If we want to create a similar rule for every subcategorization frame of every verb, the best is to start from an existing lexicon of verbs. In our application to the FTB, we have chosen Dicovalence. Of course, we want to factorize the rules for different verbs which share the same subcategorization frame. Consider again the case of transitive verbs with an indirect object introduced with a specific preposi-

tion. The rule `subj_V_obj_pobj` defined above can be adapted to all transitive verbs with an indirect object introduced with the preposition "avec". We can go one step further and consider that the preposition is also a parameter of the rule.

Thus, in order to avoid the increase in the number of rules, we define *parametrized rules*. In addition to the parametrization of the pattern, we also consider parameters in commands. Parametrized rules allow:

- several parameters (prefixed with the symbol `$` in rule code) to be used in the positive and negative patterns as input parameters;
- other parameters (prefixed with the symbol `@` in code) to be used in the commands as output parameters.

For instance, `subj_V_obj_pobj` rule is generalized into the parametrized rule below by changing its head (lines 2 and 3 in blue) and the value of some features (lines 6, 9 and 20 in red):

---

```

1 lex_rule subj_V_obj_pobj
2   (feature $lemma, $prep, @dicoval_id;
3     file "subj_V_obj_pobj.lp")
4 {
5   match{
6     V [cat=v, lemma=$lemma];
7     OBJ [];
8     obj: V -[obj]-> OBJ;
9     PREP [cat=prep, lemma=$prep];
10    pobj1:V -[p_obj]-> PREP;
11    POBJ [];
12    pobj2:PREP -[obj]-> POBJ;
13  }
14  without { V [frame=*] }
15  without {V -[a_obj|de_obj|ato|ats]-> *}
16  commands {
17    del_edge obj; del_edge pobj1;
18    del_edge pobj2; del_node PREP;
19    add_edge V -[arg2]-> OBJ; add_edge V -[arg3]-> POBJ;
20    V=@dicoval_id;
21  }
22 }

```

---

This rule refers, on line 3, to an external file (`subj_V_obj_pobj.lp`) which contains 151 entries coming from Dicovalence and corresponding to the concerned subcategorization frame. Each line of the file describes an entry: `th<e lemma, the governed preposition and the identifier in Dicovalence:`

---

```

accommoder#avec##900
accorder#avec##1090
accoupler#avec##1305
...
confondre#avec##18300
...
troquer#contre##84610
voir#en##86390

```

---

The two first elements are the possible values of the input parameters `$lemma` and `$prep` and the last one is the value of corresponding output parameter `@dicoval_id`. If the new `subj_V_obj_pobj` rule is applied to the previous graph representing the syntax of the sentence "Claude confond Pierre avec Maxime", the positive pattern matches in the same way but the matching entails the instantiation of the input parameters: `$lemma` with "confondre" and `$prep` with "avec".

Then, ones verifies that the `subj_V_obj_pobj.lp` file contains the pair ("confondre","avec"). If it is the case, the output parameter `@dicoval_id` is instantiated with the corresponding value in the lexicon<sup>3</sup>.

All lexical informations extracted from Dicovalence are automatically transformed into a set of rules (379 currently), each rule being associated with a file providing the possible instantiations of its parameters. When no directly usable resource exists, beginnings of lexicons have been built manually. This is the case for the subcategorization frames of adjectives and predicative nouns. This is also the case for scopal adverbs and non intersective adjectives.

### 2.3. Modules

The interest of rewrite rules is that their effect is local but it is a drawback if one considers the whole system of rules. It can contain several hundred of rules and so, if all rules are allowed to act in parallel, it is difficult to control their interaction. That's why the grouping of rules in modules and the ordering of modules is crucial.

Modularity makes the design, the maintenance and the adaptation of a rule system easier. A module is a set of rules that is linguistically consistent and represents a particular step of the transformation. For instance, in our proposal, there is a module transforming the syntactic arguments of verbs into their semantic arguments. Another module determines the semantic representation of attributive constructions.

