Dependency Parsing with Graph Rewriting

Bruno Guillaume
Université de Lorraine, CNRS
LORIA, UMR 7503,
Vandœuvre-lès-Nancy, France
Inria, Villers-lès-Nancy, France
Bruno.Guillaume@loria.fr

Guy Perrier
Université de Lorraine, CNRS
LORIA, UMR 7503,
Vandœuvre-lès-Nancy, France
Inria, Villers-lès-Nancy, France
Guy.Perrier@loria.fr

Abstract

We propose to use Graph Rewriting for parsing syntactic dependencies. We present a system of rewriting rules dedicated to French and we evaluate it by parsing the SEQUOIA corpus.

1 Introduction

The most popular frameworks (TAG, CG, LFG, HPSG) for symbolic parsing are based on the notion of grammar. They defined a set of initial structures (often strongly linked to a lexicon) and a set of rules to express how initial structures can combine into larger ones. In this setting, parsing consists in searching for a syntactic structure predicted by the grammar for an input sentence. Among drawbacks of these methods, there is the fact that they may be inefficient when large-coverage grammar are considered (the search space grows very quickly for large sentences) and that they are not easy to use in contexts where robust parsing is needed (grammars describe set of correct sentences but do not give structures to sentences that are not completely covered by the grammar). Another problem with grammar-based parsing is that it is a difficult task to maintain the global consistency of the grammar. Moreover, development of large coverage grammar is known to be a time-consuming task.

On the other side, statistical methods build language models with learning algorithm applied to large annotated corpora. With respect to symbolic methods, it is easier to build robust parser with these methods and it is also easier to adapt a method to a new kind of corpus or to a new natural language. The main drawbacks are that good results are obtained only if large and well-annotated corpora are available. It is also difficult to improve a system: learning provides a language model which is essentially a black box which cannot be read by a human; external mechanism must be used if someone want to include linguistic knowledge in the system.

In this paper we propose a symbolic method which is defined in a Graph Rewriting (GR) framework. The output format is dependency syntax of natural language sentences. We propose to describe the parsing process as a sequence of atomic transformations starting from a list of lexical units (a tokenized sentence) to a dependency tree[1] built on the same lexical units. Each atomic transformation is described by a handcrafted rule. Then, instead of defining a grammar that describes the set of well-formed structures, we define rules which describe linguistic contexts in which a dependency relation can appear.

The rule system input is made of lemmatized and POS-tagged sentences. For the experiments in this paper, we use the SEQUOIA corpus (Candito and Seddah, 2012) (version 6.0[2]) as the gold standard. We experiment our system in two settings: on gold POS-tagged text (taken from SEQUOIA data) and on POS-tagging given by the MElt tagger (Denis and Sagot, 2012).

We use the general framework of GR where each transformation is given by two parts: first, the conditions that control when the transformation may apply (the *pattern*) and second, a description of the way the structure should be modified.

In the system we proposed, the input format is a tokenized sentence where each lexical unit is given a lemma and a POS-tag; the output format is a dependency structure; hence, both input and output structure can be represented as trees. Nevertheless, we use GR to describe our rules. At a first sight, it may be surprising to use such a formalism to manipulate only trees. But, the first thing to notice is that matching algorithms used in GR are direct generalization of matching algorithm that can

[1] The output may a partial dependency trees.

[2] https://gforge.inria.fr/projects/sequoiabank/

Proceedings of the 14th International Conference on Parsing Technologies, pages 30–39,
Bilbao, Spain; July 22–24, 2015.

be used in tree transformation: it means that, if all structures and patterns happen to be trees, the pattern matching in the GR setting will be as efficient as the pattern matching in the tree rewriting setting. A second benefit of GR is that it becomes possible to express more information in the intermediate structures. In the rule system, we use two kinds of relation to express linear order between lexical entities: the relation `SUC` links the heads of two partial dependency structures; the relation `INIT_SUC` links two successive lexical unit of the sentence, even if they have also been integrated in partial dependency structures. Structures with these two kinds of relations are graphs and cannot be represented as trees.

In a comparison with other works from the literature, we left out data-driven approaches which are far from our proposal. In (Foth et al., 2000), (Debusmann et al., 2004), weighted rules are used to described valid dependency structures and the parsing is expressed as a constraints resolution problem. (Covington, 2000) and (Nivre, 2003) propose rule-based processes to produce dependency structures but they are presented as kind of shift-reduce algorithm where word are treated one by one following the reading order, rules describing how each word can be link to the current state. Each rule only tells that a dependency from a word to another word is acceptable.

