HAL open science

Non-size increasing Graph Rewriting for Natural Language Processing

Guillaume Bonfante, Bruno Guillaume

▶ To cite this version:

Guillaume Bonfante, Bruno Guillaume. Non-size increasing Graph Rewriting for Natural Language Processing. Mathematical Structures in Computer Science, 2018, 28 (08), pp.1451 - 1484. 10.1017/S0960129518000178 . hal-00921038v2

HAL Id: hal-00921038

https://inria.hal.science/hal-00921038v2

Submitted on 22 Nov 2018

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.

Under consideration for publication in Math. Struct. in Comp. Science

Non-size increasing Graph Rewriting for Natural Language Processing

Guillaume Bonfante

Bruno Guillaume

Received January 2012

A very large amount of work in Natural Language Processing use tree structure as the first class citizen mathematical structures to represent linguistic structures such as parsed sentences or feature structures. However, some linguistic phenomena do not cope properly with trees: for instance, in the sentence *"Max decides to leave"*, *"Max"* is the subject of the both predicates *"to_decide"* and *"to_leave"*. Tree-based linguistic formalisms generally use some encoding to manage sentences like the previous example. In former papers (Bonfante et al., 2011; Guillaume and Perrier, 2012), we discussed the interest to use graphs rather than trees to deal with linguistic structures and we have shown how Graph Rewriting could be used for their processing, for instance in the transformation of the sentence syntax into its semantics. Our experiments have shown that Graph Rewriting applications to Natural Language Processing do not require the full computational power of the general Graph Rewriting setting. The most important observation is that all graph vertices in the final structures are in some sense "predictable" from the input data and so, we can consider the framework of Non-size increasing Graph Rewriting. In our previous papers, we have formally described the Graph Rewriting calculus we used and our purpose here is to study the theoretical aspect of termination with respect to this calculus. Given that termination is undecidable (cf. Plump (Plump, 1998)) in general, we define termination criterions based on weight, we prove the termination of weighted rewriting systems and we give complexity bounds on derivation lengths for these rewriting systems.

Contents

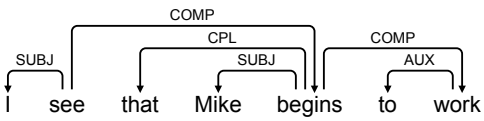
1 Introduction 2
2 Linguistic motivations 6
  2.1 Node preservation property 6
  2.2 Edge shifting 6
  2.3 Negative conditions 8
  2.4 Long distance dependencies 8
3 Graph Rewriting for NLP 9
  3.1 Graphs 9
  3.2 Graph decomposition 12

3.3 Rules 13
3.4 Graph Rewriting System 15
4 Big step semantics 15
4.1 Rules normalization 15
4.2 Big step modification 17
5 Termination 19
5.1 Termination by weight analysis 19
5.2 Termination by lexicographic weight 21
6 Properties of weighted GRS 25
6.1 The synthesis of weights 26
6.2 The derivation height of weighted GRS 26
6.3 The derivation height of lexicographically weighted GRS 28
6.4 A characterization of polynomial time 30
7 Conclusion 32
References 32
Appendix A Proof details 34

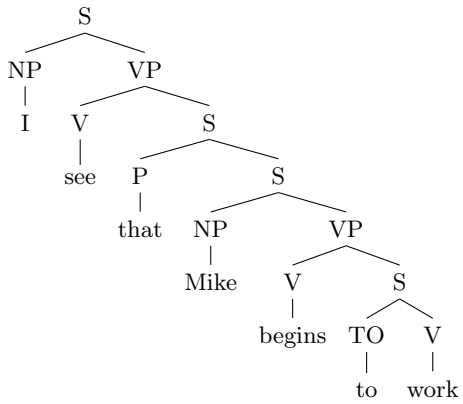
1. Introduction

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.

Linguistic theories of syntax split in two main branches. The firsts are based on the notion of *dependency structure*, the seconds on *constituents*. Dependency structures already appeared in the Ancient History, for instance in the Sanskrit grammar of Pāṇini; it is also used by Arabic grammarians of the Middle Age; a modern presentation of dependency structures is due to Tesnière (Tesnière, 1959). A dependency structure is an ordered sequence of words, together with some relations between these words. Let us consider the English sentence *"I see that Mike begins to work"*. This sentence has a dependency structure like the one below:

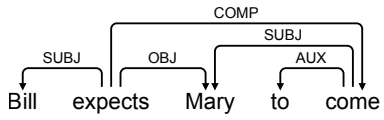


The second linguistic model of syntax is based on the notion of constituents (mainly promoted by Chomsky) in which words are grouped in a hierarchy of typed constituents.



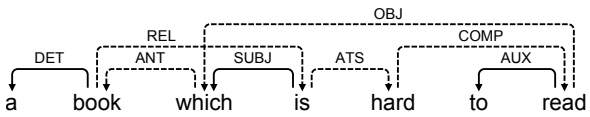
Our approach would fit both representations. For the sake of clarity, in this contribution, we will present the examples with dependency structure which we think are a little bit more intuitive.

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? Many linguistic theories argue for the tree nature of these structures which are simpler both from theoretical and practical views. However, as we shall see, it is clearly insufficient. Usually, linguists add some external structure to deal with problematic cases (they identify nodes by means of coreferences or path equations). Working directly with graphs avoids these encodings. We think that this is the best way to obtain a simple, tractable and natural computational model. Let us illustrate the limitations of tree-representations with some linguistic examples. Consider for instance the sentence *"Bill expects Mary to come"*, the node *"Mary"* is shared, being the subject of *"come"* and the object of *"expects"*:



Many examples with coordination induces sharing of some nodes: in the sentence *"Nicolas gave a book and Helen lent a newspaper about economy to Peter yesterday"*, the two verbs *"gave"* and *"lent"* shared their subject *"Nicolas"*, and the two prepositional phrases *"to Peter"* and *"yesterday"*.

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.



