Modular Graph Rewriting to Compute Semantics

Guillaume Bonfante
Nancy-Université - LORIA
bonfante@loria.fr

Bruno Guillaume
INRIA - LORIA
guillaum@loria.fr

Mathieu Morey
Nancy-Université - LORIA
moreymat@loria.fr

Guy Perrier
Nancy-Université - LORIA
perrier@loria.fr

Abstract

Taking an asynchronous perspective on the syntax-semantics interface, we propose to use modular graph rewriting systems as the model of computation. We formally define them and demonstrate their use with a set of modules which produce underspecified semantic representations from a syntactic dependency graph. We experimentally validate this approach on a set of sentences. The results open the way for the production of underspecified semantic dependency structures from corpora annotated with syntactic dependencies and, more generally, for a broader use of modular rewriting systems for computational linguistics.

Introduction

The aim of our work is to produce a semantic representation of sentences on a large scale using a formal and exact approach based on linguistic knowledge. In this perspective, the design of the syntax-semantics interface is crucial.

Based on the compositionality principle, most models of the syntax-semantics interface use a synchronous approach: the semantic representation of a sentence is built step by step in parallel with its syntactic structure. According to the choice of the syntactic formalism, this approach is implemented in different ways: in a Context-Free Grammars (CFG) style framework, every syntactic rule of a grammar is associated with a semantic composition rule, as in the classical textbook by Heim and Kratzer (1998); following the principles introduced by Montague, Categorial Grammars use an homomorphism from the syntax to the semantics (Carpenter (1992)). HPSG integrates the semantic and syntactic representations in feature structures which combine by unification (Copestake et al. (2005)). LFG follows a similar principle (Dalrymple (2001)). In a synchronous approach, the syntax-semantics interface closely depends on the grammatical formalism. Building such an interface can be very costly, especially if we aim at a large coverage for the grammar.

In our work, we have chosen an asynchronous approach in the sense that we start from a given syntactic analysis of a sentence to produce a semantic representation. With respect to the synchronous approach, a drawback is that the reaction of the semantics on the syntax is delayed. On the other hand, the computation of the semantics is made relatively independent from the syntactic formalism. The only constraint is the shape of the output of the syntactic analysis.

In the formalisms mentioned above, the syntactic structure most often takes the form of a phrase structure, but the choice of constituency for the syntax makes the relationship with the semantics more complicated. We have chosen *dependency graphs*, because syntactic dependencies are closely related to predicate-argument relations. Moreover, they can be enriched with relations derived from the syntax, which are usually ignored, such as the arguments of infinitives or the anaphora determined by the syntax. One may observe that our syntactic representation of sentences involves plain graphs and not trees. Indeed, these relations can give rise to multiple governors and dependency cycles. On the semantic side,

we have also chosen graphs, which are widely used in different formalisms and theories, such as DMRS (Copestake (2009)) or MTT (Mel'čuk (1988)) .

The principles being fixed, our problem was then to choose a model of computation well suited to transforming syntactic graphs into semantic graphs. The λ-calculus, which is widely used in formal semantics, is not a good candidate because it is appropriate for computing on trees but not on graphs. Our choice naturally went to *graph rewriting*. Graph rewriting is barely used in computational linguistics; it could be due to the difficulty to manage large sets of rules. Among the pioneers in the use of graph rewriting, we mention Hyvönen (1984); Bohnet and Wanner (2001); Crouch (2005); Jijkoun and de Rijke (2007); Bédaride and Gardent (2009); Chaumartin and Kahane (2010).

A graph rewriting system is defined as a set of *graph rewrite rules* and a computation is a sequence of rewrite rule applications to a given graph. The application of a rule is triggered via a mechanism of pattern matching, hence a sub-graph is isolated from its context and the result is a local modification of the input. This allows a linguistic phenomenon to be easily isolated for applying a transformation.

Since each step of computation is fired by some local conditions in the whole graph, it is well known that one has no grip on the sequence of rewriting steps. The more rules, the more interaction between rules, and the consistency of the whole rule system becomes difficult to maintain. This bothers 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. For instance, in our proposal, there is a module transforming the syntactic arguments of verbs, predicative nouns and adjectives into their semantic arguments. Another module resolves the anaphoric links which are internal to the sentence and determined by the syntax.

