Non-simplifying Graph Rewriting Termination

Guillaume Bonfante, Bruno Guillaume

Non-simplifying Graph Rewriting Termination

Guillaume Bonfante
LORIA
Université de Lorraine

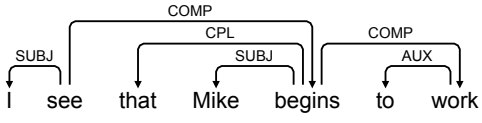
Bruno Guillaume
LORIA
Inria Nancy Grand-Est

So far, a very large amount of work in Natural Language Processing (NLP) rely on trees as the core mathematical structure to represent linguistic informations (e.g. in Chomsky's work). However, some linguistic phenomena do not cope properly with trees. In a former paper, we showed the benefit of encoding linguistic structures by graphs and of using graph rewriting rules to compute on those structures. Justified by some linguistic considerations, graph rewriting is characterized by two features: first, there is no node creation along computations and second, there are non-local edge modifications. Under these hypotheses, we show that uniform termination is undecidable and that non-uniform termination is decidable. We describe two termination techniques based on weights and we give complexity bound on the derivation length for these rewriting systems.

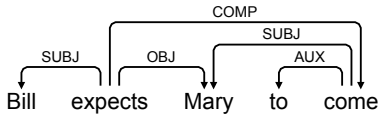
1 Introduction

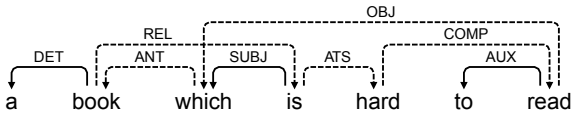
Linguists introduce different levels to describe a natural language sentence. Starting from a sentence given as a sequence of sounds or as a sequence of words; among the linguistic levels, two are deeply considered in literature: the syntactic level (a grammatical analysis of the sentence) and the semantic level (a representation of the meaning of the sentence). These two representations involve mathematical structures such as logical formulae, λ-terms, trees and graphs.

One of the usual ways to describe syntax is to use the notion of dependency [16]. A dependency structure is an ordered sequence of words, together with some relations between these words. For instance, the sentence *"I see that Mike begins to work"* can be represented by the structure on the right.



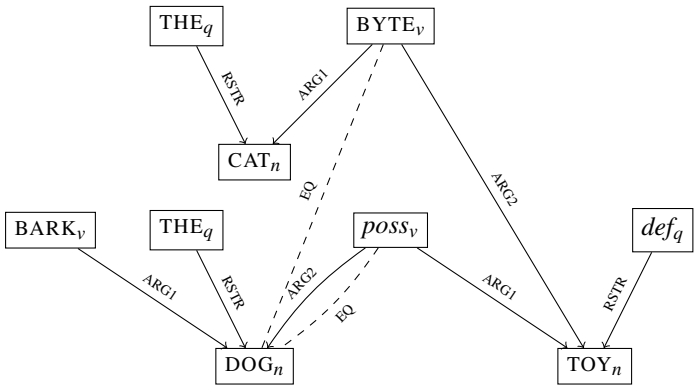
There is a large debate in the literature about the mathematical nature of the structures needed for natural language syntax: do we have to consider trees or graphs? Trees are often considered for their simplicity; however, it is clearly insufficient. Let us illustrate the limitations of tree-representations with some linguistic examples. Consider the sentence *"Bill expects Mary to come"*, the node *"Mary"* is shared, being the subject of *"come"* and the object of *"expects"* (below on the left). The situation can be even worse: cycles may appear such as in the sentence below where edges in the cycle are drawn with dashed line (below on the right).





To appear in EPTCS.

For the semantic representation of natural language sentences, first order logic formulae are widely used. To deal with natural language ambiguity, a more compact representation of a set of logic formulae (called underspecified semantic representation) is used. DMRS [4] is one of these compact representation. The DMRS structure for the sentence *"The Dog whose toy the cat bit barked"* is given in the figure on the right.



To describe transformations between syntactic and semantics structures, there are solutions based on many computational models (finite state automata, λ-calculus). It is somewhat surprising that Graph Rewrite Systems (GRS) have been hardly considered so far ([8, 1, 5, 9]). To explain that, GRS implementations are usually considered to be too inefficient to justify their extra-generality. For instance, pattern matching does not take linear time where this is usually seen as an upper limit for fast treatment.

However, if one drops for a while the issue of efficiency, the use of GRS is promising. Indeed, linguistic considerations can be most of the time expressed by some relations between a few words. Thus, they are easily translated into rules. To illustrate this point, in [3, 12], we proposed a syntax to semantics translator based on GRSs: given the syntax of a sentence, it outputs the different meaning associated to this syntax.

In the two earlier mentioned studies, we tried to delineate what are the key features of graph rewriting in the context of NLP. Roughly speaking, node creation are strictly restricted, edges may be shifted from one node to another and there is a need for negative patterns. Based on this analysis, we define here a suitable framework for NLP (see Section 3).

Compared to term rewriting, the semantics of graph rewriting is problematic: different choices can be made in the way the context is glued to the rule application [15]. As far as we see, our notion does not fit properly the DPO approach due to unguarded node deletion nor the SPO approach due to the shift command, as we shall see. Thus we will provide a complete description of our notion. We have chosen to present it in an operational way and we leave for future work a categorial semantics.

In our application, we use several hundreds of rules. To manage such a system, we use a notion of modular graph rewriting system: the full set of rules is divided in smaller subsets (called modules) that are used in turn.

In practice, we need some tools to verify termination and confluence properties of modules. In Section 4, we provide two termination methods based on a weight analysis. First, there is a direct motivation: in our NLP application, any computations should terminate. If it is not the case, it means that the rules where not correctly drawn. Then, termination ensures partly the correctness of the transformation. There is also an indirect reason to consider termination: one way of establishing confluence is through Newman's Lemma [11] which requires termination.

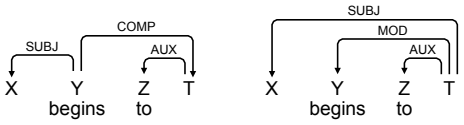
We consider two properties of the above mentioned termination methods. First, we show that they are decidable, that is the existence of weights can be computed statically from the rules, and thus we have a fully automatic tool to verify termination. Obviously, it is not complete. In a second step, we evaluate the strength of the two methods. To do that, we consider what restrictions they impose on the length of computations. We get quadratic time for the first method, polynomial time for the second. This article is an extended abstract of [2].

2 Linguistic motivations

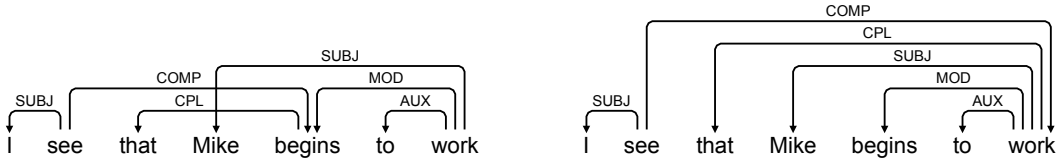
Without any linguistic exhaustivity, we highlight in this section some crucial points of the kind of linguistic transformation we are interested in and hence the relative features of rewriting we have to consider.

