HAL open science

A Linguistically-motivated 2-stage Tree to Graph Transformation

Corentin Ribeyre, Djamé Seddah, Éric Villemonte de La Clergerie

To cite this version:

Corentin Ribeyre, Djamé Seddah, Éric Villemonte de La Clergerie. A Linguistically-motivated 2-stage Tree to Graph Transformation. TAG+11 - The 11th International Workshop on Tree Adjoining Grammars and Related Formalisms - 2012, INRIA, Sep 2012, Paris, France. hal-00765422

HAL Id: hal-00765422

https://inria.hal.science/hal-00765422v1

Submitted on 14 Dec 2012

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

A Linguistically-motivated 2-stage Tree to Graph Transformation

Corentin Ribeyre[⋆] **Djamé Seddah**[⋆,◇] **Eric Villemonte de la Clergerie**[⋆]
[⋆]Alpage, INRIA Université Paris Diderot
[◇] Université Paris Sorbonne
firstname.lastname@inria.fr

Abstract

We propose a new model for transforming dependency trees into target graphs, relying on two distinct stages. During the first stage, standard local tree transformation rules based on patterns are applied to collect a first set of constrained edges to be added to the target graph. In the second stage, motivated by linguistic considerations, the constraints on edges may be used to displace them or their neighbour edges upwards, or to build new mirror edges. The main advantages of this model is to simplify the design of a transformation scheme, with a smaller set of simpler local rules for the first stage, and good properties of termination and confluence for the second level.

1 Introduction

Tree transformation is emerging as an important and recurring operation in Natural Language Processing, for instance to transform shallow syntactic trees into deeper ones or into semantic graphs (Bonfante et al., 2011a), or to transform syntactic trees from a source language to a target one (in Machine Translation). Another frequent case that initially motivated this work is the transformation from a source dependency scheme to a target one, in order to conduct parsing evaluations or to be used in some post-parsing component based on the target schema.

Two main problems arise when transforming linguistic structures. The first one is related to the diversity of syntactic configurations that have to be identified, knowing that many of these configurations are rare. A large number of rules may have to be provided to cover this diversity. A second problem arises from the non locality of some linguistic constructions, for instance for retrieving the true subject of some controlled verbs at semantic level or in case of coordination. Even when bounding these phenomena, trying to combine them with more canonical cases may lead to an explosion of the number of transformation rules, difficult to create and maintain. And, when not bounding these non local phenomena, it becomes necessary to introduce recursive transformation rules that raise delicate problems of ordering when applying them, as presented in the Grew system (Bonfante et al., 2011b) based on graph rewriting rules.

Many approaches have been proposed for tree or graph transformations, such as Top-Down or Bottom-Up Tree Transducers (Courcelle and Engelfriet, 2012), Tree-Walking Transducers (Bojańczyk, 2008), Synchronous Grammars (Shieber and Schabes, 1990) and (Matsuzaki and Tsujii, 2008) for an application on annotation scheme conversion, or Graph Rewriting Systems based, for instance, on the Single PushOut model (SPO) (Löwe et al., 1993; Geiss et al., 2006). But either they are complex to implement or they suffer from the above mentioned problems (coverage, maintenance, ordering). Moreover, they are not always suited for natural language processing, especially in case of complex phenomena.

Based on a preliminary experiment of scheme to scheme transformation, but motivated by more generic linguistic considerations, we propose a simple new two stage model. The first stage essentially addresses the local syntactic phenomena through local tree transformation rules whose action is to add a set of edges to the target graph being built. By focusing on local phenomena, fewer and simpler rules are needed. Furthermore, it is relatively easy to learn these local rules from a set of example sentences with their syntactic trees, and some partial annotation provided by the transformation designer. These local rules may easily

be expressed using the SPO model.

The main novelty is the possibility for the first stage rules to decorate the target edges with constraints. During the second stage, the constraints are essentially used to displace edges in the target graph, either the edge carrying a constraint or neighbour edges, in the direction of "*heads*" (*upward direction*). This formulation is clearly in phase with the propagation of information to handle non-local phenomena. It has also the advantage of offering good properties of termination, with no new edge created and the fact that edges can not climb up forever. However, we add a more problematic class of constraints, used to duplicate some edges, for instance to handle sharing phenomena at the semantic level (control, coordination).

The first stage being a rather standard case of SPO-based (Single Push-Out) transformation (Löwe et al., 1993), we will focus, in Section 2, on a preliminary formalization of the second stage constraint-based transformation (Ribeyre, 2012). The expressive power of the approach will be illustrated with a few complex syntactic constructions in Section 3. The conversion experiment that have initially triggered the use of constraint is presented in Section 4.