More close to our work is the proposal of (Oflazer, 2003) which defines a set of rules that are used iteratively until a fixpoint is reached and the rules application do not necessarily follow the reading order of the sentence. However, in (Oflazer, 2003) rules are encoded as regular expressions and are less flexible than a GR rule can be. To our knowledge, our proposal is the first use of the Graph Rewriting framework for symbolic dependency parsing.

In Section 2, we describe more precisely the GR framework used in the paper. In Section 3, the GR system considered is detailed. We finally give experimental results in Section 4.

2 Graph Rewriting

Unfortunately, there is no canonical formal GR definition. In this experiment, we use the GR definition which is implemented in the GREW software[3]. Rules are defined by two parts: a pattern and a set of commands. GREW was used for instance in (Bonfante et al., 2011) to build a semantic annotation of a French Treebank.

[3] `http://grew.loria.fr`

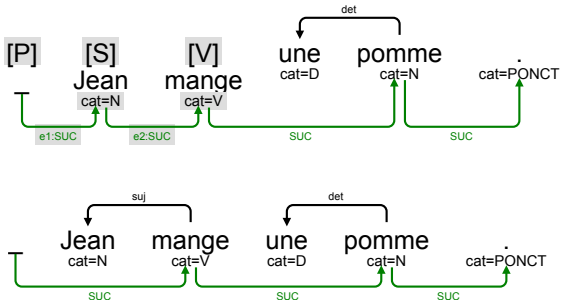

Figure 1: An example of application for the rule `subject_noun`

The reader can refer to the GREW documentation for a complete description of the GR framework and of the syntax of the rules with GREW. We give here a simple example of rule and of its application; more elaborated rules are shown in the next section. The code below is a simplified version of a rule for the subject relation.

```
1  rule subject_noun {
2    match {
3      S [cat=N|PRO];
4      V [cat=V, m=ind|subj];
5      e1:S -[SUC]-> V;
6      P []; e2:P -[SUC]-> S;
7    }
8    without { V -[suj]-> * }
9    without { S [lemma="que"|"dont"] }
10   commands {
11     del_edge e1;
12     del_edge e2;
13     add_edge V -[suj]-> S;
14     add_edge P -[SUC]-> V;
15   }
16 }
```

Different parts of rule are as follows. One **`match`** part (lines 2 to 7) describes the subgraph that must be found: here, the subgraph contains 3 nodes `S`, `V` and `P` linked by two relations `SUC` (lines 5 and 6). Any number of **`without`** parts describe negative application patterns; if any of these negative patterns is found, the rule application is blocked. For instance (line 8), if there is already a `suj` dependency starting from `V`, the rule does not apply (it prevents from putting two subjects for the same verb). Finally, a **`commands`** part describes how the graph is transformed by the rule application, in the example: add a `suj` relation from the `V` node to the `S` node and update the `SUC` relations (the verb is now the successor of the `P` node). On a very simple sentence *Jean mange une pomme. (John eats an apple.)*, the two dependency structures of Figure 1 show respec-

tively the structure before (with identifiaction of matched nodes S, V and P) and after the rule application. With GREW, it is also possible to modify feature structures in the **commands** part: add a new feature, modify or remove an existing feature. The rule above uses some lexical information in the second **without** part. As some rules make a strong usage of lexical information, it is possible to parametrize rules by external lexicons with the **lex_rule** keyword. For instance, in the rule below, a relation obj in added (line 18) only if the verb lemma (line 7) is one on the lemmas given in the lexical file verb_with_obj_noun.lp (line 3). This file contains a list of more than 3,000 French transitive verbs.

```
1  lex_rule verb_object_noun
2    ( feature $lemma;
3      file "verb_with_obj_noun.lp"
4    ) {
5    match {
6      OBJ [cat=N|PRO];
7      V [cat=V, lemma=$lemma, m=ind|subj];
8      POST [];
9      e1: V -[SUC]-> OBJ;
10     e2: OBJ -[SUC]-> POST;
11     V -[suj]-> *
12   }
13   without ...
14   without ...
15   commands {
16     del_edge e1;
17     del_edge e2;
18     add_edge V -[obj]-> OBJ;
19     add_edge V -[SUC]-> POST;
20   }
21 }
```

When the number of rules increases, some constraint should be put on the order in which the rules can be used. In this respect, a last feature, which is essential in the GREW usage, is the organization of rules into subsets called *modules*. The whole rewriting process is controlled by a total order on modules that are applied one after another; inside a module, no ordering is given and any rule may be apply anywhere in the graph.