Node preservation property. As linguistic examples above suggest, the goal of linguistic analysis is mainly to describe different kinds of relations between elements that are present in the input structure. As a consequence, the set of nodes in the output structure is directly predictable from the input and only a very restrictive notion on node creation is needed. In practice, these node creations can be anticipated in some enriched input structure on which the whole transformation can be described as a non-size increasing process.

Edge shifting. In the first example of the introduction (for the sentence *"I see that Mike begins to work"*), the verb *"begins"* is called a raising verb and we know that *"Mike"* is the *deep subject* of the verb *"work"*; *"begins"* being considered as a modifier of the verb. To recover this deep subject, one may imagine a local transformation of the graph which turns the first graph on the right into the second one.

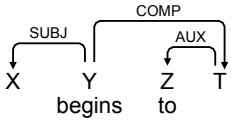


However, in our example above, a direct application of such a transformation leads to the structure below on the left which is not the right structure. Indeed, the transformation should shift what the linguists call the head of the phrase *"Mike begins to work"* from the word *"begins"* to the word *"work"* with all relative edges. In that case, the transformation should produce the structure below on the right:



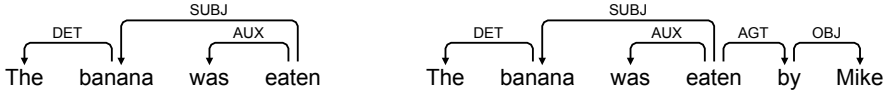
In a more general setting, our transformations may have to specify the fact that all incident edges of some node X must be transported to some other node Y. We call this operation `shift`.

To describe our graph rewriting rules, we introduce a system of commands (like in [6]) which expresses step by step the modifications applied on the input graph. The transformation described above is performed in our setting as follows:



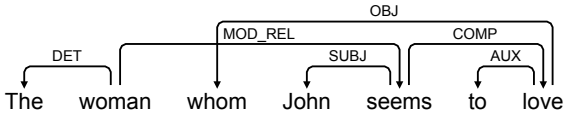
`del_edge` (Y,SUBJ,X); `del_edge` (Y,COMP,T);
`add_edge` (T,SUBJ,X); `add_edge` (T,MOD,Y);
`shift` (Y,T)

Negative conditions. In some situation, rules must be aware of the context of the pattern to avoid unwanted ambiguities. When computing semantics out of syntax, one has to deal with passive sentence; the two sentences below show that the agent is optional.

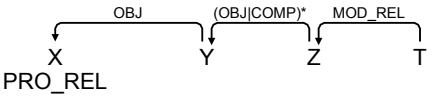


In order to switch to the corresponding active form, two different linguistic transformations have to be defined for these two sentence; but, clearly, the first graph is a subgraph of the second one. We don't want the transformation for the short passive on the left to apply on the long passive on the right. we need to express a negative condition like "there is no out edge labeled by AGT out of the main verb" to prevent the unwanted transformation to occur.

Long distance dependencies. Most of the linguistic transformation can be expressed with successive local transformation like the one above. Nevertheless, there are some cases where more global rewriting is required; consider the sentence *"The women whom John seems to love"*, for which we consider the syntactic structure on the right. One of the steps in the semantic construction of this sentence requires to compute the antecedent of the relative pronoun *"whom"* (the noun *"woman"* in our example).

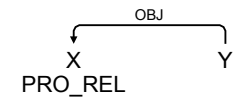


The subgraph we have to search in our graph (which is depicted as a non-local pattern) and the graph modification to perform are given on the right. The number of OBJ or COMP relations to consider (in the relation depicted as (OBJ|COMP)* in the figure) is unbounded (in linguistics, this phenomenon is called long distance dependencies); it is possible to construct grammatical sentences with an arbitrary large number of relations.

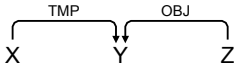


`add_edge` (X,ANT,T)

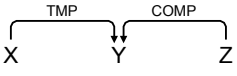
As we want to stay in the well-known framework of local rewriting, we will use several local transformations to implement such a non-local rule.



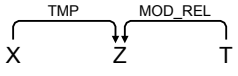
`add_edge` (X,TMP,Y)



`del_edge` (X,TMP,Y)
`add_edge` (X,TMP,Z)



`del_edge` (X,TMP,Y)
`add_edge` (X,TMP,Z)



`del_edge` (X,TMP,Z)
`add_edge` (X,ANT,T)

The second and the third rules above preserve the set of nodes and the number of edges of each kind. Hence, this kind of rule will require special treatment with respect to termination issues.

3 Graph Rewriting for NLP

Before we enter into the technical sections, let us define some useful notations. First, we use the notation $\vec{c}$ to denote sequences. The empty sequence is written $\emptyset$. The length of a sequence is denoted by $|\vec{c}|$. We use the same notation for sets: the empty set is denoted $\emptyset$ and the cardinality of a set S is written $|S|$. The context will make clear whether we are talking about sequences or sets.

Given a function $f : X \to Y$ and some sets $X' \subseteq X$ and $Y' \subseteq Y$, we define $f(X') \triangleq \{f(x) \mid x \in X'\}$ and $f^{-1}(Y') \triangleq \{x \in X \mid f(x) \in Y'\}$; the restriction of the function f to the domain X' is $f|_{X'} : x' \in X' \mapsto f(x')$. The function $\mathbf{c}_X : x \in X \mapsto c \in Y$ is the constant function on X. The identity function is written $\mathbb{1}$. Finally, given a function $f : X \to Y$ and $(x,y) \in X \times Y$, the function $f[x \mapsto y]$ maps $t \neq x$ to $f(t)$ and x to y.

The set of natural numbers is $\mathbb{N}$, integers are denoted by $\mathbb{Z}$. Given two integers a,b, we define $[a,b] = \{x \in \mathbb{Z} \mid a \leq x \leq b\}$.

3.1 Graphs

The graphs we consider are directed graphs with both labels on nodes and labels on edges. We restrict the edge set: given some edge label e, there is at most one edge labeled e between two given nodes α and β. This restriction reflects the fact that, in NLP application, our edges are used to encode linguistic information which are relations. We make no other explicit hypothesis on graphs: in particular, graphs may be disconnected, or have loops.

In this paper, we suppose given a finite set Σ_E of edge labels and another finite set Σ_N of node labels.