For the semantic representation of natural language sentences, first order logic formulae are widely used. But, natural language can be highly ambiguous; this is mainly due to the fact that scope of quantifiers are not given explicitly in sentence. To avoid to produce

many formulae for a given sentence, new representations of semantic were introduced (they are know as underspecified semantic representations). As an example, one of the underspecified semantic representations is the DMRS (Dependency Minimal Recursion Semantics) presented in (Copestake, 2009). The structure for the sentence *"The Dog whose toy the cat bit barked"* is given in Figure 1. In DMRS structure, three kind of relations are described: first, relations ARG1, ARG2 link predicate to their semantic arguments; second relation RSTR described the quantifier restrictions; the unoriented relation EQ describes scope constraints.

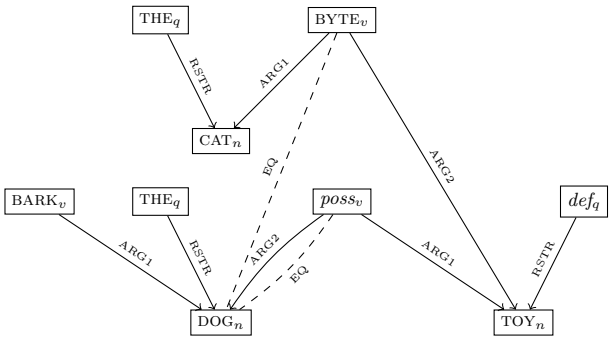


Fig. 1. DMRS structure for the sentence *"The Dog whose toy the cat bit barked"*

The UNL project (Uchida et al., 2006) is another example of a framework where graphs are used to encode deep linguistic information as an intermediate structure in a multilingual communication system.

At the syntax level, considering the fixed order of the words of the sentence, it may be possible to use some edge inversion and the transform our graphs into DAGs. However, at the semantic level there is no more fixed ordering between nodes and so there is no canonical way to turn graphs into DAGs.

To describe transformations (in both ways) between flat structures and syntactic structures or between syntactic structures and semantics structures, computational linguists have invented a lot of solutions based on many computational models. To cite some of them, we note those using finite state automata (with Markov Models, see for instance (Roche and Schabes, 1997)), those based on λ-calculus (Montague, 1970; Montague, 1973), or Natural Language Processing (NLP) specific solutions (unifications based systems like HPSG (Pollard and Sag, 1994), LFG (Kaplan and Bresnan, 1982) or Tree Adjoining Grammars (Vijay-Shanker and Joshi, 1988)).

It is somewhat surprising that Graph Rewriting Systems (GRS) have been hardly considered so far. Not so far from being extensive, we mention (Hyvönen, 1984; Bohnet and Wanner, 2001; Crouch, 2005; Jijkoun and de Rijke, 2007; Bédaride and Gardent, 2009; Chaumartin and Kahane, 2010). 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 linear time 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 (Bonfante et al., 2011), 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 (Guillaume and Perrier, 2012), the system was applied to a large corpus (the French Treebank).

In the two earlier mentioned studies, we tried to delineate what are the key features of graph rewriting in the context of Natural Language Processing. We come back to it in the next section. 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). As a matter of facts, our application performs very common transformations in NLP, which, in other words, means that the features provided by our framework can be seen as the minimal requirements in terms of expressivity of an NLP oriented Graph Rewriting Semantics.

Indeed, 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 (Rozenberg, 1997). 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. We also leave as future work the comparison of our definition with other algorithmic approaches to graph rewriting like (Janssens and Rozenberg, 1982) and (Schürr, 1997).

One of the main issues we have been faced with is to control globally computations by rewriting. Indeed, since each step of computation is fired by some local condition in the whole graph, one has no/few grip on the sequence of rewriting steps. Thus, the more rules, the more interactions between rules, and the consistency of the whole system becomes difficult to maintain. Moreover, some rules are not written by hand. Indeed, to take into account the particular behavior of some words, some rules are directly synthesized from some linguistic resources. For instance, a verb may have an intransitive form, a transitive form or both forms. This property can be extracted from syntactic lexicons which describes the sub-categorization frames for each verbs. The role of the NLP programmer is then to define the shape of the rules depending on the lexicon. To give an idea of the size of programs, our application involves 571 rules among which 379 have been produced out of the French verb valency lexicon Dicovalence (Van den Eynde and Mertens, 2003). To keep the general coherence, to ease the development and the maintenance of the system, we have organized rules in modules (applied sequentially). A module is a set of rules that is linguistically consistent and represents a particular step of the transformation. As an illustration, in our system, one of them turns a sentence in passive form to its active form. Another module resolves the anaphoric links which are internal to the sentence and determined by the syntax. In our proposal, there are 34 modules.

Generally speaking, the ambiguity of the meaning of a sentence is a very common

phenomenon which cannot be avoided. Technically, this implies that the relation between the syntax of sentences to their semantics is not functional, which implies that the graph rewriting systems themselves are not confluent. Anyway, some modules are confluent. In that case, since we can restrict computations to only one normal form, the process is much more efficient. In our system, only 8 of the 34 modules are not confluent.

The large amount of rules, the fact that they are not written by hand and the large number of modules justify to have some tools to verify some properties of modules. In Section 5, we provide two termination methods based on a weight analysis. There are two reasons we have been interested by the property of termination. 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 (Newman, 1942) which requires termination.

In Section 6, 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.

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. These examples below explain and justify the main technical choices we have made in our Graph Rewriting definition.

2.1. *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. For instance, we have to add a subject node for each infinitive verb; we also have to split some well-known words which produces several semantics items (like *"whose"* in Figure 1 and the two corresponding items in the DMRS structure $poss_v$ and def_q).

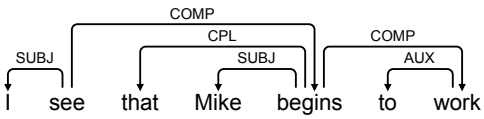
In practice, these node creations can be anticipated in some enriched input structure on which the whole transformation can be described as a non-size increasing process.

This property of node preservation is also present in many widely-used formalism. For instance, in TAG (Vijay-Shanker and Joshi, 1988), in Minimalist Grammars (Stabler, 1997) or in Abstract Categorial Grammars (de Groote, 2001), input structures are lexicalized (*i.e.* are directly linked to words of the language) and the operations used during the analysis process are non-size increasing: substitution and adjunction for TAG,

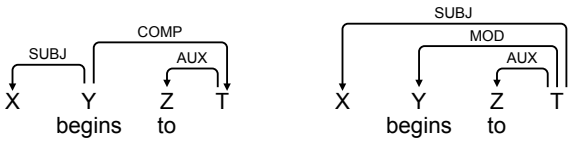
merge and move for Minimalist Grammars and linear β-reduction for Abstract Categorial Grammars.