Moreover in a general GR setting, the process may be non-confluent. Even if a total ordering is provided between modules, an arbitrary number of structures can be produced inside a module. In this work, we restrict ourself to deterministic GR, this means that we focus on the set of dependency relation that can be produced in a deterministic way. We can imagine many cases where non-confluent rewriting system can be used: for instance, PP-attachment is known to be a task that cannot be decided only with syntactic information. We leave as future work the development of non-confluent system and the problem of ranking in case of non-deterministic process.

3 FRDEP-PARSE: a System of Graph Rewriting Rules for Dependency Parsing

3.1 Input and Output Formats

FRDEP-PARSE takes a French sentence annotated with POS-tags and lemmas as input and returns a dependency syntax annotation for the same sentence as output. Let us describe the input and the output more precisely.

The input sentence is a sequence of tokens. Each token is equipped with a set of features:

- phon: the phonological form of the token;
- lemma: the lemma of the token;
- pos: the value of which is one of the 28 tags defined in the annotation guide of the French TreeBank[4];
- position: an integer indicating the position of the token in the sequence (this feature is used to express linear order between tokens).

At the beginning of the process, the 28 POS-tags are interpreted in terms of 13 grammatical categories (feature cat) and some other features. For instance, the 6 POS-tags V, VINF, VIMP, VS, VPP, VPR are all interpreted by feature cat=V and a m feature recording the mood (respectively indicative, infinitive, imperative, subjunctive, past participle and present participle).

Initially, the only relations between tokens are SUC relations of immediate succession between adjacent tokens.

The parsing output is the sentence annotated with syntactic dependencies according to the tagset used in SEQUOIA. The annotation may be partial. Figure 2 shows an example of syntactic annotation obtained with FRDEP-PARSE.

3.2 A CKY Basic Architecture

The basic form for the rewriting rules of FRDEP-PARSE aims at the implementation of a CKY-like algorithm in the dependency syntax framework. The dependency tree of a sentence is built bottom-up step by step. When two partial trees T_1 and T_2 have their terminal yields which are adjacent, a dependency may be added between the roots w_1 and w_2 of the two trees.

[4] http://alpage.inria.fr/statgram/frdep/Publications/FTB-GuideDepSurface.pdf

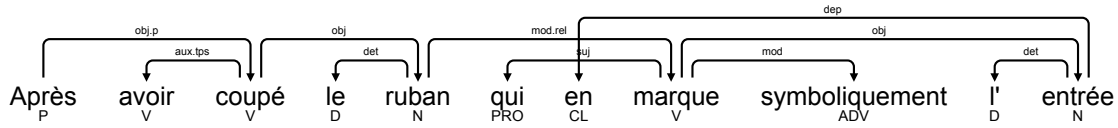

Figure 2: A non-projective result of syntactic annotation with FRDEP-PARSE.

To express adjacency between two yields, SUC is used with a larger meaning: if there is a SUC relation from a token w_1 to a token w_2, it means that w_1 and w_2 are roots of intermediate dependency trees, the yield of the first tree immediately preceding the yield of the second tree.

(Eisner, 1996) already proposed a CKY-like algorithm for parsing with dependencies, but he differs from our proposal on two points: he uses a statistical approach and to link two dependency trees, he takes only their roots into account and ignores information coming from deeper nodes.

The use of GREW for implementing the CKY algorithm in a strict manner has no point, but the Graph Rewriting approach allows to enrich the algorithm in various directions. One of them is to add internal and external constraints on the T_1 and T_2 trees.

Here is an example of constraints introduced by a rule of FRDEP-PARSE.

```
lex_rule verb_right-modif_adv
 (feature $lem; file "quant_adv.lp") {
  match {
    %positive constraint on T1
    V [cat=V];

    %positive constraints on T2
    ADV[cat=ADV];
    POST [ ];
    e1: ADV -[SUC]-> POST

    % adjacency between T1 and T2
    e2: V -[SUC]-> ADV
  }

  % negative constraints on T1
  without { ADV[pos=ADVWH] }

  %negative constraints on T2
  without { POST[cat=V,m=pastp] }
  without {
    ADV[lemma=$lem];
    POST[cat=P,lemma="de"]
  }

  commands{
    del_edge e1;
    del_edge e2;
    add_edge V -[mod]-> ADV;
    add_edge V -[SUC]-> POST
  }
}
```

The rule above is a lexical rule in which the parameter $lem represents the lemma of an adverb coming from the quant_adv.lp file gathering adverbs of quantity. It says that any adverb ADV, the yield of which immediately follows the yield of a verb V, is a modifier of the verb V. It includes three negative constraints:

- An internal constraint: ADV must be different from an interrogative adverb. For instance, in the sentence *Pierre demande combien ça coûte (Pierre asks how much it is)*, *combien* is not a modifier of *demande* but a complement of *coûte*.