Definition 3.1 (Graph)**.** *A* graph *G is defined as a triple $(\mathcal{N},\ell,\mathcal{E})$ where*

- *$\mathcal{N}$ is a finite set of nodes;*
- *ℓ is a labeling function: $\ell : \mathcal{N} \mapsto \Sigma_N$;*
- *$\mathcal{E}$ is a set of edges: $\mathcal{E} \subseteq \mathcal{N} \times \Sigma_E \times \mathcal{N}$.*

Let $n,m \in \mathcal{N}$ and $e \in \Sigma_E$. When there is an edge from n to m labelled e (*i.e.* $(n,e,m) \in \mathcal{E}$), we write $n \xrightarrow{e} m$ or $n \longrightarrow m$ if the edge label is not relevant. If G denotes some graph $(\mathcal{N},\ell,\mathcal{E})$, then $\mathcal{N}_G$, ℓ_G, $\mathcal{E}_G$ denote respectively $\mathcal{N},\ell$ and $\mathcal{E}$.

Definition 3.2 (Graph morphism)**.** *A* graph morphism *μ from the graph $G = (\mathcal{N},\ell,\mathcal{E})$ to the graph $G' = (\mathcal{N}',\ell',\mathcal{E}')$ is a function from $\mathcal{N}$ to $\mathcal{N}'$ such that:*

- *for all $n \in \mathcal{N}$, $\ell'(\mu(n)) = \ell(n)$;*
- *for all $n,m \in \mathcal{N}$ and $e \in \Sigma_E$, if $n \xrightarrow{e} m \in \mathcal{E}$ then $\mu(n) \xrightarrow{e} \mu(m) \in \mathcal{E}'$.*

A graph morphism μ is said to be injective *if $\mu(n) = \mu(m)$ implies $n = m$. We make the following abuse of notation: given some graph morphism $\mu : G \to G'$, and a set $E \subseteq \mathcal{E}_G$, we let $\mu(E) = \{\mu(n) \xrightarrow{e} \mu(m) \mid n \xrightarrow{e} m \in E\}$.*

Definition 3.3 (Basic pattern and basic matching)**.** *A* basic pattern *B is a graph. A* basic matching *μ of the basic pattern B in the graph G is an* injective *graph morphism μ (written $\mu : B \hookrightarrow G$).*

As shown in Section 2, negative conditions on patterns naturally arise in NLP. We classify negative conditions in two categories: the local ones, that is negative conditions on edges within the basic pattern and non-local ones, that is negative conditions concerning edges between a node of the basic pattern and a node of the context (either in-edges or out-edges).

Definition 3.4 (Pattern)**.** *A* pattern *is a quadruple $P = (B,\bar{\mathcal{E}},\bar{\mathcal{I}},\bar{\mathcal{O}})$ of:*

- *a basic pattern $B = (\mathcal{N}_P,\ell_P,\mathcal{E}_P)$;*
- *a set of forbidden edges $\bar{\mathcal{E}} \subset \mathcal{N}_P \times \Sigma_E \times \mathcal{N}_P$ such that $\bar{\mathcal{E}} \cap \mathcal{E}_P = \emptyset$;*
- *a set of forbidden in-edges $\bar{\mathcal{I}} \subset \mathcal{N}_P \times \Sigma_E$*
- *a set of forbidden out-edges $\bar{\mathcal{O}} \subset \mathcal{N}_P \times \Sigma_E$*

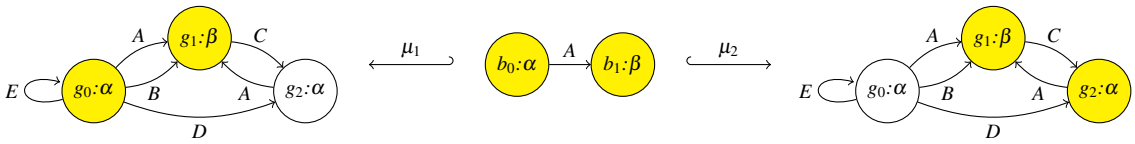
Given a basic pattern B, we shorten $(B,\emptyset,\emptyset,\emptyset)$ to $(B,\vec{\emptyset})$. In the following, given a pattern P, $\mathcal{N}_P$ and $\mathcal{E}_P$ denote respectively the set of nodes of its basic pattern and the set of edges of its basic pattern.

Definition 3.5 (Matching)**.** *Let $P = (B,\bar{\mathcal{E}},\bar{\mathcal{I}},\bar{\mathcal{O}})$ be a pattern, $G = (\mathcal{N},\ell,\mathcal{E})$ be a graph, and $\mu : B \hookrightarrow G$ be a basic matching. We say that μ is a* matching *from P into G (also written $\mu : P \hookrightarrow G$) whenever it satisfies the additional three conditions:*

- *$\mu(\bar{\mathcal{E}}) \cap \mathcal{E} = \emptyset$*

- *for each* $(n,e) \in \bar{\mathscr{I}}$, $\{p \in \mathscr{N} \setminus \mu(\mathscr{N}_P) \mid p \xrightarrow{e} \mu(n)\} = \emptyset$
- *for each* $(n,e) \in \bar{\mathscr{O}}$, $\{p \in \mathscr{N} \setminus \mu(\mathscr{N}_P) \mid \mu(n) \xrightarrow{e} p\} = \emptyset$

Example 3.1. *Negative conditions are used to remove 'unwanted' matching. To see their effect, consider for instance the basic pattern* B_0 *and its two basic matchings* μ_1 *and* μ_2 *in* G_0*:*



- *First, let* $P_0 = (B_0, \vec{\emptyset})$*. Then,* μ_1 *and* μ_2 *are (the) two matchings* $P_0 \hookrightarrow G_0$*.*
- *Second, let the pattern* $P_1 = (B_0, \{(b_1, C, b_0)\}, \emptyset, \emptyset)$*; then,* μ_1 *is the only matching* $P_1 \hookrightarrow G_0$*.*
- *Third, let the pattern* $P_2 = (B_0, \emptyset, \{(b_0, D)\}, \{(b_0, D)\})$*. Then, there is no matching of* P_2 *into* G_0*.*

In the following, patterns P_1 *and* P_2 *are depicted as:*

3.2 Graph decomposition

The proper description of actions of a rule on some graph G requires first the definition of two partitions: one on nodes and the other on edges. They are both induced by the matching of some pattern P into G.