From a computational point of view, the grouping of a small number of rules inside a module allows some optimizations in their application, thus leading to efficiency. For instance, the confluence of rewriting is a critical feature — one computes only one normal form, not all of them — for the performance of the program. Since the underlying relation from syntax to semantics is not functional but relational, the system cannot be globally confluent. Then, it is particularly interesting to isolate subsets of confluent rules. Second point, with a small number of rules, one gets much more control on their output. In particular, it is possible to automatically infer some invariant properties of graphs along the computation within a particular module. Thus, it simplifies the writing of the rules for the next modules. It is also possible to plan a strategy in the global evaluation process.

It is well known that syntactic parsers produce outputs in various formats. As a by-product of our approach, we show that the choice of the input format (that is the syntax) seems to be of low importance overall. Indeed, as far as two formats contain the same linguistic information with different representations, a system of rewrite rules can be designed to transform any graph from one format to another as a preliminary step. The same remark holds for the output formats.

To illustrate our proposal, we have chosen the *Paris7 TreeBank* (hereafter *P7TB*) dependency format defined by Candito et al. (2010) as the syntactic input format and the *Dependency MRS* format (hereafter *DMRS*) defined by Copestake (2009) as the semantic output format. We chose those two formats because the information they represent, if it is not complete, is relatively consensual and because both draw on large scale experiments: statistical dependency parsing for French[1] on the one hand and the DELPH-IN project[2] on the other hand.

Actually, in our experiments, since we do not have an appropriate corpus annotated according to the *P7TB* standard, we used our syntactic parser LEOPAR[3] whose outputs differ from this standard and we designed a rewriting system to go from one format to the other.

The paper is organized as follows. In section 1, we define our graph rewriting calculus, the β-calculus. In Section 2, we describe the particular rewriting system that is used to transform graphs from the syntactic *P7TB* format into the *DMRS* semantic format. In Section 3, we present experimental results on a test suite of sentences.

[1] http://alpage.inria.fr/statgram/frdep/fr_stat_dep_parsing.html

[2] http://www.delph-in.net/

[3] http://leopar.loria.fr

1 The β-calculus, a graph rewriting calculus

Term rewriting and tree rewriting can be defined in a straightforward and canonical way. Graph rewriting is much more problematic and there is unfortunately no canonical definition of a graph rewriting system. Graph rewriting can be defined through a categorical approach like SPO or DPO (Rozenberg (1997)). But, in practice, it is much easier to use a more operational view of rewriting where modification of the graph (the "right-hand side" of a rule) is defined by means of a set of commands; the control of the way rules are applied (the "left hand-side") still uses pattern matching as this is done in traditional graph rewriting.

In this context, a *rule* is a pair of a *pattern* and a sequence of *commands*. We give below the formal materials about graphs, patterns, matchings and commands. We illustrate the section with examples of rules and of rewriting.

1.1 Graph definition

In the following, we suppose given a finite set $\mathcal{L}$ of edge labels corresponding to the kind of dependencies used to describe sentences. They may correspond to syntax or to semantics. For instance, we use $\mathcal{L} = \{\text{SUJ}, \text{OBJ}, \text{ARG}1, \text{ANT}, \ldots\}$.

To decorate vertices, we use the standard notion of feature structures. Let $\mathcal{N}$ be a finite set of *feature names* and $\mathcal{A}$ be a finite set of *atomic feature values*. In our example, $\mathcal{N} = \{\text{cat}, \text{mood}, \ldots\}$ and $\mathcal{A} = \{\text{passive}, \text{v}, \text{n}, \ldots\}$. A *feature* is a pair made of a feature name and a set of atomic values. The feature $(cat, \{v, aux\})$ means that the feature name cat is associated to either the value v or aux. In the sequel, we use the notation $cat = v|aux$ for this feature. Two features $f = v$ and $f' = v'$ are compatible whenever $f = f'$ and $v \cap v' \neq \emptyset$.

A *feature structure* is a finite set of features such that each feature name occurs at most once. $\mathcal{F}$ denotes the set of feature structures. Two feature structures are compatible if their respective features with the same name are pairwise compatible.

A graph $\mathcal{G}$ is then defined by a 6-uple $(\mathcal{V}, \mathbf{fs}, \mathcal{E}, \mathbf{lab}, \sigma, \tau)$ with:

- a finite set $\mathcal{V}$ of vertices;
- a labelling function $\mathbf{fs}$ from $\mathcal{V}$ to $\mathcal{F}$;
- a finite set $\mathcal{E}$ of edges;
- a labelling function $\mathbf{lab}$ from $\mathcal{E}$ to $\mathcal{L}$;
- two functions σ and τ from $\mathcal{E}$ to $\mathcal{V}$ which give the source and the target of each edge.