- An external constraint: POST that immediately follows ADV is not a past participle because in this case, ADV depends on the past participle POST. For instance, in the sentence *Pierre a beaucoup travaillé (Pierre has worked a lot)*, *beaucoup* is not a modifier of the auxiliary *a* but of the past participle *travaillé*.

- A mixed constraint: ADV is not an adverb of quantity followed with the preposition *de*. For instance, in the sentence *Pierre connaît beaucoup de personnes (Pierre knows a lot of persons)*, *beaucoup* is not a modifier of *connaît* but of *personnes* at the opposite to the sentence *Pierre travaille beaucoup la nuit (Pierre works a lot in the night)*.

The following example highlights the expressivity of the graph rewriting approach, which allows the representation of constraints on deep levels in the T_1 and T_2 trees .

```
lex_rule impers_verb_obj_de_inf
 (feature $lem;
  file "il_verb_with_obj_de_inf.lp"){
  match {
    %positive constraints on T1
    V[cat=V,lemma=$lem];
    IL[phon="il"|"Il"];
    V -[suj]-> IL;

    %positive constraints on T2
    PREP[cat=P,lemma="de"];
    OBJP[cat=V,m=inf];
    PREP -[obj.p]-> OBJP;
    POST[];
    e1: PREP -[SUC]-> POST;

    % adjacency between T1 and T2
    e2: V -[SUC]-> PREP;
```

```
  }

  % negative constraints on T1
  without{SE[pos=CLR]; V -> SE}

  commands{
    del_edge e1;
    del_edge e2;
    add_edge V -[obj]-> PREP;
    add_edge V -[SUC]-> POST
  }
}
```

The rule above realizes the direct object of an impersonal verb, when this object is an infinitive introduced with the preposition *de*. The rule is lexical because it verifies that the verb is able to enter such a construction. The parameter of the rule is the lemma `$lem` of the verb.

A positive constraint on T_1 says that the verb `V` must have a subject *il* and a positive constraint on T_2 says that the tree is made of `PREP` *de*, a preposition introducing an infinitive `OBJP`.

Such constraints cannot be expressed with the classical CKY algorithm in a constituency approach, where constraints are limited to the external constituents of partial syntactic trees.

3.3 Non Projective and Disambiguation Rules

If all rules of FRDEP-PARSE had the form described above, the dependency structure resulting from their application would be projective. Therefore, we should be not able to represent some phenomena from the French grammar, which are essentially non-projective, and are present in the SEQUOIA corpus.

In FRDEP-PARSE, non-projectivity is introduced in two ways:

- in iterative modules, some rules link partial dependency trees with non-projective dependencies;
- specific rules are added in final modules with the aim of moving dependencies from provisional positions with provisional labels in order to obtain non-projective dependencies with definitive labels.

There are other final modules used for disambiguation. Indeed, at some intermediate steps of the parsing process, there is no sufficient information for deciding between several dependency labels. We could divide the search path into several paths. The repetition of such choice points would entail the time explosion of the parsing process. In this situation, we keep a unique search path by labelling the concerned dependencies with disjunction of elementary functions. At the end of parsing, the ambiguity is solved with specific rules using the information accumulated during the parsing process.

The following rule illustrates both functions: label disambiguation and expression of non-projectivity. It applies to Sentence *[annodis.er_00026]* from the SEQUOIA corpus, more specifically to the part *le ruban qui en marque symboliquement l'entrée (the ribbon, which symbolically marks the entry of it)*. Figure 1 shows the syntactic annotation produced by FRDEP-PARSE. The clitic pronoun *en* represents a complement of *entrée*. It is modelled with a dependency `dep` crossing the object dependency `mod.rel` from the noun *ruban* to the verb *marque*. The rule producing non-projectivity is the following:

```
lex_rule obj_dep_en
 (feature $lem ;
   file "verb_with_obj_deobj.lp") {
  match{
    CLIT[pos=CLO, phon="en"];
    V[cat=V];
    iobj_rel: V -[DE_OBJ-OBJ]-> CLIT
    OBJ[cat=N];
    V -[obj]-> OBJ
  }
  without { V [lemma=$lem] }
  commands {
    del_edge iobj_rel;
    add_edge OBJ -[dep]-> CLIT
  }
}
```

Initially, `CLIT` *en* depends on the verb `V`, which it cliticizes in an ambiguous relation `de_obj` (indirect object introduced with the preposition *de*) or `obj` (direct object). When the verb `V` has found a direct object `OBJ`, the source of the dependency for `CLIT` *en* is transferred from the verb to the object and its label becomes `dep`.