Definition 3.6 (Nodes decomposition: pattern image, crown and context)**.** *Let* $\mu : P \hookrightarrow G$ *a matching from the pattern* P *into the graph* $G = (\mathscr{N}, \ell, \mathscr{E})$*. Nodes of* G *can be split in a partition of three sets* $\mathscr{N} = \mathscr{P}_\mu \oplus \mathscr{K}_\mu \oplus \mathscr{C}_\mu$*:*

- *the* pattern image *is* $\mathscr{P}_\mu = \mu(\mathscr{N}_P)$*;*
- *the* crown *contains nodes outside the pattern image which are directly connected to the pattern image:* $\mathscr{K}_\mu = \{n \in \mathscr{N} \setminus \mathscr{P}_\mu \mid \exists p \in \mathscr{P}_\mu$ such that $n \longrightarrow p$ or $p \longrightarrow n\}$*;*
- *the* context *contains nodes not linked to the pattern image:* $\mathscr{C}_\mu = \mathscr{N} \setminus (\mathscr{P}_\mu \cup \mathscr{K}_\mu)$*.*

Definition 3.7 (Edges decomposition: pattern edges, crown edges, context edges and pattern-glued edges)**.** *Let* $\mu : P \hookrightarrow G$ *a matching from the pattern* P *into the graph* $G = (\mathscr{N}, \ell, \mathscr{E})$*. Edges of* G *can be split in a partition of four sets* $\mathscr{E} = \mu(\mathscr{E}_P) \oplus \mathscr{K}^\mu \oplus \mathscr{C}^\mu \oplus \mathscr{H}^\mu$*:*

- *the* pattern edges *is* $\mu(\mathscr{E}_P)$*;*
- *the* crown edges *set contains edges which links a pattern image node to a crown node:* $\mathscr{K}^\mu = \{n \longrightarrow m \in \mathscr{E} \mid n \in \mathscr{P}_\mu \wedge m \in \mathscr{K}_\mu\} \cup \{n \longrightarrow m \in \mathscr{E} \mid n \in \mathscr{K}_\mu \wedge m \in \mathscr{P}_\mu\}$*;*
- *the* context edges *set contains edges which connect two nodes that are not in the pattern image:* $\mathscr{C}^\mu = \{n \longrightarrow m \in \mathscr{E} \mid n \notin \mathscr{P}_\mu \wedge m \notin \mathscr{P}_\mu\}$*.*
- *the* pattern-glued edges *set contains edges which are not pattern edges but which connect two nodes that are in the pattern image:* $\mathscr{H}^\mu = \{n \longrightarrow m \in \mathscr{E} \mid n \in \mathscr{P}_\mu \wedge m \in \mathscr{P}_\mu\} \setminus \mu(\mathscr{E}_P)$*.*

3.3 Rules

In our graph rewriting framework, the transformation of the graph is described through some atomic commands (like in [6]). Commands definition refer to some pattern P and pattern nodes $\mathcal{N}_P$ are used as identifiers. Let $a,b \in \mathcal{N}_P$, $\alpha \in \Sigma_N$ and $e \in \Sigma_E$, the five kinds of commands are $\texttt{label}(a,\alpha)$, $\texttt{del\_edge}(a,e,b)$, $\texttt{add\_edge}(a,e,b)$, $\texttt{del\_node}(a)$ and $\texttt{shift}(a,b)$.

Their names speak for themselves, however, we will come back to their precise meaning in the subsection below. Before this, to ensure that commands always refer to valid node identifiers, we restrict command sequences to *consistent* sequences, that is sequences $c_1,\ldots,c_k$ such that for each command c_i, $1 \le i \le k$, which is a node deletion command $\texttt{del\_node}(a)$ for some $a \in \mathcal{N}_P$, then the node name a does not occur in any command c_j with $i < j \le k$.

Definition 3.8 (Rule)**.** *A* rule *R is a pair $R = \langle P, \vec{c} \rangle$ of a pattern P and a sequence of commands $\vec{c}$ consistent with respect to P. A rule $R = \langle P, \vec{c} \rangle$ is said to be* node-preserving *if $\vec{c}$ does not contain any* `del_node` *command.*

3.4 Graph Rewrite System

Let $G = (\mathcal{N}, \ell, \mathcal{E})$ a graph, $R = \langle P, \vec{c} \rangle$ a rule and $\mu : P \hookrightarrow G$ a matching. The application of the sequence $\vec{c}$ on G is a new graph which is written $G \cdot_\mu \vec{c}$ (shortened $G \cdot \vec{c}$ when μ is clear from the context) and is defined by induction on the length k of $\vec{c}$. If $k = 0$, $G \cdot \emptyset = G$. If $k > 0$, let $G' = (\mathcal{N}', \ell', \mathcal{E}')$ be the graph obtained by application of the sequence $c_1,\ldots,c_{k-1}$; then we consider each command in turn:

Label: The command $c_k = \texttt{label}(a,\alpha)$ changes the label of the node $\mu(a)$: $G \cdot \vec{c} = (\mathcal{N}', \ell'', \mathcal{E}')$ with $\ell'' = \ell'[\mu(a) \mapsto \alpha]$.

Delete: The command $c_k = \texttt{del\_edge}(a,e,b)$ deletes the edge from $\mu(a)$ to $\mu(b)$ labelled with $e \in \Sigma_E$; $G \cdot \vec{c} = (\mathcal{N}', \ell', \mathcal{E}'')$ with $\mathcal{E}'' = \mathcal{E}' \setminus \{\mu(a) \xrightarrow{e} \mu(b)\}$.

Add: The command $c_k = \texttt{add\_edge}(a,e,b)$ adds an edge from $\mu(a)$ to $\mu(b)$ labelled with $e \in \Sigma_E$; $G \cdot \vec{c} = (\mathcal{N}', \ell', \mathcal{E}'')$ with $\mathcal{E}'' = \mathcal{E}' \cup \{\mu(a) \xrightarrow{e} \mu(b)\}$.

Delete node: The command $c_k = \texttt{del\_node}(a)$ removes the node $\mu(a)$ of G'; $G \cdot \vec{c} = (\mathcal{N}'', \ell'', \mathcal{E}'')$ with $\mathcal{N}'' = \mathcal{N}' \setminus \{\mu(a)\}$, $\ell'' = \ell'|_{\mathcal{N}''}$ and $\mathcal{E}'' = \mathcal{E}' \cap \{\mathcal{N}'' \times \Sigma_E \times \mathcal{N}''\}$.

Shift edges: The command $c_k = \texttt{shift}(a,b)$ changes in-edges of $\mu(a)$ starting from the crown to in-edges of $\mu(b)$ and all out-edges of $\mu(a)$ going to the crown to out-edges of $\mu(b)$. Formally, $G \cdot \vec{c} = (\mathcal{N}', \ell', \mathcal{E}'')$ with the set $\mathcal{E}''$ defined by, for all $e \in \Sigma_E$:

- for all $p \in \mathcal{K}_\mu$, $\mu(b) \xrightarrow{e} p \in \mathcal{E}''$ iff $\mu(b) \xrightarrow{e} p \in \mathcal{E}'$ or $\mu(a) \xrightarrow{e} p \in \mathcal{E}'$;
- for all $p \in \mathcal{K}_\mu$, $p \xrightarrow{e} \mu(b) \in \mathcal{E}''$ iff $p \xrightarrow{e} \mu(b) \in \mathcal{E}'$ or $p \xrightarrow{e} \mu(a) \in \mathcal{E}'$;
- for all $p,q \in \mathcal{P}_\mu$, $p \xrightarrow{e} q \in \mathcal{E}''$ iff $p \xrightarrow{e} q \in \mathcal{E}'$;
- for all $p,q \in \mathcal{K}_\mu \cup \mathcal{C}_\mu$, $p \xrightarrow{e} q \in \mathcal{E}''$ iff $p \xrightarrow{e} q \in \mathcal{E}'$.