Moreover, we require that two edges between the same couple of nodes cannot have the same label.

1.2 Patterns and matchings

Formally, a *pattern* is a graph and a *matching* ϕ of a pattern $\mathcal{P} = (\mathcal{V}', \mathbf{fs}', \mathcal{E}', \mathbf{lab}', \sigma', \tau')$ into a graph $\mathcal{G} = (\mathcal{V}, \mathbf{fs}, \mathcal{E}, \mathbf{lab}, \sigma, \tau)$ is an *injective* graph morphism from $\mathcal{P}$ to $\mathcal{G}$. More precisely, ϕ is a couple of injective functions: $\phi_{\mathcal{V}}$ from $\mathcal{V}'$ to $\mathcal{V}$ and $\phi_{\mathcal{E}}$ from $\mathcal{E}'$ to $\mathcal{E}$ which:

- respects vertex labelling: $\mathbf{fs}(\phi_{\mathcal{V}}(v))$ and $\mathbf{fs}'(v)$ are compatible;
- respects edge labelling: $\mathbf{lab}(\phi_{\mathcal{E}}(e)) = \mathbf{lab}'(e)$;
- respects edge sources: $\sigma(\phi_{\mathcal{E}}(e)) = \phi_{\mathcal{V}}(\sigma'(e))$;
- respects edge targets: $\tau(\phi_{\mathcal{E}}(e)) = \phi_{\mathcal{V}}(\tau'(e))$.

1.3 Commands

Commands are low-level operations on graphs that are used to describe the rewriting of the graph within a rule application. In the description below, we suppose to be given a pattern matching $\phi : \mathcal{P} \to \mathcal{G}$. We describe here the set of commands which we used in our experiment so far. Naturally, this set could be extended.

- **del_edge**(α, β, ℓ) removes the edge labelled ℓ between α and β. More formally, we suppose that $\alpha \in \mathcal{V}_\mathcal{P}$, $\beta \in \mathcal{V}_\mathcal{P}$ and $\mathcal{P}$ contains an edge e from α to β with label $\ell \in \mathcal{L}$. Then, **del_edge**$(\alpha, \beta, \ell)(\mathcal{G})$ is the graph $\mathcal{G}$ without the edge $\phi(e)$. In the following, we give only the intuitive definition of the command: thanks to injectivity of the matching ϕ, we implicitly forget the distinction between x and $\phi(x)$.
- **add_edge**(α, β, ℓ) adds an edge labelled ℓ between α and β. Such an edge is supposed not to exist in $\mathcal{G}$.
- **shift_edge**(α, β) modifies all edges that are incident to α: each edge starting from α is moved to start from β; similarly each edge ending on α is moved to end on β;
- **del_node**(α) removes the α node in $\mathcal{G}$. If $\mathcal{G}$ contains edges starting from α or ending on α, they are silently removed.
- **add_node**(β) adds a new node with identifier β (a fresh name).
- **add_feat**$(\alpha, f = v)$ adds the feature $f = v$ to the node α. If α already contains a feature name f, it is replaced by the new one.
- **copy_feat**(α, β, f) copies the value of the feature named f from the node α to the node β. If α does not contain a feature named f, nothing is done. If β already contains a feature named f, it is replaced by the new value.

Note that commands define a partial function on graphs: the action **add_edge**(α, β, ℓ) is undefined on a graph which already contains an edge labelled ℓ from α to β.

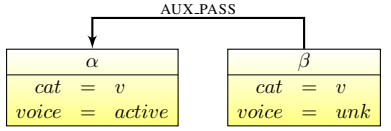
The action of a sequence of commands is the composition of actions of each command. Sequences of commands are supposed to be consistent with the pattern:

- **del_edge** always refers to an edge described in the pattern and not previously modified by a **del_edge** or a **shift_edge** command;
- each command refers only to identifiers defined either in the pattern or in a previous **add_node**;
- no command refers to a node previously deleted by a **del_node** command.

Finally, we define a *rewrite rule* to be a pair of a pattern and a consistent sequence of commands.

A first example of a rule is given below with the pattern on the left and the sequence of commands on the right. This rule called INIT_PASSIVE is used to remove the node corresponding to the auxiliary of the passive construction and to modifiy the features accordingly.