2 Constraint-based graph transformation

We assume as given a graph domain $\mathcal{G}$ where the transformations take place. A component of $\mathcal{G}$ is a set of edge labels $\mathcal{L}$, partitioned in $\mathcal{L}_1 \cup \cdots \mathcal{L}_n$, the intuition being that each $\mathcal{L}_i$ corresponds to a subset of labels for a specific dimension (for instance, a dimension for the set of labels used as thematic roles and another one for quantifiers or for anaphora as illustrated in Figure 7).

Let e denote an edge $x \xrightarrow{l} y$, with source node x, target node y and label $l \in \mathcal{L}$.

A node y for a graph $G \in \mathcal{G}$ can have several incoming edges $e = x \xrightarrow{l} y$ but only one per dimension (i.e., $\forall e_1 = x_1 \xrightarrow{l_1} y, e_2 = x_2 \xrightarrow{l_2} y, \exists k,\ l_1 \in \mathcal{L}_k \wedge l_2 \in \mathcal{L}_k \Longrightarrow e_1 = e_2$). This condition can be a bit restrictive, so we relax it by allowing several incoming edges for the same dimension, but tagging all but one as derived edges of the main one. In other words, the derived edges will be seen as clones of the main one. For a given edge e, we note $\dim(e) = k$ the dimension of e such that $l \in \mathcal{L}_k$.

Let an extended edge be $e = (x \xrightarrow{l} y, C, H)$, which carries a (possibly) empty set of constraints C to be detailed below and a (possibly empty) history list H formed of node pairs (x', y') retracing the changes of head x' and tail y'.

A configuration (G, Agenda) for the constraint-based transformation includes a graph $G \in \mathcal{G}$ and an agenda of edges to be progressively added to G. The *initial configuration* has an empty graph, while a *terminal configuration* has an empty agenda. The initial agenda provides a list of edges to add returned by the first stage based on local transformation rules.

At each step i, an edge e is selected and removed from the agenda Agenda_i and added to G_i to get $G'_i = G_i \cup \{e\}$. Because of constraints on e or on edges e' in direct contact with e, e or e' may be removed from G'_i to get G_{i+1} and new derived edges with updated history added to the agenda to get Agenda_{i+1}. The process stops with success when reaching a terminal configuration, or with failure when getting a conflict in a graph G'_i, for instance when a node gets two main incoming edges for a same dimension k, or when a cycle is detected in an edge history (i.e., an edge is moved back to a previous place).

We consider four kinds of constraints that may be carried by the edges:

- A **move up** $m\uparrow$ constraint on edge e may be used to move e upwards, as illustrated by Figure 1[1]. The displacement is controlled by an argument pair $(\mathcal{A}, q)$ where $\mathcal{A}$ is a deterministic finite-state automaton (DFA) and q a state for $\mathcal{A}$. The DFA represents all possible transitions for the constraint to move up.[2] We further impose that all transition labels of $\mathcal{A}$ are edge labels in the same $\mathcal{L}_k$ for some dimension k. For an automaton example, see the Figure 8.[3] Intuitively, the constrained edge e can only move upwards along k-paths, and when several k-edges are possible from a node, the main one is chosen.

- A **redirect up** $r\uparrow$ constraint on edge $e =$

[1] where the green edges are removed from the graph while the red ones are added to the agenda.

[2] This is reminiscent of LFG's functional uncertainty equations (Kaplan and Zaenen, 1995).

[3] Actually, weaker constraints on the automata labels seem to be possible.

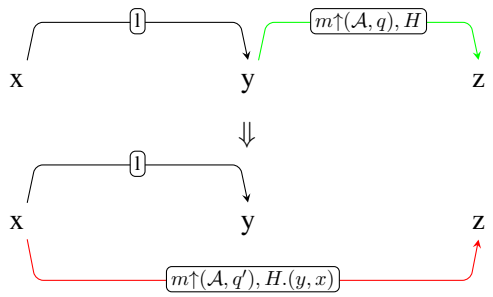

Figure 1: **move up** constraint

$x \xrightarrow{l_e} y$ may be used to move upwards the edges governed by y (the outgoing edges of y), as illustrated by Figure 2. The constraint accepts an argument $\mathbf{L} \subset \mathcal{L}_k$ for $k = \dim(e)$ that restricts the redirection to edges $e' = y \xrightarrow{l} z$ with some label $l \in \mathbf{L}$.