The lexicalization of the rule concerns the negative constraint. It aims at verifying that `V` is not a verb simultaneously taking a direct object and an indirect object introduced with *de*. In this case, *en* would be the `de_obj` complement of the verb `V`.

3.4 Modules for Controlling the Parsing Process

If we put all rewriting rules of FRDEP-PARSE in a unique bag with the same priority, the system is untractable because of the ambiguity, which will entail an time explosion of the parsing process. But GREW offers the possibility of grouping rules by modules and ordering the modules.

We use this possibility for controlling the pars-

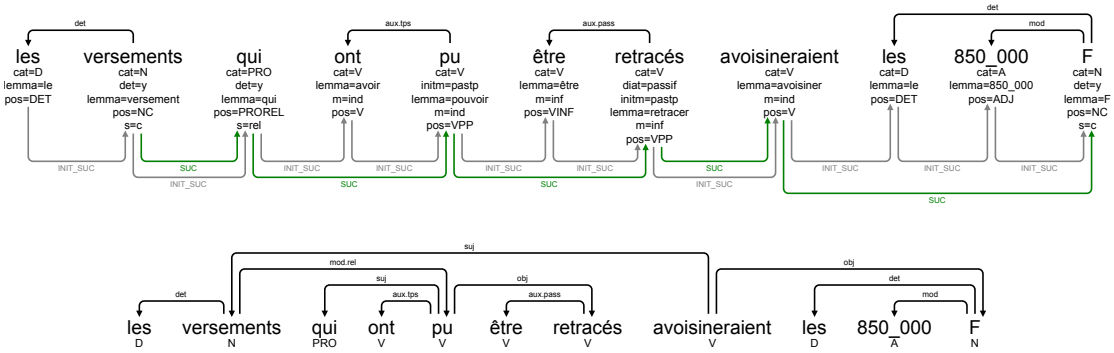

Figure 3: The annotation of a sentence after running the initial modules and at the end of parsing.

ing process. The rewriting rules model particular grammatical rules of the language (French in our case) and they are grouped by modules according to their linguistic proximity. We distinguish three classes of modules with respect to their position in the parsing process: initial modules, iterative modules and final modules.

3.4.1 Initial modules

Initial modules are carried out at the beginning of parsing. They have two different functions. Some modules, which are related to the specific annotation format, are used to prepare the actual syntactic annotation.

Other modules have a linguistic function: they realize close dependencies to verbs, nouns, adjectives and adverbs. They realize them with a great determinism, and thus they can be very well foreseen. They link verbs with their auxiliaries and clitics, nouns with their determiner and left adjectives. Finally, they make specific adverbs modifiers of verbs, adjectives or other adverbs.

Figure 3 shows an example of annotation step produced with FRDEP-PARSE on a part of Sentence *[frwiki_50.1000_00854]* from the SEQUOIA corpus. The first annotation is produced by the application of the initial modules. The module verb_aux realizes the dependencies `aux.tps` and `aux.pass` of the respective verbs *pu* and *retracés*. The module noun_dep realizes the dependencies `det` and `mod` for the noun *F* but also the dependency `det` for the noun *versements*.

3.4.2 Iterative modules

Iterative modules realize arguments of verbs, adjectives, nouns and adverbs, as well as their modifiers that can be put more or less far from them. They are called iterative because they can be repeated several times, which is made necessary by the CKY form of the syntactic composition induced by the form of rules and the recursivity of the syntactic composition for natural languages.

In the bottom part of Figure 3, the `suj`, `obj` and `mod.rel` dependencies of the second annotation are realized by iterative modules. In particular, the subject module is used twice: first, it realizes the subject *qui* of *pu*; second, it realizes the subject *versements* of *avoisineraient*. With a CKY strategy, both dependencies cannot be realized in the same application of the module subject because *versements* must be composed with the relative clause that modifies it, before being composed with the verb *avoisineraient*.

There is a specific coord module, which is dedicated to coordination. It is also iterative because coordination may be performed at more or less deep levels of syntax: word, noun phrase, clause...

3.4.3 Final modules

Final modules are carried out at the end of parsing. There are three classes of final modules.

In the order of their execution, the first class includes modules that realize disambiguation and transformation of projective trees into non-projective trees (see Subsection 3.3).