Formally, a module is a finite set of rules. A module has the *termination property*, if for any graph  $G$ , there is no infinite rewriting  $G \rightarrow G_1 \rightarrow G_2 \rightarrow \dots$  starting from  $G$ . In this work; all the modules we consider have the termination property. Given a module  $M$ , a graph  $G$  is said to be a  $M$ -normal form if there is no rule in  $M$  that can be applied to  $G$ . For a module with the termination property, for any graph  $G$ , there is a finite number of  $M$ -normal forms  $G'$  such that  $G \rightarrow^* G'$ . A module is *confluent* if, when  $G \rightarrow^* G_1$  and  $G \rightarrow^* G_2$ , there is a graph  $G'$  such that  $G_1 \rightarrow^* G'$  and  $G_2 \rightarrow^* G'$ . The most important consequence of the confluence property is that it implies that every graph has an unique normal form and hence only one reduction path has to be computed; giving a much more efficient implementation.

In our rewriting system, rules are organized in a totally ordered list of modules that are applied in turn. Each module is applied to the list of normal forms of the previous modules. In the system presented in this article, there are 562 rules distributed between 34 modules. The global process we implement is not confluent; nevertheless it is possible to restrict the non-confluence to a small number of module: 26 of the 34 modules are confluent in our experiment.

### 2.4. Filters

Depending of the design of the module, normal forms for the rewriting calculus may not be linguistically consistent structures. For instance, consider the module that transforms the syntactic arguments of verbs into semantic argu-

---

<sup>3</sup>To guarantee that the application of the rule is deterministic, we assume that the relation between input and output parameters in the lexicon file is functional.

ments. After application of this module, some graphs may still include syntactic dependencies. It can happen that an intransitive verb remains with a direct object. Several reasons can explain this situation: a wrong annotation of the initial graph, a bad choice among the different uses of a verb, the non detection of an impersonal construction . . .

To remove inconsistent normal forms, we have introduced *filters*. A filter contains only a template (*i.e.* a positive pattern and some negative ones), it has no command part. Filters are defined inside a module and they are used in a post-processing step: among normal forms obtained with usual rules of the modules, normal forms that contains a filter template are removed. For instance, in the module transforming the syntactic arguments of verbs into semantic arguments, we have introduced a filter removing all graph in which some syntactic argument remains.

### 2.5. Rewriting complexity

Every rule has to perform the matching of a graph with a sub-graph of another graph. This problem in its whole generality is known to be NP-complete. Nevertheless, graph rewriting is used here in particular conditions:

- rule patterns are very small;
- all patterns of a rule are connected;
- the number of edges going out of a given pattern vertex is bounded (in our application, the bound is 4).

In these conditions, the matching of a rule with a graph is linear in time with respect to the size of the graph.

Moreover, to describe the syntax-semantics interface of natural languages, a general mechanism of vertex creation is not necessary. In fact, all vertices of the final structure are predictable from the initial structure. It is expressed in the rule definition in the following way: every vertex that is created must be attached to a vertex present at the entrance inside the module and it is possible to attach only a finite number of new vertices to a given vertex.

In our system, it is possible to define a measure on graphs which decreases proportionally to the square of the graph size. Therefore in all confluent modules, every computation runs in a time that is cubic with respect to the size of the input graphs.

To summarize, the computation complexity is not due to the chosen method, graph rewriting, but to the essence of the problem to which it applies: the production of a semantic representation from the syntax of a sentence. More precisely, the main source of complexity is lexical ambiguity.

## 3. The rewrite rule system

### 3.1. The organization of modules

The current SYNSEM<sub>FTB</sub> system is composed of 562 rules including 402 lexical rules and organized in 34 modules. Figure 1 presents the system of modules as a graph, in which vertices represent modules and edges represent the order in the execution of modules.

These modules are themselves grouped in 6 packages:

- INIT transforms the initial CONLL annotation in a format compatible with the rules computing deep syntax;
- VERB is dedicated to the normalization of the verbal kernel: auxiliaries are deleted, transitive and pronominal verbs are marked, their voice is recognized as the active, passive or middle voice;
- SUJ is dedicated to the treatment of verb and adjective subjects<sup>4</sup>;
- ARGV computes the deep syntactic arguments of verbs by redistribution of surface arguments (transformation of impersonal, causative, passive and middle constructions) and by lexical determination of some infinitive arguments;
- PRO normalizes the syntax of several pronouns and determines the antecedent of relative and reflexive pronouns;
- SEM computes the semantic relations between predicates and individuals coming from the deep syntactic dependencies between word lemmas.