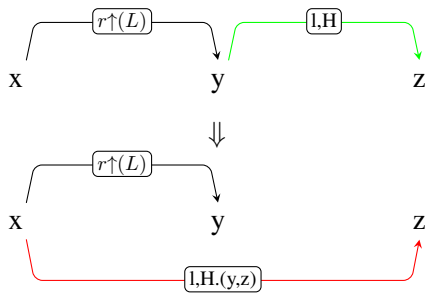

Figure 2: **redirect up** constraint

- A **share up** $s{\uparrow}$ constraint on edge $e = y \xrightarrow{l_e} z$ may be used to duplicate all incoming edges $e' = x \xrightarrow{l} y$ on y as incoming edges on z, as illustrated by Figure 3. Like the $r{\uparrow}$ constraint, the $s{\uparrow}$ constraint accepts an argument $\mathbf{L} \subset \mathcal{L}_{\dim(e)}$ that restricts the duplication to edges e' with label $l \in \mathbf{L}$.

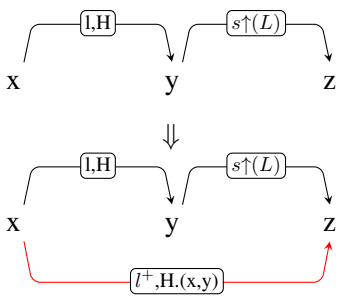

Figure 3: **share up** constraint

- A **share down** $s{\downarrow}$ constraint on edge $e = y \xrightarrow{l_e} z$ may be used to duplicate all outgoing edges of y as outgoing edges of z, as illustrated by Figure 4. The $s{\downarrow}$ constraint accepts an argument $\mathbf{L} \subset \mathcal{L}_{\dim(e)}$ that restricts the duplication to edges e' with label $l \in \mathbf{L}$. Note that the resulting edges have tagged labels l^+, indicating they are secondary edges.

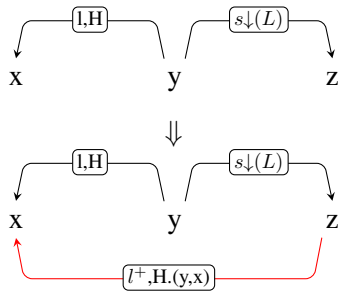

Figure 4: **share down** constraint

We impose further restrictions to handle the interactions between constraints. For instance, the edges with **share** constraints can not be redirected and the edges with **move up** constraints can not be duplicated through the application of a **share down** constraint. The definition of the graph domain and the control provided by the constraint arguments ensure the confluence of the rewriting process. The number of edges can not grow, but through the application of the **share** constraints. By controlling that we do not add an edge twice to the same place (thanks to the history information) and because all movements are oriented upwards, termination may be ensured. Some conflicts may be solved through the use of meta-rules as illustrated in Section 4.

After termination, a cleaning phase may be applied to delete some edges that were added as temporary helping edges, for instance some edges with **share** constraints.

Intuitively, the choice of constraint types is motivated by the idea that movement is driven by heads, with a component moving upward until it finds its place as an head (**move up**) or with components redirected upwards until they attach to the right head (**redirect up**). The **share** constraints are used to deal with coordination (with for instance a subject shared by several verbs), but also but other elliptic constructions as illustrated by control verbs.

In practice, this set of constraints seems to be

sufficient for most interesting situations. However, new kinds of constraints, such as the obvious **move down** and **redirect down**, and the blocking constraints **freeze up** and **freeze down**, should be investigated, in terms of interest and in terms of coherence with the other kinds of constraints (in particular to preserve the confluence).

More interestingly, we are also considering a **transfer** constraint, a generalization of the **share down** constraint that could be used to handle complex cases of edge transfer in repeated elliptic coordination, as illustrated in *Paul wants to eat an apple, Mary a pear, and John an orange.*, where we need some form of copy for *eat* and *want*, and of transfer for the subject and object grammatical functions.

3 A linguistically motivated model

We believe that our approach has a great potential and can be applied to solve in elegant ways various transformations problems. The next section will show how it can be used to handle conversion between dependency schemes, but the current section focuses on its use for transforming shallow syntactic trees into deeper ones or even into shallow semantic graphs as explored in (Ribeyre, 2012). Our motivation is to reach a semantic level with a small set of rules and reduced human cost. A first step in this direction is to induce the triggering patterns of the local transformation rules (based on the SPO approach (Löwe et al., 1993)) by partially annotating some nodes and/or edges in the parse trees of a set of carefully selected sample sentences grouped in sets illustrating the various syntactic phenomena and their configuration.