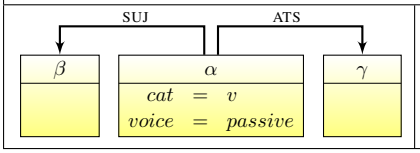
INIT_PASSIVE



$c_1 =$ **copy_feat**$(\alpha, \beta, mood)$
$c_2 =$ **copy_feat**$(\alpha, \beta, tense)$
$c_3 =$ **add_feat**$(\beta, voice = passive)$
$c_4 =$ **del_edge**$(\beta, \alpha, \text{AUX\_PASS})$
$c_5 =$ **shift_edge**(α, β)
$c_6 =$ **del_node**(α)

Our second example (PASSIVE_ATS) illustrates the **add_node** command. It is used in a passive contruction where the semantic subject of the verb is not realized syntactically.

PASSIVE_ATS

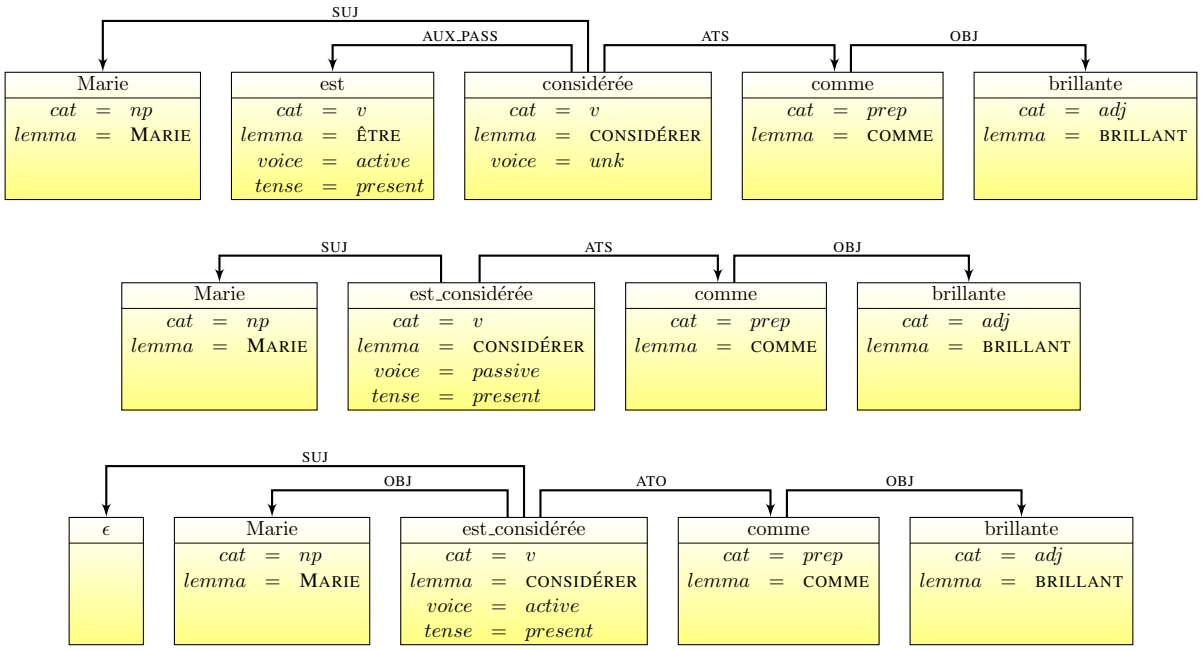


$c_1 = \textbf{del\_edge}(\alpha, \beta, \text{SUJ})$
$c_2 = \textbf{add\_edge}(\alpha, \beta, \text{OBJ})$
$c_3 = \textbf{del\_edge}(\alpha, \gamma, \text{ATS})$
$c_4 = \textbf{add\_edge}(\alpha, \gamma, \text{ATO})$
$c_5 = \textbf{add\_feat}(\alpha, voice = active)$
$c_6 = \textbf{add\_node}(\delta)$
$c_7 = \textbf{add\_edge}(\alpha, \text{SUJ}, \delta)$

1.4 Rewriting

We consider a graph $\mathcal{G}$ and a rewrite rule $r = (\mathcal{P}, [c_1, \ldots, c_k])$. We say that $\mathcal{G}'$ is obtained from $\mathcal{G}$ by a rewrite step with the r rule (written $\mathcal{G} \longrightarrow_r \mathcal{G}'$) if there is a matching morphism $\phi : \mathcal{P} \rightarrow \mathcal{G}$ and $\mathcal{G}'$ is obtained from $\mathcal{G}$ by applying the composition of commands $c_k \circ \ldots \circ c_1$.