2.2. *Edge shifting*

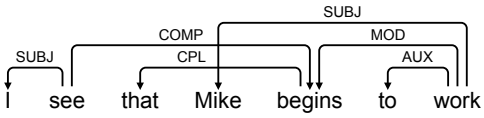
Recall the dependency structure of the sentence *"I see that Mike begins to work"*:



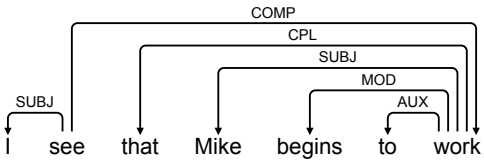
In this case, the verb *"begins"* is called a raising verb and we know that *"Mike"* is the *deep subject* of the verb *"work"*; *"begins"* being considered as a modifier of the verb. To recover this deep subject, one may imagine a local transformation of the graph which turns the first graph below into the second one.



Of course, this kind of transformation can be expressed in any graph rewriting framework. However, in our example above, a direct application of the transformation leads to the structure below:



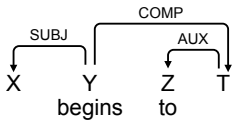
which actually is not the right structure. Indeed, the transformation should shift what the linguists call the head of the phrase *"Mike begins to work"* from the word *"begins"* to the word *"work"* with all relative edges. In that case, the transformation should produce the next structure:



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

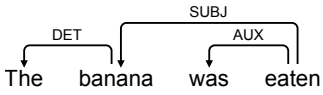
in (Echahed, 2008)) which expresses step by step the modifications applied on the input graph. The transformation described above is performed in our setting as follows:

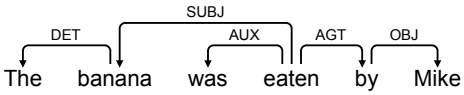


`del_edge`(Y,SUBJ,X)
`del_edge`(Y,COMP,T)
`add_edge`(T,SUBJ,X)
`add_edge`(T,MOD,Y)
`shift`(Y,T)

2.3. *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.

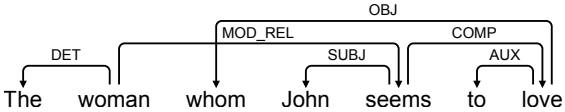




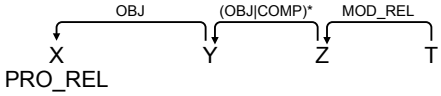
In order to switch to the corresponding active form, two different linguistic transformations have to be defined for these two sentence; but, clearly, the first graph is a subgraph of the second one. We don't want the transformation for the short passive on the left to apply on the long passive on the 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.

2.4. *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 above:



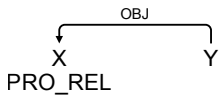
One of the steps in the semantic construction of this sentence requires to compute the antecedent of the relative pronoun *"whom"* (the noun *"woman"* in our example). The subgraph we have to search in our graph is depicted as a non-local pattern on the left and the graph modification to perform is given on the right.



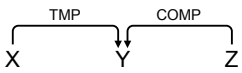
`add_edge`(X,ANT,T)

The number of OBJ or COMP relations to consider (in the relation depicted as (OBJ|COMP)* in the figure above) 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.

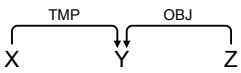
As we want to stay in the well-known framework of local rewriting, we implement such non local rules with sets of local transformations. The first rule (on the left) creates an edge TMP which serves as a pointer. The second rules updates the pointer to find the end of the chain made of COMP and OBJ labels.



`add_edge`(X,TMP,Y)

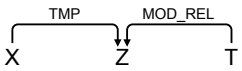


`del_edge`(X,TMP,Y)
`add_edge`(X,TMP,Z)



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

When the TMP pointer reaches the end of the chain, we can update the link between the PRO_REL and its target. In the end, we remove the temporary link TMP.



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

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 $c_X : x \in X \mapsto c \in Y$ 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 n and m. 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.

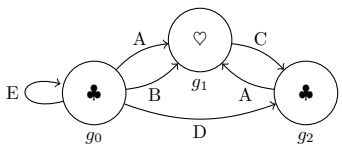
All along this paper, we suppose given two finite sets Σ_E of edge labels, Σ_N of node labels and an enumerable set $\mathcal{U}$ called *universe*, where graph nodes live.

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: $\mathcal{N} \subset \mathcal{U}$;
— 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 $\mathcal{E}$. The number of nodes of a graph G is written $|G|$. Given a graph G, two nodes $n, m \in \mathcal{N}_G$ and $e \in \Sigma_E$, let $G + \{n \xrightarrow{e} m\} \triangleq \langle \mathcal{N}_G, \ell_G, \mathcal{E}_G \cup \{n \xrightarrow{e} m\}\rangle$. We use the $+$ notation to stress the fact that $n \xrightarrow{e} m$ is not *intended* to be in G.

Example 3.1. Let $\Sigma_E = \{A, B, C, D, E\}$, $\Sigma_N = \{\clubsuit, \heartsuit\}$ and $G_0 = (\{g_0, g_1, g_2\}, \ell, \{(g_0, A, g_1), (g_2, A, g_1), (g_0, B, g_1), (g_1, C, g_2), (g_0, D, g_2), (g_0, E, g_0)\})$ with $\ell(g_0) = \ell(g_2) = \clubsuit$ and $\ell(g_1) = \heartsuit$. It is pictured as follows:



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 $\mu(E) = \{\mu(n) \xrightarrow{e} \mu(m) \mid n \xrightarrow{e} m \in E\}$.

A subgraph G of G' is a graph such that $\mathcal{N}_G \subseteq \mathcal{N}_{G'}$, $\mathcal{E}_G \subseteq \mathcal{E}_{G'}$ and $\ell_G = \ell_{G'}|_{\mathcal{N}_G}$. The graph G is said to be full in G' whenever it is a subgraph and $\mathcal{E}_G = \mathcal{E}_{G'} \cap (\mathcal{N}_G \times \Sigma_E \times \mathcal{N}_G)$. Given a subgraph G of G', we denote by $1_{G,G'} : n \in G \mapsto n \in G'$. It is an injective morphism from G to G'.