The four packages VERB, SUJ, ARGV and PRO contribute to the production of the sentence annotation with deep syntactic dependencies. Hence they are gathered in the same super-package DSYNT. Then, the rules of the package SEM compute a semantic representation of sentences in the DMRS format. Consider the following sentence:

- (1) *Jean a pu être autorisé à partir et à  
John might be allowed to leave and to  
regagner son domicile.  
return home.  
John has managed to be allowed to leave and to return  
home.*

Figure 2 illustrates the two step computations with sentence (1):

- the initial annotation with surface syntactic dependencies according to the annotation guide of the FTB,
- the annotation with deep syntactic dependencies resulting from the application of the rules from INIT, VERB, SUJ, ARGV and PRO packages,
- the semantic representation in the DMRS format after execution of the rules from SEM package.

Even if the number of rules in the system is high, the effect on the computation complexity is limited due to the shape of rules. For every rule, the patterns are connected graphs. These graphs are generally trees with a depth of 3, except for 5 rules from ANT\_REL\_PRO module, in which the patterns have two roots.

The main source of complexity comes from the non confluence of computations but non confluence is confined in 8 among the 34 modules<sup>5</sup>.

<sup>4</sup> The systematic assignment of a subject to an adjective facilitates the subsequent treatment in a more uniform way.

<sup>5</sup>On Figure 1, they are represented with opaque ovals.

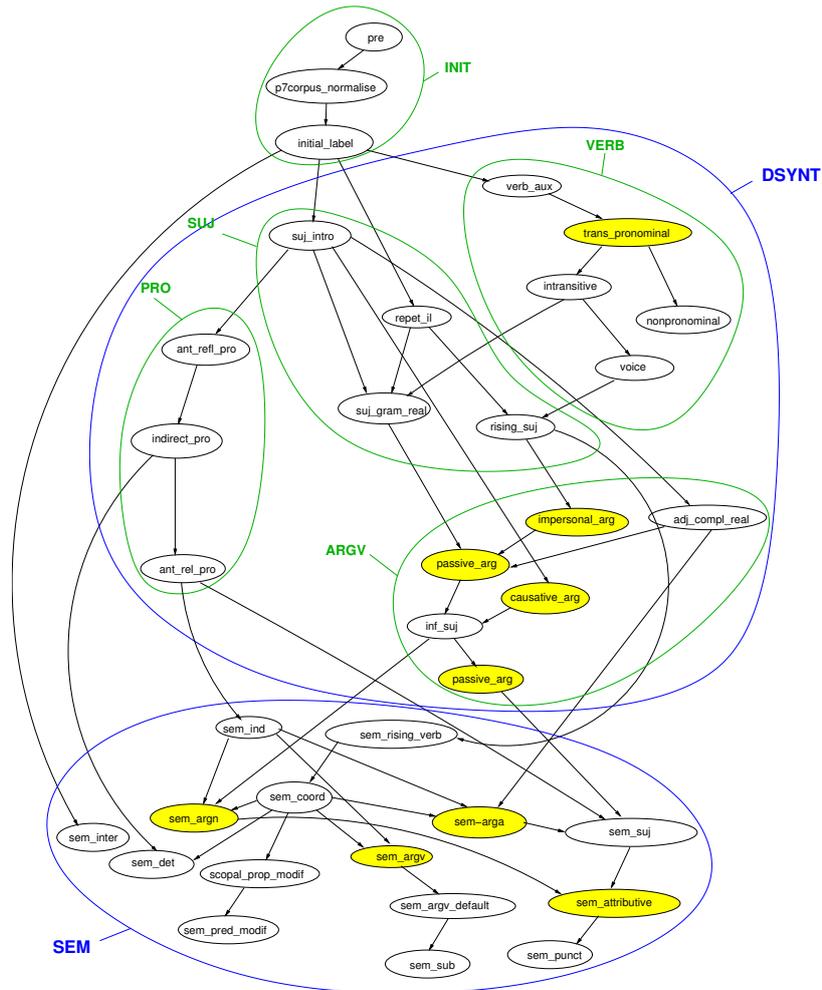


Figure 1: Diagram of the module system

### 3.2. The grammatical coverage

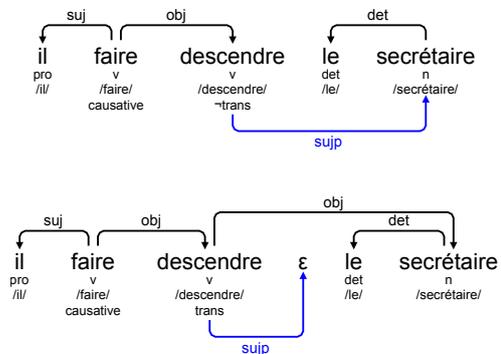
The SYNSEM<sub>FTB</sub> system covers the most common grammatical phenomena in French related to the syntax-semantics interface, except comparative constructions and parenthetical expressions, which are very frequent in the corpus. Regarding lexical informations, Dicovalence gives a satisfying coverage for verbs but for the predicate nouns and adjectives, we have written small lexicons.