The commands `label`, `del_edge` and `add_edge` are called local commands: they modify only the edges and the nodes described in the pattern. The commands `del_node` and `shift` are non-local: they can modify edges outside the pattern. Note that a rule `add_edge` (resp. `del_edge`) may have no effect if the edge already exists (resp. does not exist). Note also that we can suppose that for a given sequence $\vec{c}$ and a given triple (a,e,b), there is at most one rule $\texttt{del\_edge}(a,e,b)$ or $\texttt{add\_edge}(a,e,b)$ in $\vec{c}$ (if not, only the last one is effective). Hence, we can define uniform rules:

Definition 3.9 (Uniform rule). *For $\vec{c} = c_1, \ldots, c_k$ without node deletion, the rule $\langle P, \vec{c} \rangle$ is* uniform *iff for all $1 \leq i \leq k$, if $c_i = \texttt{add\_edge}(n,e,m)$ then $(n,e,m) \in \bar{\mathscr{E}}_P$ and if $c_i = \texttt{del\_edge}(n,e,m)$ then $(n,e,m) \in \mathscr{E}_P$.*

Definition 3.10 (Rewrite step). *Let $G = (\mathscr{N}, \ell, \mathscr{E})$ a graph, $R = \langle P, \vec{c} \rangle$ a rule and $\mu : P \hookrightarrow G$ a matching. Let $G' = G \cdot \vec{c}$, then we say that G rewrites to G' with respect to the rule R and the matching μ. We write it $G \to_{R,\mu} G'$ or $G \to_R G'$ or even simply $G \to G'$.*

Definition 3.11 (Graph Rewrite System). *A* Graph Rewrite System *$\mathscr{G}$ is a finite set of rules.*

In our application, the translation of the syntax to semantics is split into several independent levels of transformation driven by linguistic consideration (such as translation of passive forms to active ones, computation of the deep subject of infinites). Rules are then grouped in subsets called *modules* and modules apply sequentially; each module being used as a graph rewrite system on the outputs of the previous module.

Lemma 3.1 (Linear modification). *Given a GRS $\mathscr{G}$, there is a constant $C > 0$ such that, for any rewriting step $G \to_{R,\mu} G'$ the two canonical corresponding edge decompositions $\mathscr{E}_G = \mathscr{C}^\mu \oplus \mathscr{Q}^\mu$ and $\mathscr{E}_{G'} = \mathscr{C}^\mu \oplus \mathscr{Q}'^\mu$ satisfy:*

$$|\mathscr{Q}^\mu| \leq C \times (|G|+1) \text{ and } |\mathscr{Q}'^\mu| \leq C \times (|G|+1)$$

Proof. Let $C = \max\{2 \times |P|^2 \times |\Sigma_E| \} \mid \langle P, \vec{c} \rangle \in \mathscr{G}\}$. Both in G and G', edges that are not in the context are either between two pattern nodes or between a pattern node and a crown node. The total number of edges of the first kind (either pattern edges or glued-pattern edges) is bounded by $|P|^2 \times |\Sigma_E|$. For each pattern node, the number of edges which connect this node to some non-pattern node is bounded by $2 \times |G| \times |\Sigma_E|$ and so the total number of edges which link some pattern node to some non-pattern node is bounded by $2 \times |G| \times |\Sigma_E| \times |P|$. Putting everything together, $|\mathscr{Q}^\mu| \leq C \times (|G|+1)$ and $|\mathscr{Q}'^\mu| \leq C \times (|G|+1)$. □

4 Weighted GRS

We recall that a GRS is said to be (strongly) terminating whenever there is no infinite sequence $G_1 \to G_2 \to \cdots$. Given a terminating GRS $\mathscr{G}$ and a graph G, we define the derivation height of G, next denoted $h_\mathscr{G}(G)$, to be the length of the longest derivation starting from G if such a derivation exists. If $h_\mathscr{G}(G)$ is defined for all G such that $|G| \leq n$, then we define the derivation height of $\mathscr{G}$ by: $h_\mathscr{G}(n) = \max\{h_\mathscr{G}(G) \mid |G| \leq n\}$.

Actually, for non-size increasing GRS as presented above, we have immediately the decidability of non-uniform termination. That is, given some GRS $\mathscr{G}$ and some graph G, one may decide whether there is an infinite sequence $G_1 \to G_2 \to G_3 \to \cdots$. Indeed, one may observe that for such sequence, for all $i \in \mathbb{N}$, $|G_i| \leq |G|$. Thus, the G_i's range in the finite set $\mathfrak{G}_{\leq |G|}$ of graphs of size less or equal to $|G|$. Consequently, either the system terminates or there is some $j \leq |\mathfrak{G}_{\leq |G|}|$ and some $k \leq j$ such that $G_j = G_k$. To conclude, to decide non-uniform termination, it is sufficient to compute all the (finitely many) possibilities of rewriting G in less than $|\mathfrak{G}_{\leq |G|}|$ steps and to verify the existence of such a j and k above. Finally, since $|\mathfrak{G}_{\leq |G|}| \leq 2^{O(|G|^2)}$, the procedure as described above takes exponential time.

However, uniform termination— given a GRS, is it terminating?— of non-size increasing GRS remains an open problem. Uniform termination was proved undecidable when we drop the property of non-size increasingness (cf. Plump [14]). As a consequence, there is a need to define some termination method pertaining to non-size increasing GRS. Compared to standard work in termination [13, 7], there are two difficulties: first, our graphs may be cyclic, thus forbidding methods developed for DAGs such as term-graphs. Second, using term rewriting terminology, our method should operate for some

non-simplifying GRS, that is GRS for which the output may be "bigger" than the input. Indeed, the NLP programmer sometimes wants to compute some *new* relations, so that the input graph is a strict sub-graph of the resulting graph.

4.1 Termination by weight analysis

In the context of term-rewriting systems, the use of weights is very common to prove termination. There are many examples of such orderings, Knuth-Bendix Ordering [10] to cite one of them. We recall that all graphs we consider are defined relatively to two signatures Σ_E of edge labels and Σ_N of node labels.

Definition 4.1 (Edge weight, node weight)**.** *An* edge weight *is a mapping $w : \Sigma_E \to \mathbb{Z}$. Given some subset E of edges of G, the weight of E is $w(E) = \sum_{n \xrightarrow{e} m \in E} w(e)$. The edge weight of a graph G is $w(G) = w(\mathscr{E}_G)$. A* node weight *is a mapping $\eta : \Sigma_N \to \mathbb{Z}$. For a graph $G = (\mathscr{N}_G, \ell_G, \mathscr{E}_G)$, we define $\eta(G) = \sum_{n \in \mathscr{N}_G} \eta(\ell_G(n))$.*

Let us make some observations. Let $|G|_e$ denote the number of edges in G which have the label e, then $w(G) = \sum_{e \in \Sigma_E} w(e) \times |G|_e$. Second, for a pattern matching $\mu : P \hookrightarrow G$, $w(\mu(P)) = w(P)$.