Definition 3.3 (Positive pattern and positive matching). A *positive pattern* B is

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

As shown in Section 2, negative conditions on the subgraph to match naturally arise in NLP. The positive patterns describes *what must be present in the image*, not what is *absent*. Negative conditions fulfill the gap.

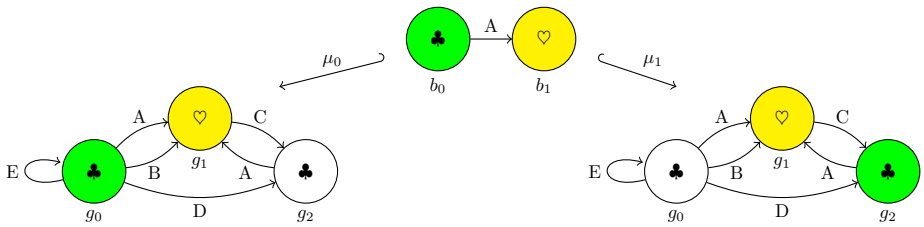
Definition 3.4 (Pattern). A *pattern* is a pair $P = \langle B, \vec{\nu} \rangle$ made of a positive pattern B and a sequence of injective morphisms $\vec{\nu} = (\nu_1 : B \hookrightarrow B_1, \ldots, \nu_k : B \hookrightarrow B_k)$ that share the graph B. The morphisms ν_i within $\vec{\nu}$ are the negative conditions.

In a pattern $P = \langle B, \vec{\nu} \rangle$, the positive pattern B plays a central role. Thus, we use in the sequel $\mathcal{N}_P$ for $\mathcal{N}_B$, $\mathcal{E}_P$ for $\mathcal{E}_B$ and so on. We do as if the pattern were a graph, that is its positive pattern. The notations should not be ambiguous. The empty sequence is $()$ and thus, $P = (B, ())$ denotes a pattern with an empty set of negative conditions.

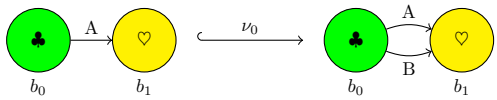
Definition 3.5 (Matching). Given a pattern $P = \langle B, \vec{\nu} \rangle$ defined as above and a graph G, a matching from P to G is a positive matching $\mu : B \hookrightarrow G$ such that there is no index $1 \leq i \leq k$, for which there is an *injective morphism* ξ_i with $\xi_i \circ \nu_i = \mu$:

$$\begin{tikzcd} B \arrow[r, hook, "\mu"] \arrow[d, hook, "\nu_i"'] & G \\ B_i \arrow[ur, hook, "\xi_i"] \arrow[ur, phantom, "\times"] & \end{tikzcd}$$

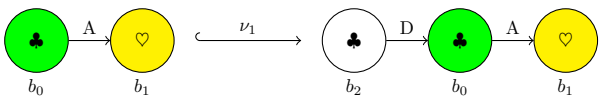
Negative conditions remove "bad matchings". To see them in action, let us come back to the graph of Example 3.1 and consider the two matchings $\mu_0, \mu_1 : B_0 \hookrightarrow G_0$.



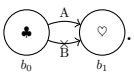
The two morphisms μ_0 and μ_1 are matchings for the pattern $\langle B_0, () \rangle$. Let us consider now the morphism $\nu_0 : B_0 \hookrightarrow B'_0$ defined by:

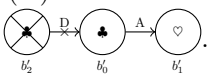


The morphism μ_0 is not a matching for the pattern $\langle B_0, (\nu_0) \rangle$. Indeed, there is a morphism $\xi_0 = [b_0 \mapsto g_0, b_1 \mapsto g_1]$ such that $\xi_0 \circ \nu_0 = \mu_0$. Let us consider the case of the morphism ν_1 defined as follows:



The morphism μ_1 is not a matching for the pattern $\langle B_0, (\nu_1)\rangle$. Indeed, the morphism $\xi_1 = [b_0 \mapsto g_2, b_1 \mapsto g_1, b_2 \mapsto g_0]$ verifies $\xi_1 \circ \nu_1 = \mu_1$.

To describe one negative condition $\nu : B \hookrightarrow B'$, we use the following convention. We "show" the graph B', crossing parts of it that are not in the image of $\nu(B)$. For the morphism ν_0, we get: 

. For the negative condition ν_1, we have: 

.

Let us end the section with the following observation. Given a pattern $P = \langle B, \vec{\nu}\rangle$, suppose that there is a negative condition ν_i among $\vec{\nu}$ which is an isomorphism. Then, there is no matchings for P. Indeed, let us suppose a positive matching $\mu : B \hookrightarrow G$. Let $\nu_i : B \hookrightarrow B_i$ be the incriminated isomorphism, there exists thus a morphism η_i such that $\eta_i \circ \nu_i = 1_B$. Set $\xi_i = \mu \circ \eta_i$. It is injective (as the composition of two injective morphisms). But $\xi_i \circ \nu_i = \mu \circ \eta_i \circ \nu_i = \mu \circ 1_B = \mu$. And thus μ is not a matching. In other words, as long as there is an isomorphism within negative conditions, there is no more matchings. Such patterns become dubious and we suppose from now on that negative conditions are never isomorphisms.

Once the condition on negative conditions is set, we can observe that the identity is a matching: given a pattern $\langle P, \vec{\nu}\rangle$, $1 : P \hookrightarrow P$ is a matching from P to P.

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 relative to the matching of some pattern P into G.

For technical reasons, we will have to consider the decomposition of some other graphs (for instance obtained after a subpart of the global transformation). To capture these cases, we need a more general definition. Let μ be an injective matching $\mu : P \hookrightarrow H$; as $\mathcal{N}_H \subset \mathcal{U}$, we can see μ as an injective function $\mu : \mathcal{N}_P \hookrightarrow \mathcal{U}$ and define the decomposition of a graph G with respect to μ as follows.