As an illustration, here are some significant examples. Consider causative constructions. In the FTB, a causative verb is viewed as an auxiliary of the complement infinitive. This infinitive is then the head of the verbal kernel and the governor of all its arguments. Here is the annotation in dependencies according to the FTB guide of the sentence "il fait descendre le secrétaire":



As (Abeillé et al., 1997) shows it, this view is too simplistic and does not take some aspects into account. In the example above, "secrétaire" is the direct object of "descendre" but it is ambiguous: if "secrétaire" is a person ["secretary"], it

can be the deep subject of "descendre" ["to go down"]; if it is furniture ["secrétaire"] or if "descendre" is used with the meaning of ["to kill"], "secrétaire" is then the deep object of "descendre". In order to distinguish the two readings, SYNSEM<sub>FTB</sub> transforms the causative auxiliary into a full verb, which makes possible to express the two readings through the annotations below:



In both diagrams above, a dependency representing the deep subject of "descendre" has been introduced. In the first one, this subject is "secrétaire" and in the second one, it is not expressed in the sentence, which is represented with an empty word denoted ε. In a general way, all surface syntactic dependencies are represented over the text and the

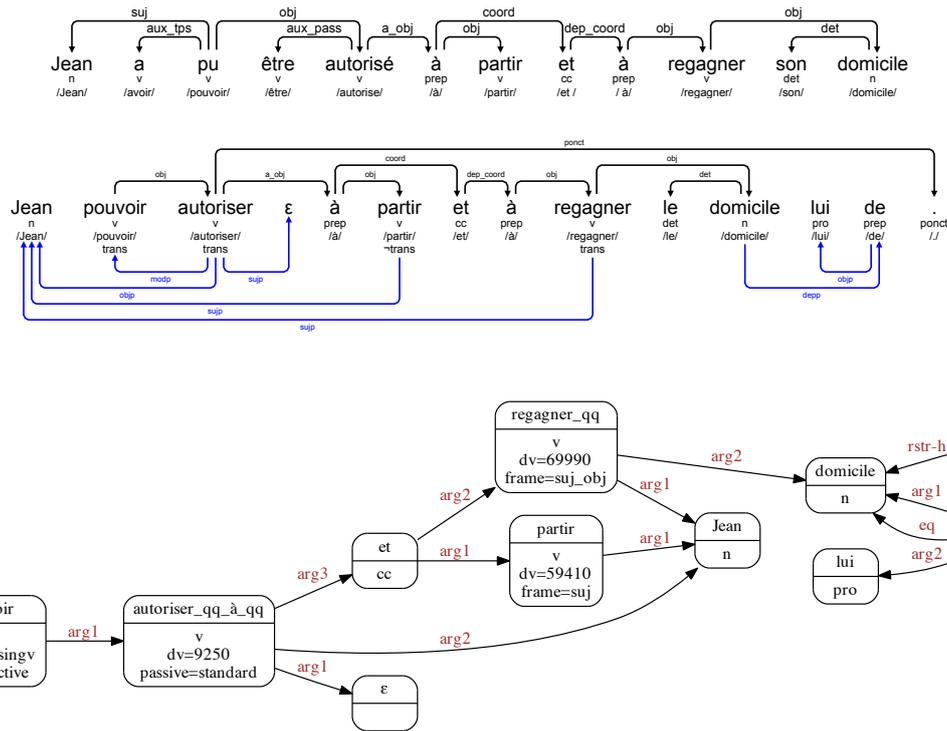


Figure 2: Surface syntax, deep syntax and semantics for the sentence "Jean a pu être autorisé à partir et à regagner son domicile".

deep (syntactic and semantic) dependencies are drawn under the text. Deep syntactic dependencies have a label ending with "p" (*subj*, *objp*, ...) to be distinguished from surface syntactic dependencies.