Then, a second class includes modules that aim at closing the dependency structure of the sentence in two ways: adding default dependencies between non-connected partial trees and removing relations that are not syntactic dependencies but that were used by FRDEP-PARSE in intermediate steps.

Finally, a last class of modules transforms the annotation in a format conform to the annotation scheme of SEQUOIA. The reason is that our

linguistic choices of annotation differ from those made for annotating SEQUOIA on some points.

First, we aim at simplifying the system of rewriting rules with a more uniform representation of dependencies. For instance, all contractions between a preposition and a determiner (for instance *à le* is contracted in *au*) are decomposed, so that rules related to determiners or prepositions can apply to these cases in the same way as in the standard cases.

Second, for some phenomena, there are good arguments for two different interpretations and we have made another choice than SEQUOIA. For instance, we consider that the head of a coordination is the conjunction of coordination. The main argument is that it allows the modifiers or arguments of a coordination to be distinguished from those of the first conjunct, which is not possible if we choose the head of the first conjunct as the head of the coordination.

3.5 Relaxing the CKY strategy

The goal of grouping rewriting rules by modules and ordering these ones, is the efficiency of parsing. The challenge is to keep accuracy at the same time. For this, we can play with two factors:

- the delimitation of modules (between two extremes, all rules in one module and one separate module for each rule),
- the order between modules, taking into account that iterative modules can be repeated as many times as needed,

If we constrain all iterative rules to respect the form described in Subsection 3.2, which induces a CKY strategy of parsing, we cannot obtain some needed dependency structures: for most iterative modules, the constraints imposed by the French grammar are contradictory.

Let us take again Example 1 to illustrate this contradiction. The first dependency `obj` from *coupé* to *ruban* must be realized after the dependency `mod.rel`, which entails the order head_verb_obj / head_noun_modrel between the corresponding modules. At the opposite, the second dependency `obj` from *marque* to *entrée* must be realized before the dependency `mod.rel`, which entails the order head_noun_modrel / head_verb_obj .

The contradiction cannot be solved by iteration of the module head_verb_obj because both dependencies will be realized at the first pass through the module.

We choose to relax the CKY strategy, which allows to link two partial dependency trees only by their roots. From empirical considerations, we propose new rules in the following form: if the yield of a partial dependency tree T_1 immediately precedes the yield of another partial dependency tree T_2, the rule tries to realize a dependency between the rightmost token wr_1 of the T_1 yield with the root of T_2.

This new kind of rules aims at implementing a strategy of parsing that gives priority to closest dependencies, contrary to the rules implementing a CKY-like strategy aiming at linking heads of dependency trees. We call the first rules *close linking rules* and the second ones *head linking rules*. The corresponding modules are respectively called *close linking modules* and *head linking modules*.

As for the head linking rules, the adjacency between T_1 and T_2 in a close linking rule is expressed with a `SUC` relation between their roots r_1 and r_2. The difference is that an `INIT_SUC` relation expresses the immediate succession between wr_1 and the left border wl_2 of the T_2 yield. The token wl_2 is linked to r_2 by an explicit chain of dependencies, which may reduce to the empty chain. The rule introduces a dependency from wr_1 to r_2.

Let us illustrate this new kind of rules with Example 1. Using only head linking rules, we obtain the annotation shown on Figure 4. We fail to link the word *ruban* with the head *marque* of the relative clause because the concerned module head_noun_modrel, the function of which is to realize `mod.rel` dependencies, comes after the module head_verb_obj realizing the dependency `obj` from *coupé* to *ruban*.

To solve the problem, we introduce the following close linking rule:

```
1  rule close-noun_modrel_verb {
2    match {
3      % positive constraints on T1
4      PRE [];
5      N [cat=N|PRO];
6
7      % positive constraints on T2
8      PROREL [pos=PROREL];
9      V [cat=V, m=ind|subj];
10     POST [];
11     V -> PROREL;
12     e2: V -[SUC]-> POST;
13
14     %adjacency between T1 and T2
15     e1: PRE -[SUC]-> V;
16
17     % N rightmost token in T1 yield
18     N -[INIT_SUC]-> PROREL
19   }
```