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

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

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

— the *pattern edges* set is $\mathcal{P}^\mu = \mu(\mathcal{E}_P) \cap \mathcal{E}_G$;
— the *crown edges* set contains edges which links a pattern image node to a crown node: $\mathcal{K}^\mu = \{n \longrightarrow m \in \mathcal{E}_G \mid n \in \mathcal{P}_\mu \wedge m \in \mathcal{K}_\mu\} \cup \{n \longrightarrow m \in \mathcal{E}_G \mid n \in \mathcal{K}_\mu \wedge m \in \mathcal{P}_\mu\}$;

— the *context edges* set contains edges which connect two nodes that are not in the pattern image: $\mathcal{C}^\mu = \{n \longrightarrow m \in \mathcal{E}_G \mid n \notin \mathcal{P}_\mu \wedge m \notin \mathcal{P}_\mu\}$.
— the *pattern-glued edges* set contains edges which are not pattern edges but which connect two nodes that are in the pattern image: $\mathcal{H}^\mu = \{n \longrightarrow m \in \mathcal{E} \setminus \mu(\mathcal{E}_P) \mid n \in \mathcal{P}_\mu \wedge m \in \mathcal{P}_\mu\}$.

In Figure 2, for the particular case where $H = G$ we draw a pattern, a matching of this pattern in a larger graph as an illustration of the nodes and edges decomposition (see the legend in the figure).

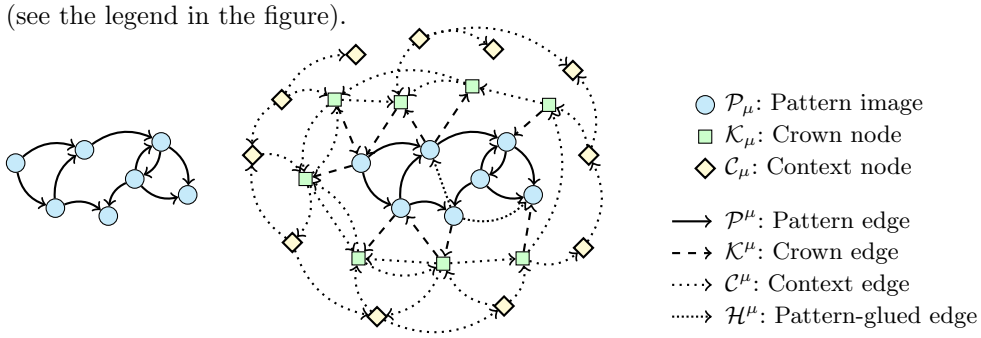


Fig. 2. A pattern, a matching and its decomposition

3.3. *Rules*

In our graph rewriting framework, the transformation of the graph is described through some atomic commands (like in (Echahed, 2008)). Commands definition refer to some pattern P and pattern nodes $\mathcal{N}_P$ are used as identifiers. The five kinds of commands are described below (where $a, b \in \mathcal{N}_P$, $\alpha \in \Sigma_N$ and $e \in \Sigma_E$):

— `change_label`(a, α)
— `del_edge`(a, e, b)
— `add_edge`(a, e, b)
— `del_node`(a)
— `shift`(a, b)

Their names speak for themselves, however, we will come back to their precise meaning below.

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 $\vec{c}$. A rule R is said to be *node-preserving* if it does not contain any `del_node` command.

Let G 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 $k = 0$,

$G \cdot () = G$. 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:

Change label: If $\mu(a) \in \mathcal{N}'$, the command $c_k = \texttt{change_label}(a, \alpha)$ changes the label of the node $\mu(a)$ of the graph G' and $G \cdot \vec{c} = (\mathcal{N}', \ell'', \mathcal{E}')$ with $\ell'' = \ell'[\mu(a) \mapsto \alpha]$. If $\mu(a) \notin \mathcal{N}'$ then $G \cdot \vec{c} = G'$.

Delete edge: If $\mu(a) \in \mathcal{N}'$, $\mu(b) \in \mathcal{N}'$ and $\mu(a) \xrightarrow{e} \mu(b) \in \mathcal{E}'$, the command $c_k = \texttt{del_edge}(a, e, b)$ deletes the edge from $\mu(a)$ to $\mu(b)$ labelled with e and $G \cdot \vec{c} = (\mathcal{N}', \ell', \mathcal{E}'')$ with $\mathcal{E}'' = \mathcal{E}' \setminus \{\mu(a) \xrightarrow{e} \mu(b)\}$. Note that, in all other cases ($\mu(a) \notin \mathcal{N}'$, $\mu(b) \notin \mathcal{N}'$ or if no edge $\mu(a) \xrightarrow{e} \mu(b)$ exists in the graph), then $G' \cdot \texttt{del_edge}(a, e, b) = G'$, which implies $G \cdot \vec{c} = G'$.

Add edge: If $\mu(a) \in \mathcal{N}'$, $\mu(b) \in \mathcal{N}'$ and $\mu(a) \xrightarrow{e} \mu(b) \notin \mathcal{E}'$, the command $c_k = \texttt{add_edge}(a, e, b)$ adds an edge from $\mu(a)$ to $\mu(b)$ labelled with e and $G \cdot \vec{c} = (\mathcal{N}', \ell', \mathcal{E}'')$ with $\mathcal{E}'' = \mathcal{E}' \cup \{\mu(a) \xrightarrow{e} \mu(b)\}$. Note that, in all other cases (for instance if such an edge already exists in the graph), then $G' \cdot \texttt{add_edge}(a, e, b) = G'$ which implies $G \cdot \vec{c} = G'$.

Delete node: If $\mu(a) \in \mathcal{N}'$, the command $c_k = \texttt{del_node}(a)$ removes the node $\mu(a)$ of the graph 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: If $\mu(a) \notin \mathcal{N}'$ or $\mu(b) \notin \mathcal{N}'$, the command $c_k = \texttt{shift}(a, b)$ does not change the graph and $G \cdot \vec{c} = G'$. Else, $\mu(a) \in \mathcal{N}'$ and $\mu(b) \in \mathcal{N}'$, the command $c_k = \texttt{shift}(a, b)$ changes in-edges of $n = \mu(a)$ starting from the crown to in-edges of $m = \mu(b)$ and all out-edges of n going to the crown to out-edges of m. We consider the decompositon of nodes of G': $\mathcal{N}' = \mathcal{P}_\mu \oplus \mathcal{K}_\mu \oplus \mathcal{C}_\mu$. 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 $p \in \mathcal{K}_\mu$, $m \xrightarrow{e} p \in \mathcal{E}''$ iff $m \xrightarrow{e} p \in \mathcal{E}'$ or $n \xrightarrow{e} p \in \mathcal{E}'$;
— for all $p \in \mathcal{K}_\mu$, $p \xrightarrow{e} m \in \mathcal{E}''$ iff $p \xrightarrow{e} m \in \mathcal{E}'$ or $p \xrightarrow{e} n \in \mathcal{E}'$;
— for all p, q such that $\{p, q\} \cap \{n, m\} = \emptyset$, $p \xrightarrow{e} q \in \mathcal{E}''$ iff $p \xrightarrow{e} q \in \mathcal{E}'$.