The weight of a graph may be negative. This is not standard, but it is useful here to cope with non-simplifying rules, that is rules which add new edges. Since a graph G has at most $|\Sigma_E| \times |G|^2$ edges, the following lemma is immediate.

Lemma 4.1. *Given an edge weight w and a node weight η, let $K_w = \max_{e \in \Sigma_E}(|w(e)|)$, $K_E = |\Sigma_E| \times K_w$, $K_\eta = \max_{\alpha \in \Sigma_N}(|\eta(\alpha)|)$, then*

(a) for each subset of edges $E \subset \mathscr{E}_G$ of some graph G, we have $w(E) \leq K_w \times |E|$.

(b) for each graph G, we have $-K_E \times |G|^2 \leq w(G) \leq K_E \times |G|^2$;

(c) for each graph G, we have $|\eta(G)| \leq K_\eta \times |G|$.

Definition 4.2. *Let $R = \langle P, \vec{c} \rangle$ a rule, we define inductively $\Phi_{\vec{c}} : \mathscr{N}_P \to \mathscr{N}_P$ which describes the global effect of the* `shift` *commands in a rule: $\Phi_\emptyset = \mathbb{1}$; $\Phi_{\vec{c},\texttt{shift}(m,n)} = \mathbb{1}[m \mapsto n] \circ \Phi_{\vec{c}}$ and $\Phi_{\vec{c},c} = \Phi_{\vec{c}}$ if c is not a* `shift` *command.*

Definition 4.3 (Compatible weight)**.** *Given a rule $R = \langle (P, \bar{\mathscr{E}}, \bar{\mathscr{I}}, \bar{\mathscr{O}}), \vec{c} \rangle$, an edge weight w is said to be* compatible *with R if:*

1. *either $\vec{c}$ contains a* `del_node` *command*
2. *or R is a node-preserving rule and satisfy the three properties:*
   (a) *R is uniform,*
   (b) *$w(P \cdot_{\mathbb{1}} \vec{c}) < w(P)$,*
   (c) *for all $e \in \Sigma_E$ such that $w(e) < 0$, for all $n \in \Phi(\mathscr{N}_P)$, let $M_n \subset \mathscr{E}_P$ be the set $\Phi_{\vec{c}}^{-1}(n)$; then M_n contains at most one element m such that $(m, e) \notin \bar{\mathscr{I}}$ and M_n contains at most one element $m' \in M_n$ such that $(m', e) \notin \bar{\mathscr{O}}$.*

An edge weight is said to be compatible with a GRS $\mathscr{G}$ if it is compatible with all its rules. A weighted GRS is a pair $(\mathscr{G}, w)$ of a GRS and a compatible weight.

Hypothesis (2.b) will serve to manage edges in the pattern images while Hypothesis (2.c) will serve for the crown edges. One may note that when there is no `shift` commands in the rule, the Hypothesis (2.c) holds whatever w is. Indeed, in that case, Φ is the identity function and all the sets M_n are singletons.

Lemma 4.2. *Let $(\mathscr{G},w)$ a weighted GRS, let $G \to G'$ be a rewrite step of $\mathscr{G}$. Either $|G| > |G'|$ or $|G| = |G'|$ and $w(G) > w(G')$.*

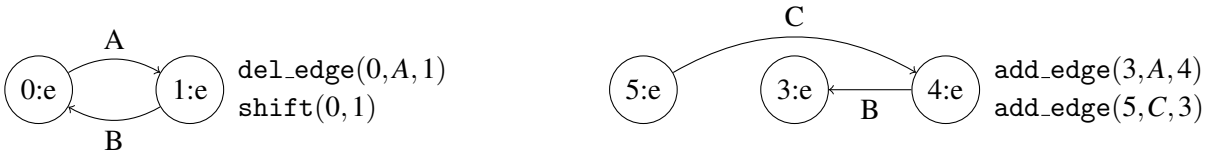
The problem of the synthesis is the following. Given a GRS $\mathscr{G}$, is there a weight w compatible with $\mathscr{G}$? Since the existence of weights can be described in Presburger's arithmetic, we have a positive answer:

Theorem 4.1. *Given a GRS $\mathscr{G}$, one may decide whether or not it has a compatible weight.*

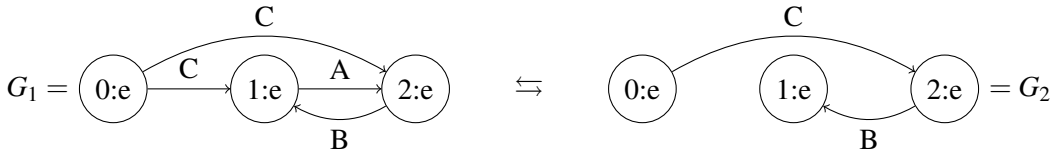
Second point, the existence of weights induce termination:

Theorem 4.2. *Any weighted GRS $(\mathscr{G},w)$ is strongly terminating in quadratic time. Moreover, this quadratic bound is a lower bound: there is a GRS $\mathscr{G}$ with a compatible weight such that $h_{\mathscr{G}}(n) \geq O(n^2)$.*

Condition (2.c) of Definition 4.3 is necessary. Here is a counter-example of a non-terminating system with a compatible weight up to this condition. Consider the two rules $\langle Q_1, \vec{c_1} \rangle$ and $\langle Q_2, \vec{c_2} \rangle$:



Set $w(A) = w(B) = 1$ and $w(C) = -2$. Observe that $w(Q_1 \cdot \vec{c_1}) = 1 < 2 = w(Q_1)$ and $w(Q_2 \cdot \vec{c_2}) = -2 < -1 = w(Q_2)$. However, there is an infinite sequence $G_1 \to_{R_1} G_2 \to_{R_2} G_1 \to_{R_1} \cdots$ with G_1 and G_2 being:



Proof sketch of Theorem 4.2. We begin to show the lower bound. Let $\Sigma_E = \{E\}$, $\Sigma_N = \{e\}$. Consider the two rules GRS $\mathscr{G}$ defined by the two basic patterns:

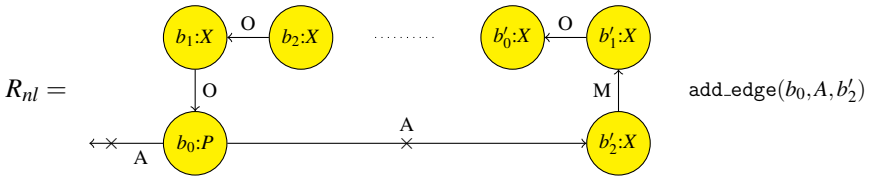
Set $w(E) = 1$. The rules are compatible with w. Each rule deleting exactly one edge, since the clique C_n of size n has n^2 edges, the derivation height $h_{\mathscr{G}}(C_n) = n^2$. The lower bound follows.