Given the annotations and the parse trees, the algorithm basically tries to generalize over the selected nodes and edges, through the following steps:

1. Extract a graph from the annotations of the first annotated parse tree
2. Extract a graph from the annotations of the second one
3. Find the maximum common subgraph(s) between these graphs
4. If there is more than one common subgraph, find the most general subgraph by comparing the features structures attached to the nodes and edges

The algorithm is actually pretty simple, but seems to be powerful enough in most cases.[4] In fact, it provides the possibility of quickly developing a set of rules for a particular application, because most users prefer to select something rather than writing code from scratch. That is the reason why we developed a GUI as part of our Graph Rewriting System. Figure 3 provides a screenshot of the interface, which is divided in three parts:

1. A hierarchical view (left hand panel) where one can manage the set of sentence examples, grouping them by syntactic phenomena;
2. The tree view (right hand panel) where one can select nodes and edges (the red dotted lines);
3. The triggering part (bottom panel) of the induced transformation rule, to be then edited and completed by the transformation part.

The examples described below are annotated with the CoNLL scheme used for the dependency version of the French Treebank (FTB) (Candito et al., 2010a; Abeillé et al., 2003). Our goal is to construct a new version of the FTB with deeper syntactic annotations, as a first step towards a shallow semantic representation for FTB. Hence, in the figures 6, 7 and 9, we illustrate some complex syntactic constructions and try to exhibit simple transformations, using the constraints. In the examples, we use the following color code:

- Red edges for the final edges after all constraint applications,
- Green edges for the initial constrained edges,
- Blue edges for non constrained edges used by constrained ones
- Dotted orange edges for intermediary temporary edges

For instance, in Figure 6, we would like to insert the missing link between a deep subject and

[4] but we are aware of its limits: for instance, unless adding negative examples, it is not possible to induce tests on the non existence of an edge.