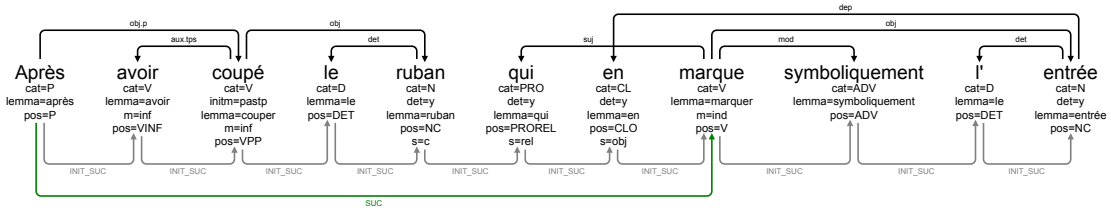

Figure 4: The annotation of the phrase from Figure 1 with the CKY strategy

```
20
21    % PROREL leftmost token in T2 yield
22    without {
23      DEP []; V -> DEP;
24      DEP.position < PROREL.position
25    }
26    without { V -> N}
27
28    commands {
29      del_edge e1;
30      del_edge e2;
31      add_edge N -[mod.rel]-> V;
32      add_edge PRE -[SUC]-> POST;
33    }
34  }
```

The rule links a noun N with the head verb V of a relative clause in a mod.rel dependency, expressing the modification of the noun by the relative clause.

The token PRE is a node of T_1, without any specific features. N is also a node of T_1 but the relationship between PRE and T_1 is not specified. Some constraints will be added further to make PRE the root of T_1 and N the rightmost token of its yield. The tree T_2 is made of the verb V governing a relative pronoun PROREL.

Any rule from an initial or iterative module preserves the following property for the SUC relation: the two tokens linked by the relation are roots and they have adjacent yields. This entails that PRE and V are roots of respectively T_1 and T_2 and their yields are adjacent.

As the initial and iterative modules preserve projectivity, the negative constraints express that PROREL is the leftmost token wl_2 of the T_2 yield.

Since the relation INIT_SUC expresses the adjacency of two tokens, therefore PROREL is the immediate successor of N. As PROREL is the leftmost token of the T_2 yield and the T_1 yield immediately precedes the T_2 yield, it entails that the rightmost token wr_1 of the T_1 yield is N.

The rewriting commands link the head V of the relative clause with its antecedent N in a mod.rel relation and they update the SUC relations.

As the experimental results shows it in the next section, both head linking and close linking strategies capture most configurations of syntactic dependencies but they fail to parse such phrases as the following one extracted from *[fr-wiki_50.1000_00464]*: *des listings de comptes de la chambre de compensation luxembourgeoise*. If we ignore agreement constraints, the head linking strategy leads to binding of the adjective *luxembourgeoise* with the head *listings* of the dependency tree of *des listings de comptes de la chambre de compensation*. With the close linking strategy, binding is realized with the closest leaf *compensation*. Both solutions are wrong because *luxembourgeoise* is a modifier of *chambre*. We need a more flexible strategy to cover all different cases of linking between two partial dependency trees.

4 Experimental results

The gold standard data used for evaluation is SEQUOIA version 6.0, which contains 3,099 French sentences taken from various sources (newspaper, medical texts, Europarl and Wikipedia). The full corpus was divided in two homogeneous subcorpora (DEV-SEQUOIA and TEST-SEQUOIA) of the same size. We used the DEV-SEQUOIA corpus to develop and to improve the rule system; the final evaluation reported below being done on the other part TEST-SEQUOIA.

The input of our rule-based system FRDEP-PARSE are sentences which are tokenized, lemmatized and tagged with the refined system of 28 pos labels defined in (Candito et al., 2011). In order to evaluate the FRDEP-PARSE rule system alone, we have made a first experiment (called GOLD-POS) where the input data are taken from the gold corpus: we consider the tokenization, lemmatization and enriched POS of SEQUOIA version 6.0 as input. The second experiment tries to evaluate our proposal in a more realistic setting where no gold tagging is available. In this second case, we first use a French tagger to build the input of out system from the raw test. Our tests are based on MElt (Denis and Sagot, 2012), so we call this sec-

ond experiment MELT-POS.

In SEQUOIA, there is no clear rules in the annotation guide which explain how punctuation sign should be linked to the rest of the sentence; hence, the punctuation is not annotated in a consistent way. Here, we report scores on the Corpus TEST-SEQUOIA without taking into account the relation punctuation; nevertheless, comma used as a coordination in enumeration are annotated with relation `coord` and `dep.coord` and so there are included in the evaluation. In this context, the LAS (Labelled Attachment Score) corresponds to what is usually called the recall (*i.e.* the proportion of relations in the reference corpus which are correctly predicted by the system). The objective of FRDEP-PARSE is not to build complete dependency parse and when it is not sensible to compute a relation with a deterministic rewrite rule, only partial dependency structures are returned and some lexical units are left unattached. This explain why the precision (*i.e.* the proportion of relations predicted by the system which are correct) is much higher than the recall.