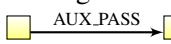
Let us now illustrate two rewrite steps with the rules above. Consider the first graph below which is a syntactic dependency structure for the French sentence *"Marie est considérée comme brillante"* [*Mary is considered as bright*]. The second graph is obtained by application of the INIT_PASSIVE rewrite rule and the last one with the PASSIVE_ATS rewrite rule.



1.5 Modules and normal forms

A *module* contains a set of rewrite rules but, in order to have a finer control on the output of these modules, it is useful to declare some forbidden patterns. Hence a module is defined by a set $\mathcal{R}$ of rules and a set $\mathcal{P}$ of forbidden patterns.

For a given module $\mathcal{M} = (\mathcal{R}, \mathcal{P})$, we say that $\mathcal{G}'$ is an *$\mathcal{M}$-normal form* of the graph $\mathcal{G}$ if there is a sequence of rewriting steps with rules of $\mathcal{R}$ from $\mathcal{G}$ to $\mathcal{G}'$: $\mathcal{G} \longrightarrow_{r_1} \mathcal{G}_1 \longrightarrow_{r_2} \mathcal{G}_2 \ldots \longrightarrow_{r_k} \mathcal{G}'$, if no rule of $\mathcal{R}$ can be applied to $\mathcal{G}'$ and no pattern of $\mathcal{P}$ matches in $\mathcal{G}'$.

In our experiment, forbidden patterns are often used to control the subset of edges allowed in normal forms. For instance, the *NORMAL* module contains the fordidden pattern: 

. Hence, we can then safely suppose that no graph contains any AUX_PASS edge afterward.

2 From syntactic dependency graphs to semantic graphs

Linguistic theories diverge on many issues including the exact definition of the linguistic levels and the relationships between them. Our aim here is not to commit to any linguistic theory but rather to

demonstrate that graph rewriting is an adequate and realistic computational framework for the syntax-semantics interface. Consequently, our approach is bound to neither the (syntactic and semantic) formats we have chosen nor the transformation modules we have designed; both are mainly meant to exemplify our proposal.

2.1 Representational formats

Our syntactic and semantic formats both rely on the notion of linguistic dependency. The syntactic format is an enrichment of the one which was designed to annotate the French Treebank (Abeillé and Barrier (2004)) with surface syntactic dependencies (Candito et al. (2010)). The enrichment is twofold:

- if they are present in the sentence, the deep arguments of infinitives and participles (from participial subordinate clauses) are marked with the usual labels of syntactic functions,
- the anaphora relations that are predictable from the syntax (i.e. the antecedents of relative, reflexive and repeated pronouns) are marked with a special label ANT.

This additional information can already be provided by many syntactic parsers and is particularly interesting to compute semantics.

The semantic format is Dependency Minimal Recursion Semantics (*DMRS*) which 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 ARGi, where i is an integer following a fixed order of obliqueness SUJ, OBJ, ATS, ATO, A-OBJ, DE-OBJ.... Naturally, the lexicon must be consistent with this ordering. The restrictions of generalized quantifiers are labelled RSTR ; their bodies are not overtly expressed but can be retrieved from the graph. There are three ways of combining predicates:

- EQ when two predicates are elements of a same conjunction;
- H when a predicate is in the scope of another predicate; it is not necessarily one of its arguments because quantifiers may occur between them;
- NEQ for all other cases.

2.2 Modular rewriting system

Graph rewriting allows to proceed step by step to the transformation of a syntactic graph into a semantic one, by associating a rewrite rule to each linguistic rule. While the effect of every rule is local, grouping rules in modules allows a better control on the global effect of all rules.

We do not have the space here to propose a system of rules that covers the whole French grammar. We however propose six modules which cover a significative part of this grammar (cleft clauses, coordination, enumeration, comparatives and ellipses are left aside but they can be handled by other rewrite modules):

- *NORMAL* handles the regular syntactic transformations involving predicates: it computes tense and transforms all redistributions of arguments (passive and middle voices, impersonal constructions and the combination of them) to the active canonical form. This reduces the number of rules required to produce the predicate-argument relations in the *ARG* module below.
- *PREP* removes affixes, prepositions and complementizers.
- *ARG* transforms the verbal, nominal and adjectival predicative phrases into predicate-argument relations.