Rising verbs are dealt with in the FTB like full verbs in the same way as control verbs. However there are good reasons (Rooryck, 1989) to consider them as hybrid objects which sometimes behave as auxiliaries. Consider the following examples where rising verbs are written in bold and their complement infinitives are underlined:

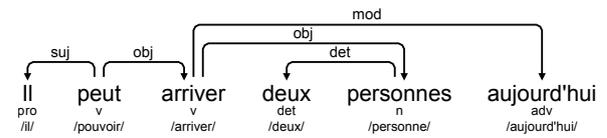
- (2) *Il **peut** arriver deux personnes aujourd'hui.*  
It may arrive two persons today.  
'Two persons may arrive today.'
- (3) *La maison **peut** être vendue aujourd'hui.*  
The house may be sold today.
- (4) *La maison **peut** se vendre aujourd'hui.*  
The house may itself sold today.  
'The house may be sold today.'

They successively illustrate an impersonal construction, a passive and a middle voice. To avoid the increase of the number of rules computing the semantic arguments, it is necessary to transform every construction into a canonical construction. Usually the canonical construction is the personal construction in the active voice. For the three previous examples, we obtain:

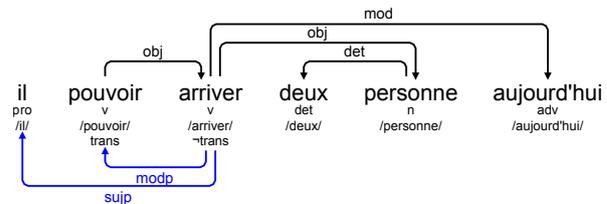
- (2') *Deux personnes **peuvent** arriver aujourd'hui.*  
Two persons may arrive today.

- (3'), (4') *On **peut** vendre la maison aujourd'hui.*  
One may sell the house today.

The rewrite rules used for this transformation are made simpler if the rising verbs are dealt with as auxiliaries. Then, the same rules as for verbs not depending on rising verbs can apply. Let us describe the method on example (2). Here is the initial annotation given by the FTB guide.

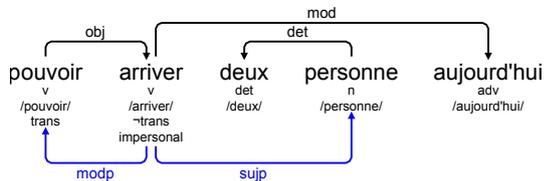


A first rule transforms the rising verb "pouvoir" into an auxiliary of "arriver" which becomes the sentence head.



As the dependency structure is not restricted to be a tree, more expressive power is available. So the verb "pouvoir" keeps "arriver" as its object while being a modifier of it, which produces a cycle in the graph. As a consequence, the sentence head is no longer a tree root but it remains the distinguished vertex which is aimed at receiving any external dependency, especially when the sentence is inserted as a

subordinated clause into another complex sentence. After the transformation, the impersonal construction can be processed as any impersonal construction because it is anchored to the verb "arriver". A rule aims at recovering the corresponding canonical construction. It removes the impersonal subject and the surface object becomes the deep subject of the verb. Hence, the resulting annotation:



### 3.3. Module Ordering

The organization of rewrite rules in modules is inseparable from the definition of a partial execution order for the modules. This order is determined by linguistic and representation choices. For instance, the module dedicated to impersonal constructions precedes the module dedicated to the passive voice so that impersonal passive constructions can be processed correctly.

Some cases may be very complicated. Consider the following examples:

(5) *Jean est **autorisé** à arriver plus tard.*  
John is allowed to arrive later.

(6) *Jean **demande** à Marie d'être accompagnée par sa fille.*  
John asks to Mary to be accompanied by her daughter.

In every sentence above, a verb in bold controls the subject of an infinitive which is underlined. For sentence (5), the module redistributing the passive construction, denoted PASSIVE\_ARG, must apply first to produce the following canonical construction:

(5') *On **autorise** Jean à arriver plus tard.*  
One allows John to arrive later.

Then, the module determining the subject of infinitives depending on control verbs, denoted INF\_SUJ, can apply. In this module, a lexical rule indicates that the subject of "arriver" is the direct object of "autorise".