	LAS(recall)	prec.	F-measure
GOLD	80.61%	89.21%	84.69%
MELT	76.04%	85.96%	80.69%

Figure 5: Experimental results

The TEST-SEQUOIA corpus contains 1,550 sentences and 33,662 lexical units. This corpus is parsed in 51.9 seconds (with a 2.4GHz Intel Core i7) and 32,035 relations are produced; this corresponds to a mean of 617 relations produced per second.

We provide below some comparison with other dependency parsers evaluated on French.

The talismane parser (Urieli, 2013) is a statistical parsed where some rules based on linguistic knowledge can be used to guide the parser. For the same reason we gave above about punctuation, Talismane is evaluated on the set of dependencies different from the punctuation. On a corpus similar to the one we used, the LAS (labelled attachment score) ranges from 86% to 88%.

In (Villemonte De La Clergerie, 2014), some experiments on the SEQUOIA corpus are also reported. Again, results are given without taking into account the punctuation. Different combination of parsers are tested on different sub-corpora; the LAS scores obtained range from 83.53% to 88.94%.

The scores we obtain with FRDEP-PARSE are significantly lower than the two other systems but we still consider them as very encouraging. Indeed, the FRDEP-PARSE system was written in a few weeks and is the first attempt to write of a set of graph rewriting rules for dependency parsing. Moreover, we make two very strong restrictions on our system: we consider only deterministic application of rules and we do not use any statistical information. Relaxing these restrictions is far from being an easy task. If we consider non-deterministic uses of Graph Rewriting in the general setting, we will necessarily have to deal with exponential number of solutions. It will be needed to use statistical information to guide the graph rewriting process and to avoid exponential explosion. This challenge is part of the future work to do in this area.

5 Conclusion

In this paper, we have presented a first attempt to build a dependency parser in the framework of Graph Rewriting. Despite severe restrictions on the rule system, we obtain a LAS-score of 80% if the POS-tagging is taken from the Gold standard corpus and 76% if we use MElt as the POS-tagger. These results are lower than state-of-the-art approaches based on combination of statistical and symbolic dependency parsing but we think that there are many possible ways of improvement from this first attempt; for instance, using non-deterministic graph rewriting rules together with a selection system based on statistical information taken from a parsed corpus is one of the future plan for improving our system.

References

Guillaume Bonfante, Bruno Guillaume, Mathieu Morey, and Guy Perrier. 2011. Modular Graph Rewriting to Compute Semantics. In Johan Bos and Stephen Pulman, editors, *9th International Conference on Computational Semantics - IWCS 2011*, pages 65–74, Oxford, United Kingdom, January.

Marie Candito and Djamé Seddah. 2012. Le corpus Sequoia : annotation syntaxique et exploitation pour l'adaptation d'analyseur par pont lexical. In *Proc. of TALN*, Grenoble, France.

Marie Candito, Benoît Crabbé, and Mathieu Falco, 2011. *Dépendances syntaxiques de surface pour le français (version 1.2)*.

Michael A. Covington. 2000. A fundamental algorithm for dependency parsing. In *In Proceedings of the 39th Annual ACM Southeast Conference*, pages 95–102.

Ralph Debusmann, Denys Duchier, and Geert-Jan Kruijff. 2004. Extensible dependency grammar: A new methodology. In *Recent Advances in Dependency Grammars*, August.

Pascal Denis and Benoît Sagot. 2012. Coupling an annotated corpus and a lexicon for state-of-the-art POS tagging. *Language Resources and Evaluation*, 46(4):721–736.

Jason M. Eisner. 1996. Three new probabilistic models for dependency parsing: An exploration. In *Proceedings of the 16th Conference on Computational Linguistics - Volume 1*, COLING '96, pages 340–345. Association for Computational Linguistics.

Kilian Foth, Ingo Schröder, and Wolfgang Menzel. 2000. A transformation-based parsing technique with anytime properties. In *in proceedings of the 4th International Workshop on Parsing Technologies*, pages 89–100.

Joakim Nivre. 2003. An efficient algorithm for projective dependency parsing. In *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT*, pages 149–160.

Kemal Oflazer. 2003. Dependency parsing with an extended finite-state approach. *Computational Linguistics*, 29:515–544.

Assaf Urieli. 2013. *Robust French syntax analysis: reconciling statistical methods and linguistic knowledge in the Talismane toolkit*. Ph.D. thesis, Université de Toulouse II le Mirail.

Éric Villemonte De La Clergerie. 2014. Jouer avec des analyseurs syntaxiques. In *TALN 2014*, Marseilles, France, July. ATALA.