- *DET* translates the determiner dependencies (denoted DET) to generalized quantifiers.
- *MOD* interprets the various modifier dependencies (denoted MOD), according to their specificity: adjectives, adverbs, adjunct prepositional phrases, participial clauses, relative clauses, adjunct clauses.
- *ANA* interprets all anaphoric relations that are determined by the syntax (denoted ANT).

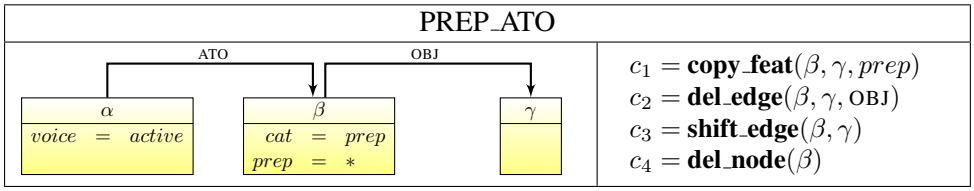
Modules provide an easy way to control the order in which rules are fired. In order to properly set up the rules in modules, we first have to fix the global ordering of the modules. Some ordering constraints are evident: for instance, *NORMAL* must precede *PREP*, which must precede *ARG*. The rules we present in the following are based on the order *NORMAL*, *PREP*, *ARG*, *DET*, *MOD*, *ANA*.

2.2.1 Normalization of syntactic dependencies

The *NORMAL* module has two effects: it merges tense and voice auxiliaries with their past participle and brings all the argument redistributions back to the canonical active form. This module accounts for the passive and middle voices and the impersonal construction for verbs that are not essentially impersonal. The combination of the two voices with the impersonal construction is naturally expressed by the composition of the corresponding rewrite rules. The two rules given in section 1.4 are part of this module. The first rule (INIT_PASSIVE) merges the past participle of the verb with its passive auxiliary. The auxiliary brings its mood and tense to the verb, which is marked as being passive. The second rule (PASSIVE_ATS) transforms a passive verb with a subject and an attribute of the subject into its active equivalent with a semantically undetermined subject, an object (which corresponds to the subject of the passive form) and an attribute of the object (which corresponds to the attribute of the subject of the passive form).

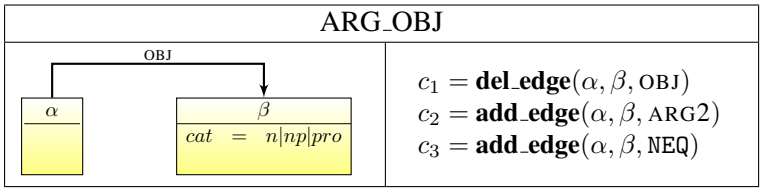
2.2.2 Erasure of affixes, prepositions and complementizers

The *PREP* module removes affixes, prepositions and complementizers. For example, the rule given here merges prepositions with the attribute of the object that they introduce. The value of the preposition is kept to compute the semantics.



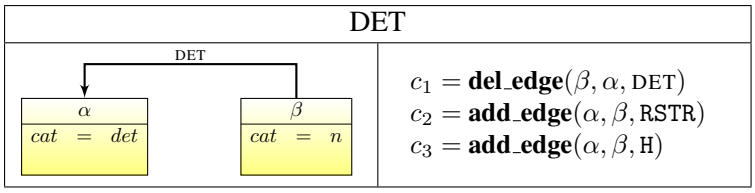
2.2.3 From lexical predicative phrases to semantic predicates

The *ARG* module transforms the syntactic arguments of a predicative word (a verb, a common noun or an adjective) into its semantic arguments. Following *DMRS*, the predicate-argument relations are not labelled with thematic roles but only numbered. The numbering reflects the syntactic obliqueness.



2.2.4 From determiners to generalized quantifiers

DET reverts the determiner dependencies (labelled DET) from common nouns to determiners into dependencies of type `RSTR` from the corresponding generalized quantifier to the nominal predicate which is the core of their restriction.