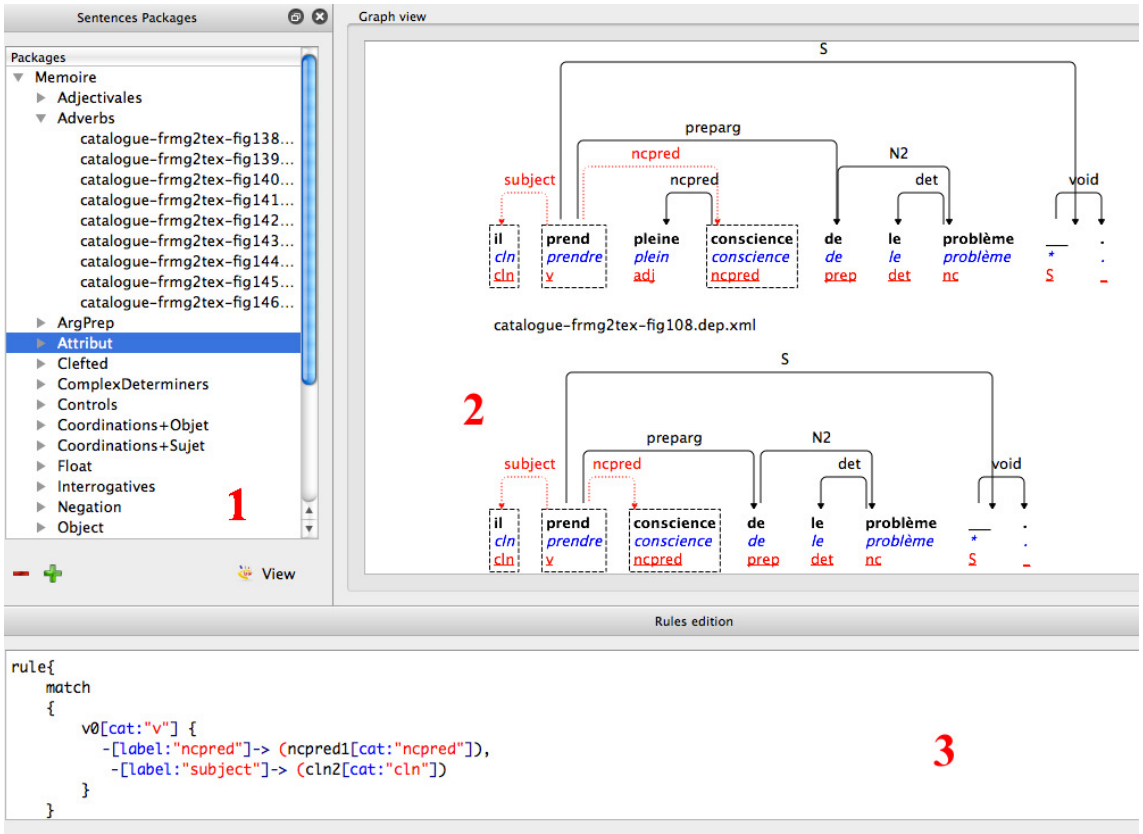

Figure 5: GUI of the Graph Rewriting System

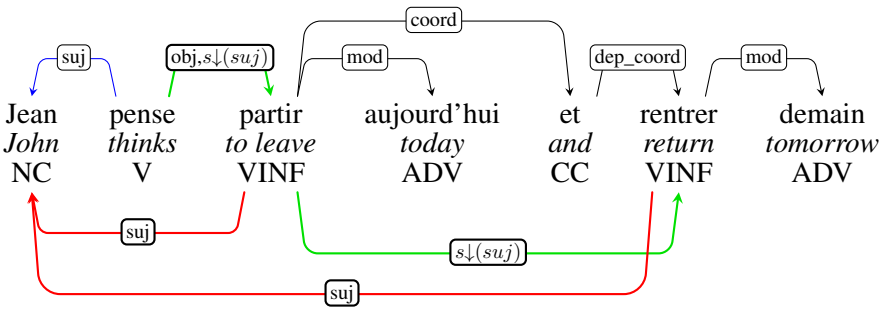

Figure 6: Subject ellipsis + control verb

its verb in the case of subject ellipsis in the sentence: *Jean*$_1$ *pense* ϵ_1 *partir aujourd'hui et* ϵ_1 *rentrer demain* (*John*$_1$ *thinks about* ϵ_1 *leaving today and* ϵ_1 *coming back tomorrow*). This example is interesting because of the subject ellipsis and the control verb *penser*. We need to add the following **subject** dependencies, derived from the subject dependency between *Jean* and *pense* :

1. between *Jean* and *partir*, because *Jean* is also subject of *partir*
2. between *Jean* and *rentrer*, because *Jean* is the elliptical subject of *rentrer*

To solve our problem, we simply need to put two **share down** constraints as illustrated in Figure 6. The first constrained edge between *pense* and *partir* results from a rule dealing with subject-controlled verbs. The second constrained edge between *partir* et *renter* results from a rule dealing with coordination between clauses with no subject in the last clause. After applying the constraints, we get the 2 extra subject dependencies (in red). Of course, we can solve these two problems with the two following local rules:

1. One for solving the control verb issue.

2. One for solving the subject ellipsis case.

But, in that case, we may observe that the first rule has to be applied before the second rule, imposing some ordering between the rules. So, the confluence will not be guaranteed by the system.

The system of constraints has been designed to be easy to use but nevertheless expressive enough for powerful transformations. Our second example, in Figure 7, illustrates the use of a **move up** constraint. In this example, we want to retrieve the antecedent "*Jean*" of the relative pronoun "*dont*". We have to follow a potentially unbounded chain of dependencies starting from "*dont*" until we reach a `mod_rel` dependency. In order to do that, the original `de_obj` dependency between *dont* and *mère* (*mother*) triggers the addition of an initial `ant` dependency between the same words (in green) but with a `move up` constraint built on the automaton $\mathcal{A}$ of Figure 8. The constrained edge will then move up following the `obj` (green) edges staying in state q_0 of $\mathcal{A}$ (orange `ant` edges). Finally, the edge moves up through the `mod_rel` edge, switching to the final state q_1 (red `ant` edge). One can note that *dont* has several heads, namely the syntactic head *mère* and an head for the **ant** relation. Typically, the **ant** relation will be present in some new dimension for anaphora.

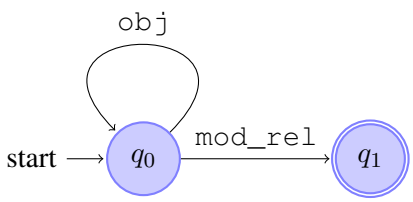

Figure 8: Automaton $\mathcal{A}$ for the **move up** constraint in Figure 7

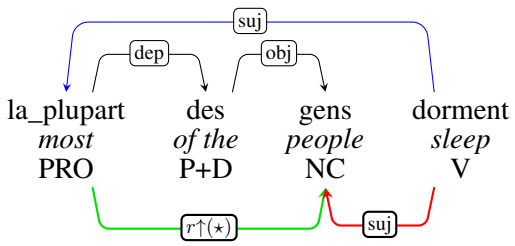

Figure 9: Linguistic application of constraint **redirect**

Finally, Figure 9 illustrates the *redirect up* constraint. In the sentence, "*la plupart*" (*most*) is the shallow syntactic subject of "*dorment*" (*sleep*). However the semantic subject of "*dorment*" is actually "*gens*" (*people*). So we add a redirect edge from *la plupart* to *gens* that can redirect all edges entering *la plupart* towards *gens*, including the subject dependency. The same mechanism would work as well in sentences such as *il parle à la plupart des enfants.* (*he talks to most of the kids*), with a redirection of an `a_obj` dependency.