The commands `change_label`, `del_edge` and `add_edge` are called local commands: they modify only the edges and the nodes described in the image of the pattern. The commands `del_node` and `shift` are non-local, they can modify edges outside the pattern: pattern-glued edges and crown edges.

In a sequence of commands, there may be some commands without any effects, whatever is the graph to which the sequence is applied. First, if a command is a $\texttt{del_node}(a)$, all following commands refering to a has no effect. Second, if some sequence of commands contains two `change_label` commands which refer to the same node, the first one can be removed without changing the effect of the whole sequence. Third, if a sequence contains two commands `add_edge` or `del_edge` referring to the same triplet (a, e, b), the first one can be removed without changing the behavior of the rule. The definition below describes sequences without these spurious commands.

Definition 3.9 (Consistent sequence of commands). A sequence of commands $c_1, \ldots, c_n$ is called *consistent* if it verifies the following conditions:

— if $c_i = \texttt{del_node}(a)$ then for all j such that $i < j \leq n$ the command c_j does not refer to the node a;

— if $c_i = \texttt{change_label}(a, \alpha)$ then for all j such that $i < j \leq n$ the command $c_j \neq \texttt{change_label}(a, \alpha')$;
— if $c_i = \texttt{del_edge}(p, e, q)$ or $c_i = \texttt{add_edge}(p, e, q)$ then for all j such that $i < j \leq n$ the command $c_j \neq \texttt{del_edge}(p, e, q)$ and $c_j \neq \texttt{add_edge}(p, e, q)$.

Lemma 3.1. Any non consistent sequence $\vec{c}$ of commands can be replaced by a shorter consistent sequence $\vec{c'}$ which as the same effect: $G \rightarrow_{\langle P, \vec{c} \rangle} G'$ iff $G \rightarrow_{\langle P, \vec{c'} \rangle} G'$

Proof. From a non consistent sequence $\vec{c}$, we built a new sequence with the procedure:
— if $1 \leq i < j \leq n$, $c_i = \texttt{del_node}(a)$ and c_j refers to a, the command c_j is removed from the sequence;
— if $1 \leq i < j \leq n$, $c_i = \texttt{change_label}(a, \alpha)$ and $c_j = \texttt{change_label}(a, \alpha')$, the command c_i is removed from the sequence;
— if $1 \leq i < j \leq n$, $c_i = \texttt{del_edge}(p, e, q)$ or $c_i = \texttt{add_edge}(p, e, q)$ and $c_j = \texttt{del_edge}(p, e, q)$ or $c_j = \texttt{add_edge}(p, e, q)$, the command c_i is removed from the sequence.

It is easy to verify that each step of the procedure build a strictly smaller sequence with the same effet. The procedure can be iterated until a consistent sequence $\vec{c'}$ is reached. □

3.4. *Graph Rewriting System*

Definition 3.10 (Rewrite step). Let G a graph, $R = \langle P, \vec{c} \rangle$ a rule and $\mu : P \hookrightarrow G$ a matching. Let $G' = G \cdot_\mu \vec{c}$, then we say that G rewrites to G' with respect to the rule R and the matching μ. We write it $G \rightarrow_{R,\mu} G'$ or $G \rightarrow_R G'$ or even simply $G \rightarrow G'$.

We will also use the usual notation $\rightarrow^*$ for the transitive and reflexive closure of the relation $\rightarrow$. Given some GRS $\mathcal{G}$, a graph G is a normal form with respect to $\mathcal{G}$ if there is no H such that $G \rightarrow_{\mathcal{G}} H$. Finally, if H is a normal form such that $G \rightarrow^* H$, we write $G \rightarrow_! H$.

Let us recall that the identity is always a matching. Thus, one may apply a rule $R = \langle P, \vec{c} \rangle$ on itself. That is $1_P : P \hookrightarrow P$ is a matching, and thus we can compute $P \cdot_{1_P} \vec{c}$. This is the auto-application of a rule, and we will shorten the notation to $P \cdot \vec{c}$ in the sequel.

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

Definition 3.12. A GRS $\mathcal{G}$ induces a relation $[\![\mathcal{G}]\!]$ from graphs to graphs defined as follows:

$$(G, H) \in [\![\mathcal{G}]\!] \Longleftrightarrow G \rightarrow_! H$$

Conversely, a relation R on graphs is computed by a GRS $\mathcal{G}$ iff $[\![\mathcal{G}]\!] = R$.

In our application, the translation of the syntax to semantics splits 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 rewriting system on the outputs of the previous module.

4. Big step semantics

The previous section described semantics of rules step by step. To assert global properties of rewriting like termination, we need a big step semantics to observe the full effect of some rule rather than the effect of individual commands.

4.1. *Rules normalization*

Recall the decomposition of nodes and edges as given by Definitions 3.6 and 3.7. Node deletion modifies nodes in $\mathcal{P}_\mu$, edges in $\mu(\mathcal{E}_P)$, $\mathcal{K}^\mu$ and $\mathcal{H}^\mu$. Shifting modifies edges in the crown $\mathcal{K}^\mu$. Edge addition and edge deletion modify edges in the pattern image $\mu(\mathcal{E}_P)$ or in the pattern-glued edges $\mathcal{H}^\mu$ and renaming modify $\mathcal{P}_\mu$. Apart from the node deletion command which has a more global scope, the three sets $\{\texttt{change_label}\}$, $\{\texttt{add_edge}, \texttt{del_edge}\}$ and $\{\texttt{shift}\}$ operate respectively on the three distinct sets $\mathcal{P}_\mu$, $\mu(\mathcal{E}_P) \oplus \mathcal{H}^\mu$ and $\mathcal{K}^\mu$. Thus, if one supposes that there is no node deletion, commands may be separated according to the three sets above, and the global transformation can be computed independently.