For the upper bound, let C be the constant as defined by Lemma 3.1, let $K = \max(1, K_w)$ (we recall that $K_w = \max_{e \in \Sigma_E}(|w(e)|)$). Finally, let $H = \max\{n \mid (P, c_1, \ldots, c_n) \in \mathscr{G}\}$. Let $A = 2 \times K \times C \times (H+1) + 1$. Let Ω be the 'energy function' defined on graphs $\Omega(G) = w(G) + A \times |G|^2$. For each rule application $G \to G'$, one may verify that $\Omega(G) > \Omega(G')$. The last inequality together with Lemma 4.1 leads to the conclusion. □

Full proofs of Theorem 4.1 and of Theorem 4.2 are given in [2].

4.2 Termination by lexicographic weight

In our experiments, in most cases, the weight analysis of the preceding section was sufficient. The main counter-example is however systems composed of rules as given in Section 2. The GRS is strongly terminating but there is no compatible weight. This section provides a conciliable extension of this termination proof method. With a little abstraction, the linguistic example of Section 2 about long distance dependencies is computed by some 'non-local rule' R_{nl}:



Such non-local rules can be implemented by rules:

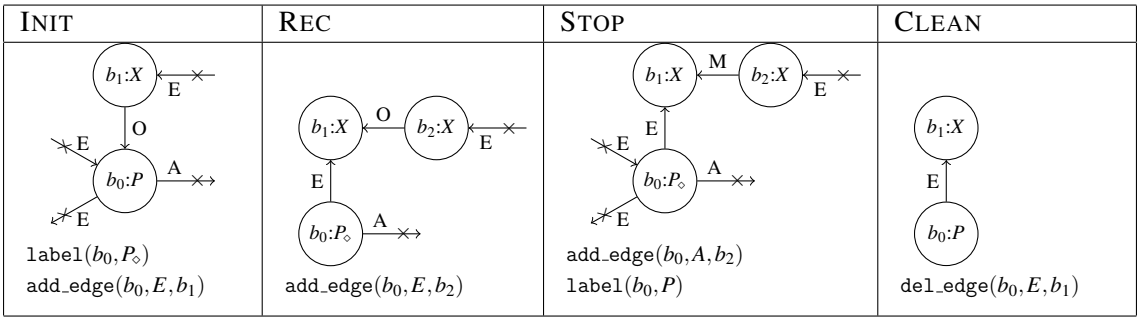


Figure 1: Local implementation of the non-local rule

However, these rules are not compatible with any weight. Actually, as justified in [2], there is no implementation of such a rule by some weighted rules.

Given an order $\prec$ on some set U, its lexicographic extension to sequences in U is defined by $(u_1,\dots,u_k) \prec_{\text{lex}} (v_1,\dots,v_m)$ iff $\exists j \leq \min(m,k) : u_j \prec v_j \wedge \forall i < j : u_i = v_i$. The order $\prec_{\text{lex}}$ is not well-founded in general, but its restriction to sequences of equal length is such as soon as $\prec$ is well-founded.

Definition 4.4 (Contextual weight)**.** *An* edge contextual weight *is a (finite) map* $\omega : \Sigma_N \times \Sigma_E \times \Sigma_N \to \mathbb{Z}$. *As for weights, it extends to any set* $E \subseteq \mathscr{E}_G$ *of some graph* G *by:* $\omega(E) = \sum_{n \xrightarrow{e} m \in E} \omega(\ell(n), e, \ell(m))$. *And the weight of a graph is* $\omega(G) = \omega(\mathscr{E}_G)$.

A contextual weight *is a 4-tuple* $\pi = (a, \omega, b, \eta)$ *with* $a, b \in \mathbb{N}$, ω *an edge contextual weight and* η *a node weight. We define* $\pi(G) = a \times \omega(G) + b \times \eta(G)$.

Let $e \in \Sigma_E$, *if* $a \neq 0$ *and there are* $\alpha, \beta, \alpha', \beta' \in \Sigma_N$ *such that* $\omega(\alpha, e, \beta) \neq \omega(\alpha', e, \beta')$, *then we say that* π *is* e-fragile.

Definition 4.5. *Given an edge weight* $w_0 : \Sigma_E \to \mathbb{Z}$, *given* k *contextual weights* $\pi_1, \dots, \pi_k$ *and a rule* $R = \langle P, \vec{c} \rangle$, *we write* $P' = P \cdot_{\mathbb{1}} \vec{c}$. *We say that* R *is* compatible *with* $(w_0, \pi_1, \dots, \pi_k)$ *iff:*

1. *either* $\vec{c}$ *contains a* `del_node` *command,*
2. *or* R *is an uniform and node-preserving rule such that:*
   - *(a) either the two properties below hold*
     - *(i)* $w_0(P') < w_0(P)$;
     - *(ii) and for all* $e \in \Sigma_E$ *such that* $w(e) < 0$, *for all* $n \in \Phi(\mathscr{N}_P)$, *let* M_n *be the set* $\Phi^{-1}(n)$; *then* M_n *contains at most one element* m *such that* $(m, e) \notin \bar{\mathscr{F}}$ *and* M_n *contains at most one element* $m' \in M_n$ *such that* $(q, m') \notin \bar{\mathscr{O}}$.
   - *(b) or the four properties below hold*
     - *(i)* $w_0(P') = w_0(P)$;
     - *(ii)* $(\pi_1(P'), \dots, \pi_k(P')) <_{lex} (\pi_1(P), \dots, \pi_k(P))$;

(iii) if $\vec{c}$ contains a command `label`(n,α) *and if some π_i is e-fragile, then $(n,e) \in \bar{\mathscr{I}} \cup \bar{\mathscr{O}}$;*
(iv) $\vec{c}$ does not contain any `shift` *commands.*

When a weight w_0 and k contextual weights are compatible with all the rules of some GRS $\mathscr{G}$, we say that $\mathscr{G}$ is lexicographically weighted by $(w_0, \pi_1, \ldots, \pi_k)$.