For sentence (6), it is the contrary. Module INF\_SUJ must be applied first to find that the subject of "être accompagnée" is the indirect object of "demande". Then, module PASSIVE\_ARG is applied to transform the passive voice into the active voice:

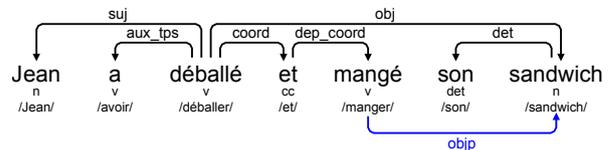
(5') *Jean **demande** à Marie que sa fille l' accompagne.*  
John asks to Mary that her daughter accompanies her.  
John asks to Mary that her daughter accompanies her.

Hence, in the order of module application, is present the sequence PASSIVE\_ARG, INF\_SUJ and again PASSIVE\_ARG, as diagram 1 shows it.

Coordination is very specific and cannot be dealt with as most syntactic constructions within a particular module, which would be inserted somewhere in the module ordering graph. It allows sharing between structures and thus interacts with other syntactic constructions. This interaction is modeled with rules which are distributed between the different modules computing the deep syntax. Of course, these rules depend on the choice made for annotating coordination in the FTB; in particular, imposing the dependency structure to be a tree has strong consequences. So, the sharing of structures frequently introduced by coordination cannot be expressed because it needs that two dependents share the same governor. Rules of SYNSEM<sub>FTB</sub> are dedicated to recovering sharing between coordinated structures. Depending on whether the shared structure is the subject of a verb, a passive or tense auxiliary or the antecedent of a relative pronoun, the phenomenon is respectively dealt with in the module introducing subjects (SUJ\_INTRO), removing auxiliaries (VERB\_AUX), or finally determining the antecedent of relative pronouns (ANT\_REL\_PRO).

The sharing of complements is more difficult to deal with because some complements may be optional, which makes the annotation of the FTB ambiguous. Consider the following sentence annotated according to the FTB guide.

(7) *Jean a déballé et mangé son sandwich.*  
John has unpacked and eaten his sandwich.



The *objp* dependency under the sentence must be added to express that "sandwich" is shared by "déballé" and "mangé". Unfortunately, the FTB does not allow such a dependency because "sandwich" would have two governors, which violates the treeness constraint on the dependency structure. Therefore, it is not possible to distinguish the sentence in which "son sandwich" is shared, from sentence "Jean a déballé son sandwich et mangé" if one merely takes the dependency structure into account. Only the linear order between words allows the distinction. In SYNSEM<sub>FTB</sub> we have chosen to leave this problem aside provisionally.

## 4. Experimental results

The SYNSEM<sub>FTB</sub> module system has been applied to the 12 351 sentences of the FTB with the GREW<sup>6</sup> software. Complete statistics about the rule use by module on the whole corpus and detailed results (with the produced structures) on 1% of the corpus are available online<sup>7</sup>.

Since computations are not confluent, we are interested first in the number of normal forms produced for each input sen-

<sup>6</sup><http://grew.loria.fr>

<sup>7</sup><http://wikilligramme.loria.fr/doku.php?id=lrec2012>

aff	arg	dep	aux_caus	aux_tps	aux_pass	comp	coord	dep_coord	det
1581	465	32821	128	3806	1672	37	6358	7299	40355
0	366	2240	5	26	15	37	194	1205	597
0,0%	78,7%	6,8%	3,9%	0,7%	0,9%	100,0%	3,1%	16,5%	1,5%
mod	mod_rel	ponct	subj	obj	a_obj	de_obj	p_obj	ato	ats
58353	2352	35810	15100	58132	2192	1668	1663	160	2521
13914	407	10598	158	3127	15	23	4	15	15
23,8%	17,3%	29,6%	1,0%	5,4%	0,7%	1,4%	0,2%	9,4%	0,6%

Table 1: Dependency types processed by the SYNSEM<sub>FTB</sub> rewriting system

tence. On one hand, long sentences can be highly ambiguous. On the other hand, filters at the end of some modules can discard an important number of inconsistent annotations. Some sentences may even have no normal form; it is the case for 16.0% of the sentences. For 33.1% of the sentences, the system returns one normal form, for 20.6%, it returns two normal forms. 92.3% of the sentences lead to a number of normal forms that is less than or equal to 8.