2.2.5 Interpretation of different kinds of modification

MOD deals with the modifier dependencies (labelled MOD, MOD_REL and MOD_LOC), providing rules for the different kinds of modifiers. Adjectives and adverbs are translated as predicates whose first argument is the modified entity. The modifier and modified entities are in a conjunction (EQ), except for scopal adverbs which take scope (H) over the modified predicate. Because only lexical information enables to differentiate scopal from non-scopal adverbs, we consider all adverbs to be systematically ambiguous at the moment. Adjunct prepositional phrases (resp. clauses) have a similar rule except that their corresponding predicate is the translation of the preposition (resp. complementizer), which has two arguments: the modified entity and the noun (resp. verb) which heads the phrase (resp. clause). Participial and relative clauses exhibit a relation labelled EQ or NEQ between the head of the clause and the antecedent, depending on the restrictive or appositive type of the clause.

2.2.6 Resolution of syntactic anaphora

ANA deals with dependencies of type ANT and merges their source and their target. We apply them to reflexive, relative and repeated pronouns.

3 Experiments

For the experimentation, we are interested in a test suite which is at the same time small enough to be manually validated and large enough to cover a rich variety of linguistic phenomena. As said earlier, we use the P7 surface dependency format as input, so the first attempt at building a test suite is to consider examples in the guide which describes the format. By nature, an annotation guide tries to cover a large range of phenomena with a small set of examples.

The latest version[4] of this guide (Candito et al. (2010)) contains 186 linguistic examples. In our current implementation of the semantic constructions, we leave out clefts, coordinations and comparatives. We also leave out a small set of exotic sentences for which we are not able to give a sensible syntactic structure. Finally, our experiment runs on 116 French sentences. Syntactic structures following P7 specifications are obtained with some graph rewriting on the output of our parser. Each syntactic structure was manually checked and corrected when needed. Then, graph rewriting with the modules described in the previous section is performed.

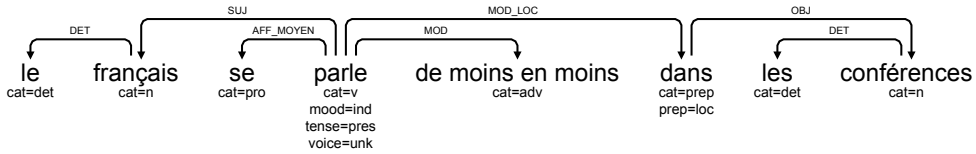
For all of these sentences, we produce at least one normal form. Even if *DMRS* is underspecified, our system can output several semantic representations for one syntactic structure (for instance, for appositive and restrictive relative clauses). We sometimes overgenerate because we do not use lexical information like the difference between scopal and non-scopal adverbs.

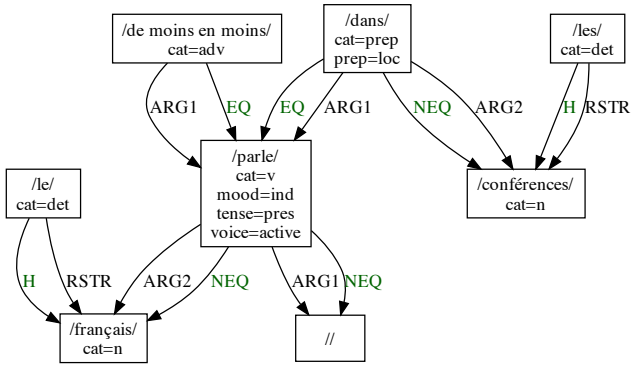
The result for three sentences is given below and the full set is available on a web page [5].

[4] version 1.1, january 2010

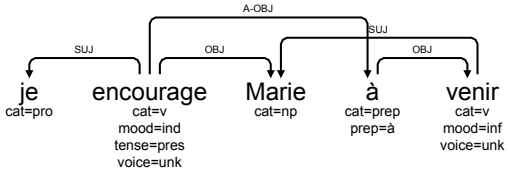
[5] http://leopar.loria.fr/doku.php?id=iwcs2011

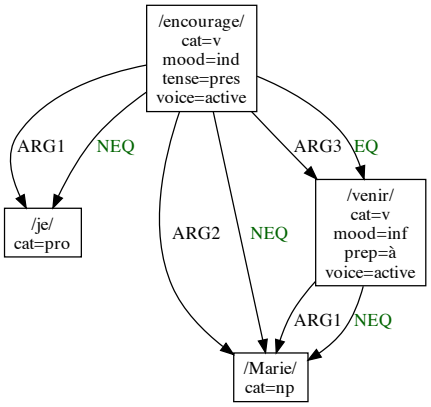
[012] *"Le français se parle de moins en moins dans les conférences."* [*The French language is less and less spoken in conferences.*]