4 A use case: a scheme to scheme conversion

The formalization sketched in this paper is a derivative of a preliminary experiment of conversion between two syntactic dependency schemes. The source schema **depFRMG** is a rich and deep dependency schema produced from the output of FRMG, a French wide coverage hybrid TAG/TIG parser resulting from the compilation of a metagrammar (Villemonte de La Clergerie, 2005). The target schema **depFTB** is the dependency version of the French TreeBank (Candito et al., 2010a; Abeillé et al., 2003), and the choice was initially motivated by evaluation purposes for FRMG.

The **depFRMG** schema, represented using the DepXML format[5], is produced by converting the TAG derivations returned by FRMG, using the elementary tree anchors as heads for lexicalized trees and introducing pseudo anchors for non lexicalized trees (using their root category as label) (Villemonte de la Clergerie, 2010). The resulting dependencies are non necessarily projective, for instance in the case of superlative constructions. The **depFRMG** schema may actually be used to represent a shared forest of dependency trees representing the whole set of analysis returned by FRMG for a sentence. In practice, a phase of disambiguation is applied to return the best dependency tree as illustrated by the above edges[6] in Figure 10 for the following sentence:

[5]There are often some confusions, but one should clearly distinguish the notions of *format* such as DepXML or CONLL, *model* to specify the structures and their properties (such as projectivity or shared forests), and, finally, *schema* as an instantiation of a model for a specific resource, associated with a tagset and annotation guidelines. Here, we are really interested by a conversion between two schema.

[6]The edge color indicates the kind of the underlying TAG operation, with blue for substitution, red for adjoining, and purple for co-anchoring and lexical nodes in a tree. The S node corresponds to a case of pseudo-anchor for a non-lexicalized elementary tree.

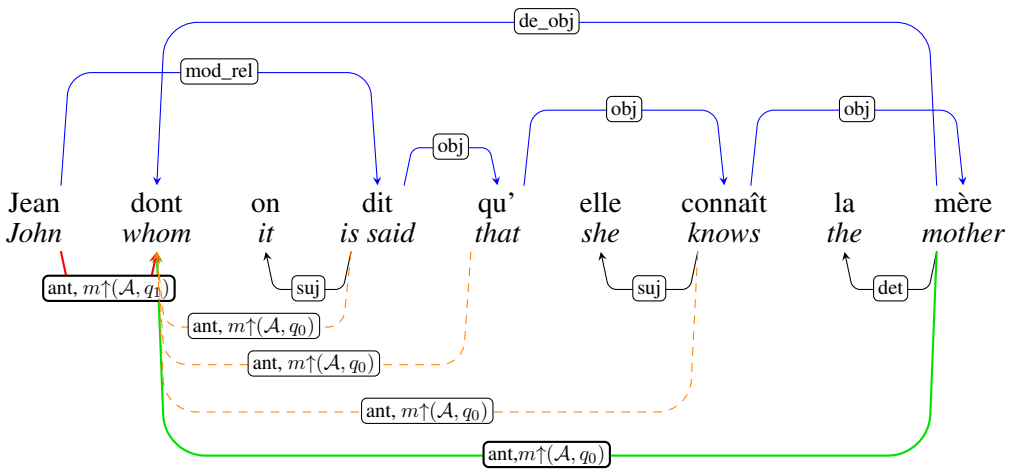

Figure 7: Linguistic application of constraint **move**

par qui a-t-elle voulu que ces deux livres
by whom did-she want that these two books
et ce DVD lui soient rendus ?
and this DVD to-her be returned ?

The **depFTB** schema used for the dependency version of the FTB is expressed in CONLL format, a format largely used for training statistical parsers and evaluating them (Nivre et al., 2007). The schema corresponds to projective shallow dependencies using a relatively small numbers of dependency labels. Figure 10 shows a **depFTB** version of our illustrative sentence, as produced by the conversion, with the converted edges below the sentence.

Most cases of conversion are straightforward and may be handled by local rules (for instance for attaching determiners to nouns with `det`, adjective on nouns with `mod`, or "object" introduced by prepositions with `obj`). However, we also already observe non obvious modifications in Figure 10, for instance, the root of the dependency tree is *rendu* for **depFRMG** (because of the argument extraction) while it is *voulu* for **depFTB**. The subject *-t-elle* is attached to the auxiliary *a* in **depFRMG** (because of subject inversion), but to the main verb *voulu* in **depFTB**. These more complex cases may involve potentially non-bounded propagations and changes of heads (with redirection for the dependants). It is also worth noting that our resulting conversion is non projective (because of the attachment of *par* on *rendu*), breaking the expected **depFTB** guidelines that would propose an attachment on *voulu*, but only for projectivization reasons that are considered as more and more questionable in such situations (so we decided not to do the projectivization).