To estimate the coverage of the SYNSEM<sub>FTB</sub> system, we have observed the number of dependency types that are present in the FTB and that are ignored in the rewriting process. Table 1 shows the number of dependencies present in the resulting annotation with respect to the number present in the initial annotation for every dependency type and for the sentences having one normal form at least.

It is difficult to distinguish the cases coming from an annotation error from the cases that are not covered by SYNSEM<sub>FTB</sub>. Nevertheless, for dependency types having more than 5% of occurrences remaining at the end, we can give some comments. Since comparatives are not dealt with by SYNSEM<sub>FTB</sub>, the *comp* dependencies are not rewritten. Punctuation dependencies related to parenthetical expressions are not dealt with too. The remaining *mod* dependencies mainly concern nouns modifying nouns or verbs and the remaining *mod\_rel* often correspond to relative clauses in which the head verb is missing. For coordination, it is not rare to find several *dep\_coord* dependencies starting from the same governor, which is an annotation error. *dep* dependency type is underspecified and used in various contexts, which are not all predicted by the FTB annotation guide.

Finally, we are interested in the use of Dicovalece in the rewriting of the FTB. The FTB contains 39 104 verbs and the sentences having at least one normal form contain 29 782 verbs. For these sentences, we find 17 542 verbs associated with a Dicovalece entry in the final annotation. It represents 58.9% (17 542 out of 29 782) of the rewritten verbs and 44,9% (17 542 out of 39 104) of all verbs.

We have studied the inconsistency cases in a precise way on the 1% of the corpus that is available online. It is composed of 120 sentences and 12 of them lead to no normal form. For 5 of them, Dicovalece does not describe the needed subcategorization frame: "hispaniser" transitive, "acheter" with indirect object and without direct object, "maintenir" with locative complement, "souscrire" in its transitive use and "vendre" in its intransitive use. Other cases correspond to annotation errors.

## Conclusion

Through the example of the FTB, we have shown that graph rewriting is relevant for the automatic annotation of large

corpora with semantic dependencies starting from surface syntactic dependencies.

Modules play a major role in the rewriting system both for controlling computations and maintaining the global consistency of the system. SYNSEM<sub>FTB</sub> has been designed according to particular input and output formats, but the modularity of the system allows its adaptation to other formats at minimal cost. The fact that the module PASSIVE\_ARG is used twice in the list of modules is not satisfactory. We can imagine sentences where it must be used more and it suggests to enrich the strategy language with the possibility to apply iteratively a sequence of modules until a fixpoint is reached; we leave this for further work.

Regarding the specific application to the FTB, the resulting annotation presents two main limits: the initial annotation contains many errors and the formal framework chosen for semantic annotation, DMRS, offers no solution for the modeling of some semantic properties (intentionality, comparatives, ...). We hope to push the first limit by using graph rewriting for correcting the most systematic errors. The solution to the second limit depends on advances in formal semantics.

## 5. References

- A. Abeillé, D. Godard, and P. Miller. 1997. Les causatives en français, un cas de compétition syntaxique. *Langue française*, 115:62–74.
- A. Abeillé, L. Clément, and F. Toussenet, 2003. *Building a Treebank for French*, chapter 10. Kluwer Academic Publishers.
- G. Bonfante, B. Guillaume, M. Morey, and G. Perrier. 2011. Modular graph rewriting to compute semantics. In *IWCS 2011*, pages 65–74, Oxford, UK, January.
- M.-H. Candito, B. Crabbé, P. Denis, and F. Guérin. 2009. Analyse syntaxique statistique du français : des constituants aux dépendances. In *Actes de TALN 2009 (Traitement automatique des langues naturelles)*, Senlis, June. ATALA, LIPN.
- A. Copestake. 2007. Semantic composition with (robust) minimal recursion semantics. In *Proceedings of the Workshop on Deep Linguistic Processing*, pages 73–80. Association for Computational Linguistics.
- A. Copestake. 2009. *Invited Talk*: Slacker semantics: Why superficiality, dependency and avoidance of commitment can be the right way to go. In *Proceedings of EACL 2009*, pages 1–9, Athens, Greece.
- J. Rooryck. 1989. Les verbes à montée et à contrôle "ambigus". *Revue québécoise de linguistique*, 18(1):189–206.
- K. Van den Eynde and P. Mertens. 2003. La valence : l'approche pronominale et son application au lexique verbal. *French Language Studies*, 13:63–104.