Even with the consistent sequences (Def. 3.9), a command like `change_label`, `add_edge` or `del_edge` may be ineffective. Consider, for instance, the graph $G = (g:\alpha) \quad (h:\alpha) \circlearrowleft \mathrm{E}$. Given the pattern $P = (b:\alpha)$, it may be applied on G via the morphism $\mu_g : b \mapsto g$ or via the morphism $\mu_h : b \mapsto h$. In the first case, the command $c = \texttt{del_edge}(b, E, b)$ does not modify G. Whereas, in the second case, the command c removes the loop in $\mathcal{H}^{\mu_h}$ within G. A similar remark holds for the command `add_edge`. Uniform rules avoid such an unpredictable behavior.

Definition 4.1 (Uniform rule). We say that a rule $\langle\langle B, \vec{\nu}\rangle, \vec{c}\rangle$ is uniform if:

1 if there is an index i such that $c_i = \texttt{change_label}(p, \alpha)$, then $\ell_B(p) \neq \alpha$;
2 if there is an index i such that $c_i = \texttt{add_edge}(p, e, q)$, then $1_{B, B+\{p \xrightarrow{e} q\}} \in \vec{\nu}$;
3 if there is an index i such that $c_i = \texttt{del_edge}(p, e, q)$, then $p \xrightarrow{e} q \in B$.

Lemma 4.1. Given an uniform rule $R = \langle P, \vec{c}\rangle$. If $G \to_{R,\mu} G'$ then $\mathcal{H}^\mu$ is left unchanged in G'.

Proof. By induction on the length of the sequence $\vec{c}$. □

The non-predictable behavior of `add_edge` and `del_edge` commands definitely compromises a static analysis of GRSs, and, in particular their termination property. In uniform rules, `add_edge` and `del_edge` commands have a predictable behavior. Actually, sticking to uniform rules is not restrictive: we show that any set of rules can be replaced by an equivalent set of uniform rules.

Lemma 4.2 (Uniformisation). Let R a rule, there is a finite set of uniform rules $\mathcal{S}(R)$ such that $G \to_R G'$ iff $G \to_{\mathcal{S}(R)} G'$.

The idea of the proof is to replace a rule with an `add_edge` or `del_edge` command which may be ineffective by a couple of rules: one for the effective case, one for the other possibility. Ineffective `change_label` commands are simply removed.

Let us introduce some notations. For a rule $R = \langle P, \vec{c} \rangle$, if it is not uniform we write $\gamma(R)$ the index of the first command that makes it non uniform: *i.e* the smallest index i such that $\langle P, (c_1, \ldots, c_{i-1}) \rangle$ is uniform but $\langle P, (c_1, \ldots, c_i) \rangle$ is not. For a rule $R = \langle P, \vec{c} \rangle$, we define $\delta(R)$ to be 0 if R is uniform and $|\vec{c}| - \gamma(R)$ else. Note that in any case, $\delta(R) \leq |\vec{c}|$. We write $\vec{c}_{-j} = (c_1, \ldots, c_{j-1}, c_{j+1}, \ldots, c_{|\vec{c}|})$.

Proof. We will give below a procedure which construct from a non-uniform rule R a set $\mathcal{R}$ of at most 2 rules such that:

— for all $R' \in \mathcal{R}$, we have $\delta(R') < \delta(R)$
— for any $\mathcal{G}$, $[\![\mathcal{G} \cup \{R\}]\!] = [\![\mathcal{G} \cup \mathcal{R}]\!]$.

This procedure can be iterated to replace in a GRS any rule $R = \langle P, \vec{c} \rangle$ by a subset of at most $2^{|\vec{c}|}$ uniform rules without changing the relation induced.

Let us now describe the procedure. Let $R = \langle P, \vec{c} \rangle$ be a rule such that $\delta(R) > 0$. We consider the command c_j with $j = \gamma(R)$ (*i.e.* the first non-uniform command). We are necessarily in one of the 3 following cases:

1 $c_j = \texttt{change_label}(p, \alpha)$ and $\ell_B(p) = \alpha$, let $\mathcal{R} = \{R_1\}$ with $R_1 = \langle P, \vec{c}_{-j} \rangle$
2 $c_j = \texttt{add_edge}(p, e, q)$, and $1_{B, B+\{p \xrightarrow{e} q\}} \notin \vec{\nu}$, let $\mathcal{R} = \{R_1, R_2\}$ with
  (a)$R_1 = \langle \langle B \cup \{p \xrightarrow{e} q\}, \vec{\nu} \rangle, \vec{c}_{-j} \rangle$
  (b)$R_2 = \langle \langle B, \vec{\nu} \cup 1_{B, B+\{p \xrightarrow{e} q\}} \rangle, \vec{c} \rangle$
3 $c_j = \texttt{del_edge}(p, e, q)$, and $p \xrightarrow{e} q \notin B$, let $\mathcal{R} = \{R_1, R_2\}$ with
  (a)$R_1 = \langle \langle B, \vec{\nu} \cup 1_{B, B+\{p \xrightarrow{e} q\}} \rangle, \vec{c}_{-j} \rangle$
  (b)$R_2 = \langle \langle B \cup \{p \xrightarrow{e} q\}, \vec{\nu} \rangle, \vec{c} \rangle$

In cases 2(b) and 3(b), the pattern was changed in such a way that $\gamma(R_2) > \gamma(R)$ and as the length of the sequence of commands is the same $\delta(R_2) < \delta(R)$. In the 3 other cases, the sequence was not changed until index $j - 1$, and so $\gamma(R_1) \geq \gamma(R)$; but the length strictly decreases and we can conclude $\delta(R_1) < \delta(R)$. □

4.2. *Big step modification*

To describe the effect of a sequence of `shift` commands, we define a global function Φ which sums up in a single step the composition of the commands in the sequence.