In practice, we designed 86 local transformation rules, a few of them carrying constraints such as `move_up`, `redirect`, `frozen` (to block displacement), and `mirror` (a variant of the `share` constraints). Furthermore, a few meta-rules were added to handle conflicts, for instance when a word gets two distinct governors or when a dependency gets two distinct labels. An example of such mediating rule is given by the following: *if two dependencies d_1 and d_2 share the same target w and if $h(d_1) < h(d_2) < w$, then d_1 is rerooted to have $h(d_2)$ as target* (in other word, the closest preceding potential head wins). A mechanism of log was used to track, on corpus, the most frequent conflicts and check the effectiveness of the mediating rules.

The resulting conversion scheme (coupled with disambiguation tuning learned from the 9881 sentences of the train part `ftb6_1` of FTB) allows us to reach (with no prior tagging) a Labelled Attachment Score (LAS) of 85.8% on the test part `ftb6_3` of FTB (1235 sentences), not counting the punctuation dependencies (as usually done). This figure may be honorably compared to the 88.2% obtained by the best statistical parser directly trained on the FTB (Candito et al., 2010b). Even if it difficult to measure the impact of the conversion, it seems that the conversion is relatively good. Still, we are aware of some loss coming from the conversion, essentially due to phenomena that can not be directly handled by tree transformation and constraints. More precisely,

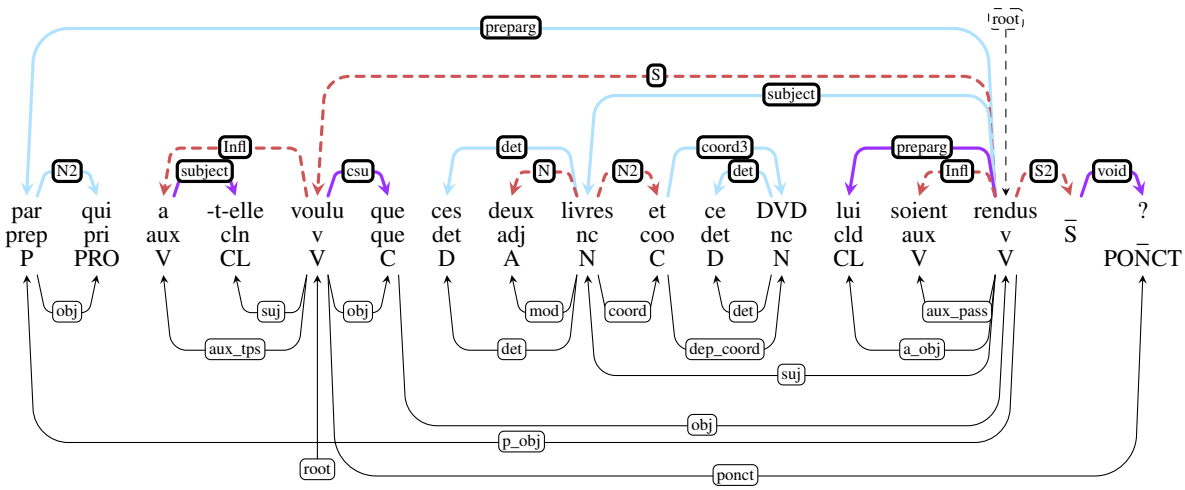

Figure 10: Disambiguated FRMG output, with **depFRMG** edges above and converted **depFTB** edges below

FRMG and FTB do not use the same set of compound words (for instance for complex prepositions, complex adverbs, or named entities) and it is therefore necessary to retrieve the missing **depFTB** dependencies for the FRMG compound words that are not compound in FTB.

If we look more closely at the importance of the constraints in the conversion process, we can observe that their impact is limited but associated with a potentially large number of occasionally rare configurations that would have been difficult to identify.

Only 5 rules out of the 86 rules carry constraints:[7]

- One rule (`R_ponct`) with a constraint `move up`, used to attach the final punctuation to the FTB head of the sentence.
- Two rules with a `redirect up` constraint. Rule `R_aux_caus` is used to handle the (rare) cases of causative constructions while Rule `R_Monsieur` is used to handle honorific construction such as *M. Teulade* (*Mr Teulade*) where *Teulade* would be the head for FRMG and *M.* the head for FTB.
- Two rules with a `mirror` constraint, used to handle enumerations.

[7]The small number of constrained rules may be partially explained by the fact that the need for constraints became progressively apparent after some rules were already written. It is possible that the set of 86 rules could be slightly reduced by adding constraints to more of them.

On the 278,083 dependencies present in `ftb6_1`, we get 5,352 `move up` resolutions (2%), 1,167 `redirect` resolutions (0.4%), and 638 `mirror` resolutions. However, these cases correspond to configurations involving 46 distinct pairs of rules for `move up`, 27 for `redirect`, and 24 for `mirror`. Intuitively, these figures provide a rough estimate of the number of rules (around 97) that should have been found and added to avoid the use of the constraints (uniquely on `ftb6_1`). It may be noted that this set of potential rules is already greater than the 86 current rules, with no guarantee of being complete.