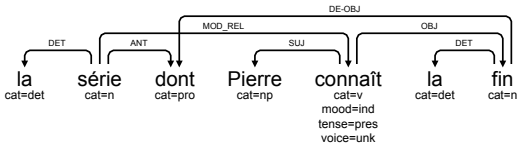


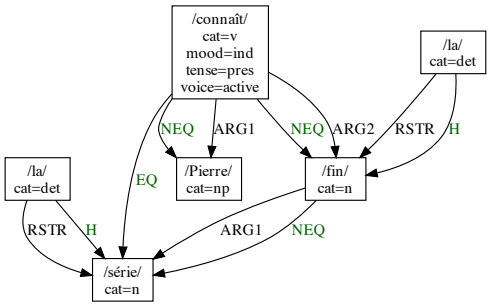
[057] *"J'encourage Marie à venir."* [*I invite Mary to come.*]





[106] *"La série dont Pierre connaît la fin"* [*The story Peter knows the end of*]





Conclusion

In this paper, we have shown the relevance of modular graph rewriting to compute semantic representations from graph-shaped syntactic structures. The positive results of our experiments on a test suite of varied sentences make us confident that the method can apply to large corpora.

The particular modular graph rewriting system presented in the paper was merely here to illustrate the method, which can be used for other input and output formats. There is another aspect to the flexibility of the method: we may start from the same system of rules and enrich it with new rules to get a finer semantic analysis — if *DMRS* is considered as providing a minimal analysis — or integrate lexical information. The method allows the semantic ambiguity to remain unsolved within underspecified representations or to be solved with a rule system aiming at computing models of underspecified representations. Moreover, we believe that its flexibility makes graph rewriting a convenient framework to deal with idiomatic expressions.

References

Abeillé, A. and N. Barrier (2004). Enriching a french treebank. In *Proceedings of LREC*.

Bédaride, P. and C. Gardent (2009). Semantic Normalisation : a Framework and an Experiment. In *Proceedings of IWCS*, Tilburg Netherlands.

Bohnet, B. and L. Wanner (2001). On using a parallel graph rewriting formalism in generation. In *Proceedings of EWNLG '01*, pp. 1–11. Association for Computational Linguistics.

Candito, M., B. Crabbé, and P. Denis (2010). Statistical french dependency parsing: Treebank conversion and first results. *Proceedings of LREC2010*.

Candito, M., B. Crabbé, and M. Falco (2010). *Dépendances syntaxiques de surface pour le français*.

Carpenter, B. (1992). *The logic of typed feature structures*. Cambridge: Cambridge University Press.

Chaumartin, F.-R. and S. Kahane (2010). Une approche paresseuse de l'analyse sémantique ou comment construire une interface syntaxe-sémantique à partir d'exemples. In *TALN 2010, Montreal, Canada*.

Copestake, A. (2007). Semantic composition with (robust) minimal recursion semantics. In *Proceedings of the Workshop on Deep Linguistic Processing*, pp. 73–80. Association for Computational Linguistics.

Copestake, A. (2009). *Invited Talk:* Slacker semantics: Why superficiality, dependency and avoidance of commitment can be the right way to go. In *Proceedings of EACL 2009*, Athens, Greece, pp. 1–9.

Copestake, A., D. Flickinger, C. Pollard, and I. Sag (2005). Minimal Recursion Semantics - an Introduction. *Research on Language and Computation 3*, 281–332.

Crouch, D. (2005). Packed Rewriting for Mapping Semantics to KR. In *Proceedings of IWCS*.

Dalrymple, M. (2001). *Lexical Functional Grammar*. New York: Academic Press.

Heim, I. and A. Kratzer (1998). *Semantics in generative grammar*. Wiley-Blackwell.

Hyvönen, E. (1984). Semantic Parsing as Graph Language Transformation - a Multidimensional Approach to Parsing Highly Inflectional Languages. In *COLING*, pp. 517–520.

Jijkoun, V. and M. de Rijke (2007). Learning to transform linguistic graphs. In *Second Workshop on TextGraphs: Graph-Based Algorithms for Natural Language Processing, Rochester, NY, USA*.

Mel'čuk, I. (1988). *Dependency Syntax: Theory and Practice*. Albany: State Univ. of New York Press.

Rozenberg, G. (Ed.) (1997). *Handbook of Graph Grammars and Computing by Graph Transformations, Volume 1: Foundations*. World Scientific.