Example 4.1. *We define $w_0 = \mathbf{0}_{\Sigma_E}[A \mapsto -1]$, and $\omega = \mathbf{0}_{\Sigma_N \times \Sigma_E \times \Sigma_N}[(P,E,X) \mapsto 1, (P_\diamond, E, X) \mapsto -1]$. Consider the lexicographic weight $\pi = (1, \omega, 0, \mathbf{0}_{\Sigma_N})$. For rules in Figure 1, we have: rule* STOP *decreases by (2.a); rules* INIT *and* REC *decrease by (2.b): there is one more edge labeled E starting from $P_\diamond$ and rule* CLEAN *decreases by (2.b): one edge labeled E starting from P disappears.*

Theorem 4.3. *Whenever a program $\mathscr{G}$ is compatible with the lexicographic weight $(w_0, \pi_1, \ldots, \pi_k)$, it is strongly terminating in polynomial time. The bound is tight, that is for all $k > 0$, there is a GRS whose derivation height is $O(n^k)$.*

Proof. Examples for the lower bound are proposed in [2]. For the upper bound, let

$$K_\omega = \max\{|\omega(n,e,m)| \mid (n,e,m) \in \Sigma_N \times \Sigma_E \times \Sigma_N\} \quad \text{and} \quad K_\pi = a \times |\Sigma_E| \times K_\omega + b \times K_\eta$$

Then, adapting Lemma 4.1(b) to the present context, we can state that $|\omega(G)| \leq K_\omega \times |\Sigma_E| \times |G|^2$. With Lemma 4.1(c), we have $|\eta(G)| \leq K_\eta \times |G|$ and finally $|\pi(G)| \leq a \times K_\omega \times |\Sigma_E| \times |G|^2 + b \times K_\eta \times |G| \leq K_\pi \times |G|^2$.

Let $K_0 = \max_{i \in [1,k]}(K_{\pi_i})$. Finally, let K_E be the constant as given by Lemma 4.1 for w_0, we define $K = \max(K_0, K_E)$. Then, for all $i \leq k$, we have: $|\pi_i(G)| \leq K \times |G|^2$ and $|w_0(G)| \leq K \times |G|^2$.

Let $\kappa(G) = (|G|, w_0(G), \pi_1(G), \ldots, \pi_k(G))$. If $G \to G'$, then $\kappa(G) > \kappa(G')$. Consider a sequence $G_1 \to G_2 \to \cdots$. For all graph G_i of the sequence, $|G_i| \leq |G_1|$. Due to previous equations, $\kappa(G_i)$ is ranging in $L = [0, |G_1|] \times [-K \times |G_1|^2, K \times |G_1|^2]^{k+1}$. Thus the result. □

5 Conclusion

The polynomial derivation height that we have proved in the last section can be reconsidered in the following way. The example of a GRS working in $O(n^k)$ can be used as a clock. Then, since each transition of a (non-size increasing) Turing-Machine can be easily simulated by graph rewriting, we can state that any PTIME-predicate can be simulated by a lexicographically weighted GRS (up to a polynomial reduction). Since lexicographically weighted confluent GRS can be computed in polynomial time (each rewriting step can be simulated in linear time), it becomes clear that lexicographically weighted GRSs actually characterize PTIME. This provides a precise description of the computational content of the method.

We have implemented a software —called GREW (`grew.loria.fr`)— based on the Graph Rewriting definition presented in this article. In [12], the software was used to produce a semantically annotated version of the French Treebank; in this experiment, the system contains 34 modules and 571 rules and the corpus is constituted of 12 000 sentences of length up to 100 words. This experiment is a large scale application which shows that the proposed approach can be used in real-size applications.

As said earlier, despite the global non-confluence of the system, we can isolate subsets of rules that are confluent and use our system of modules to benefit from this confluence in implementation. In our last experiment, 26 of our 34 modules are confluent, but confluence proofs are tedious. We leave for further work the study of the local confluence of terminating GRS and the general study of confluence of Graph Rewriting Systems.

References

[1] B. Bohnet & L. Wanner (2001): *On using a parallel graph rewriting formalism in generation*. In: *EWNLG '01: Proceedings of the 8th European workshop on Natural Language Generation*, Association for Computational Linguistics, pp. 1–11, doi:10.3115/1117840.1117847.

[2] G. Bonfante & B. Guillaume (2013): *Non-size increasing Graph Rewriting for Natural Language Processing*. to appear in *Mathematical Structures for Computer Science*.

[3] G. Bonfante, B. Guillaume, M. Morey & G. Perrier (2011): *Modular Graph Rewriting to Compute Semantics*. In: *IWCS 2011*, Oxford, UK, pp. 65–74.

[4] A. Copestake (2009): Invited Talk: *Slacker Semantics: Why Superficiality, Dependency and Avoidance of Commitment can be the Right Way to Go*. In: *Proceedings of the 12th Conference of the European Chapter of the ACL (EACL 2009)*, Association for Computational Linguistics, Athens, Greece, pp. 1–9.

[5] D. Crouch (2005): *Packed Rewriting for Mapping Semantics to KR*. In: *Proceedings of IWCS*.

[6] R. Echahed (2008): *Inductively Sequential Term-Graph Rewrite Systems*. In: *Proceedings of the 4th international conference on Graph Transformations*, ICGT '08, Springer-Verlag, Berlin, Heidelberg, pp. 84–98, doi:10.1007/978-3-540-87405-8_7.

[7] E. Godard, Y. Métivier, M. Mosbah & A. Sellami (2002): *Termination Detection of Distributed Algorithms by Graph Relabelling Systems*. In A. Corradini, H. Ehrig, H.-J. Kreowski & G. Rozenberg, editors: *ICGT*, *Lecture Notes in Computer Science* 2505, Springer, pp. 106–119, doi:10.1007/3-540-45832-8_10.

[8] E. Hyvönen (1984): *Semantic Parsing as Graph Language Transformation - a Multidimensional Approach to Parsing Highly Inflectional Languages*. In: *COLING*, pp. 517–520, doi:10.3115/980491.980601.

[9] V. Jijkoun & M. de Rijke (2007): *Learning to Transform Linguistic Graphs*. In: *Second Workshop on TextGraphs: Graph-Based Algorithms for Natural Language Processing, Rochester, NY, USA*.

[10] D.E. Knuth & P.B. Bendix (1970): *Simple word problems in universal algebras*. In J. Leech, editor: *Computational problems in abstract algebra*, Pergamon, pp. 263–277.

[11] M. Newman (1942): *On Theories With a Combinatorial Definition of "Equivalence"*. *Annals of Math.* 43(2), pp. 223–243, doi:10.2307/1968867.

[12] G. Perrier & B. Guillaume (2012): *Semantic Annotation of the French Treebank with Modular Graph Rewriting*. In Jan Hajic, editor: *META-RESEARCH Workshop on Advanced Treebanking, LREC 2012 Workshop*, META-NET, Istanbul, Turquie. Available at `http://hal.inria.fr/hal-00760577`.

[13] D. Plump (1995): *On Termination of Graph Rewriting*. In: *Proceedings of the 21st International Workshop on Graph-Theoretic Concepts in Computer Science*, WG '95, Springer-Verlag, London, UK, pp. 88–100, doi:10.1007/3-540-60618-1_68.

[14] D. Plump (1998): *Termination of Graph Rewriting is Undecidable*. *Fundamenta Informaticae* 33(2), pp. 201–209, doi:10.3233/FI-1998-33204.

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

[16] L. Tesnière (1959): *Eléments de syntaxe structurale*. Librairie C. Klincksieck, Paris.