We also have some indications about the resolution of the conflicts with 5,665 potential head conflicts (2%), 5,320 of them (94%) being handled by the mediating rules.

5 Conclusion

We have sketched a constraint-based graph transformation, motivated by linguistic considerations on the displacement or the sharing of dependencies. This constraint-based transformation may be used as the second stage of a transformation process based on more standard local transformation rules.

We have shown that a very small set of constraint types is sufficient to handle relatively complex syntactic phenomena in elegant ways. A few more types (to be investigated) could prove themselves very valuable to handle complex cases of elliptic coordination.

We believe that the two-stage approach is a

promising one, because of a better division of work and an economy both in terms of number of rules and of development time. The local transformation rules may be relatively easily learned from partially annotated simple examples, the second stage results from a few rules constrained with only a small set of constraint types. The constraints avoid many rule ordering issues related to recursive transformations, providing more stable systems (wrt the addition or modification of rules). Proof of the confluence is also easier.

A preliminary implementation was tried for a conversion process between two dependency schemes, but a cleaner implementation based on the formalization presented in this paper is underway. It will be tested on a larger spectrum of transformations, for instance to build semantic graphs from dependency trees.

References

Anne Abeillé, Lionel Clément, and François Toussenel. 2003. Building a treebank for French. In Anne Abeillé, editor, *Treebanks*. Kluwer, Dordrecht.

M. Bojańczyk. 2008. Tree-walking automata. In *International Conference on Language and Automata Theory and Applications*.

G. Bonfante, B. Guillaume, and M. Morey. 2011a. Modular graph rewriting to compute semantics. In *International Workshop on Computional Semantics 2011*.

G. Bonfante, B. Guillaume, M. Morey, and G. Perrier. 2011b. Enrichissement de structures en dépendances par réécriture de graphes. In *TALN 2011*.

Marie Candito, Benoît Crabbé, and Pascal Denis. 2010a. Statistical french dependency parsing: treebank conversion and first results. In *Proceedings of the 7th Language Resources and Evaluation Conference (LREC'10)*, La Valette, Malte.

Marie Candito, Joakim Nivre, Pascal Denis, and Enrique Henestroza Anguiano. 2010b. Benchmarking of statistical dependency parsers for french. In *Proceedings of COLING'2010 (poster session)*, Beijing, China.

B. Courcelle and J. Engelfriet. 2012. *Graph Structure and Monadic Second-Order Logic, a Language-Theoretic Approach*. Cambridge University Press.

R. Geiss, G. Veit Batz, D. Grund, S. Hack, and A. Szalkowski. 2006. GrGen: A fast SPO-based graph rewriting tool. In *International Conference on Graph Transformation*.

R.M. Kaplan and A. Zaenen. 1995. Long-distance dependencies, constituent structure, and functional uncertainty. *Formal Issues in Lexical-Functional Grammar*, 47:137–165.

M. Löwe, H. Ehrig, R. Heckel, L. Ribeiro, A. Wagner, and A. Corradini. 1993. Algebraic approches to graph transformations. *Theoritical Computer Science*.

T. Matsuzaki and J. Tsujii. 2008. Comparative parser performance analysis across grammar frameworks through automatic tree conversion using synchronous grammars. In *Proceedings of the 22nd International Conference on Computational Linguistics-Volume 1*, pages 545–552. Association for Computational Linguistics.

Joakim Nivre, Johan Hall, Sandra Kuebler, Ryan McDonald, Jens Nilsson, Sebastian Riedel, and Deniz Yuret. 2007. The CoNLL 2007 shared task on dependency parsing. In *The CoNLL 2007 shared task on dependency parsing*.

Corentin Ribeyre. 2012. Mise en place d'un systéme de réécriture de graphes appliqués á l'interface syntaxe-sémantique. M2, Univ. Paris Diderot 7, June.

Stuart M. Shieber and Yves Schabes. 1990. Synchronous tree-adjoining grammars. In *Proceedings of the 13th conference on Computational linguistics - Volume 3*, COLING '90, pages 253–258, Stroudsburg, PA, USA. Association for Computational Linguistics.

Éric Villemonte de La Clergerie. 2005. From metagrammars to factorized TAG/TIG parsers. In *Proceedings of IWPT'05 (poster)*, pages 190–191, Vancouver, Canada.

Éric Villemonte de la Clergerie. 2010. Convertir des dérivations TAG en dépendances. In ATALA, editor, *17e Conférence sur le Traitement Automatique des Langues Naturelles - TALN 2010*, July.