Definition 4.2. Let P a pattern and a sequence $\vec{c}$ of commands. We define inductively the big step shift-edges function $\Phi_{\vec{c}} : \mathcal{N}_P \to \mathcal{N}_P$:

— $\Phi_\emptyset = \mathbb{1}$
— $\Phi_{\vec{c}, \texttt{shift}(m,n)} = \mathbb{1}[m \mapsto n] \circ \Phi_{\vec{c}}$
— $\Phi_{\vec{c}, c} = \Phi_{\vec{c}}$ if c is not a `shift` command

Using definition 3.6, the following lemma describes node decomposition with respect to a rule application:

Lemma 4.3 (Graph node decomposition). Given a graph G and a rule $R = \langle P, \vec{c} \rangle$

and a rule application $G \to_{R,\mu} G'$. Then, $\mathcal{N}_G = \mu(\mathcal{N}_\mathcal{P}) \oplus \mathcal{K}_\mu \oplus \mathcal{C}_\mu$ and $\mathcal{N}_{G'} = \mu(\mathcal{N}_{\mathcal{P}\cdot_{\mathbb{1}}\vec{c}}) \oplus \mathcal{K}_\mu \oplus \mathcal{C}_\mu$. If the rule is node preserving, then $\mathcal{N}_G = \mathcal{N}_{G'} = \mathcal{P}_\mu \oplus \mathcal{K}_\mu \oplus \mathcal{C}_\mu$.

Definition 4.3 (Crown image). Let $R = \langle P, \vec{c}\rangle$ a node-preserving rule and $G \to_{R,\mu} G'$ a rule application. Let $\mathcal{N}_{G'} = \mathcal{P}_\mu \oplus \mathcal{K}_\mu \oplus \mathcal{C}_\mu$ the decomposition given by Lemma 4.3. The *crown image* is the set of edges of G' which links the pattern nodes to the crown nodes:

$$\mathcal{K}'^\mu = \{n \longrightarrow m \in \mathcal{E}_{G'} \mid n \in \mathcal{P}_\mu \wedge m \in \mathcal{K}_\mu\} \cup \{n \longrightarrow m \in \mathcal{E}_{G'} \mid n \in \mathcal{K}_\mu \wedge m \in \mathcal{P}_\mu\}$$

Lemma 4.4. Let $R = \langle P, \vec{c}\rangle$ a node-preserving rule, $G \to_{R,\mu} G'$ a rule application and $\mathcal{K}'^\mu$ the crown image. Let $n \in \mathcal{P}_\mu$ and $p \in \mathcal{K}_\mu$

— $n \xrightarrow{e} p \in \mathcal{K}'^\mu$ iff there is a node n' in $\Phi_{\vec{c}}^{-1}(n)$ such that $n' \xrightarrow{e} p \in \mathcal{K}^\mu$.
— $p \xrightarrow{e} n \in \mathcal{K}'^\mu$ iff there is a node n' in $\Phi_{\vec{c}}^{-1}(n)$ such that $p \xrightarrow{e} n' \in \mathcal{K}^\mu$.

Proof. By induction on the length of the sequence of commands $\vec{c}$. □

Corollary 4.1. If a rule does not contain any `shift` command, $\mathcal{K}^\mu = \mathcal{K}'^\mu$.

Lemma 4.5. Given the notations of the preceding lemma, given an edge label $e \in \Sigma_E$, we have $|\mathcal{K}'^\mu|_e \leq |\mathcal{K}^\mu|_e$.

Proof. The preceding lemma induces a function : $T : n \xrightarrow{e} p \in \mathcal{K}'^\mu \mapsto n' \xrightarrow{e} p \in \mathcal{K}^\mu$ and $p \xrightarrow{e} n \in \mathcal{K}'^\mu \mapsto p \xrightarrow{e} n' \in \mathcal{K}^\mu$. Let us verify that the function T is injective which is a sufficient condition to prove the lemma.

Suppose that $n_1 \xrightarrow{e} p_1 \neq n_2 \xrightarrow{e} p_2 \in \mathcal{K}'^\mu$ (the case $p_1 \xrightarrow{e} n_1 \neq p_2 \xrightarrow{e} n_2$ is similar, the cases $p_1 \xrightarrow{e} n_1 \neq n_2 \xrightarrow{e} p_2$ are trivial since n_1 and p_2 are not in the same set). Let $T(n_1 \xrightarrow{e} p_1) = n_1' \xrightarrow{e} p_1$ and $T(n_2 \xrightarrow{e} p_2) = n_2' \xrightarrow{e} p_2$. The inequality $p_1 \xrightarrow{e} n_1 \neq n_2 \xrightarrow{e} p_2$ means that either $p_1 \neq p_2$ or that $n_1 \neq n_2$. If $p_1 \neq p_2$, trivially $n_1' \xrightarrow{e} p_1 \neq n_2' \xrightarrow{e} p_2$. If $n_1 \neq n_2$, then $\Phi_{\vec{c}}^{-1}(n_1) \cap \Phi_{\vec{c}}^{-1}(n_2) = \emptyset$. Thus $n_1' \in \Phi_{\vec{c}}^{-1}(n_1)$ is different to $n_2' \in \Phi_{\vec{c}}^{-1}(n_2)$. □

With the definition above, we can state how the graph decomposition is modified by a rule application.

Theorem 4.1 (Big step semantics). Given $R = \langle P, \vec{c}\rangle$ a node-preserving and uniform rule and a rule application $G \to_{R,\mu} G'$. Let $\mathcal{K}'^\mu$ the crown image and $\mathcal{E}_G = \mu(\mathcal{E}_P) \oplus \mathcal{K}^\mu \oplus \mathcal{C}^\mu \oplus \mathcal{H}^\mu$ the decomposition of $\mathcal{E}_G$ with respect to μ (Definition 3.7). Then, with $P' = P\cdot_{\mathbb{1}}\vec{c}$, edges of G' can be described as follows: $\mathcal{E}_{G'} = \mu(\mathcal{E}_{P'}) \oplus \mathcal{K}'^\mu \oplus \mathcal{C}^\mu \oplus \mathcal{H}^\mu$.

With the previous theorem, we have completely described a rewriting step when no node deletion is involved. The theorem could be generalized to node deletion, but at the price of clarity. In any case the extra-generality is not needed in the following.

The following lemma states that a rule with some node deletions modifies at most a linear number (with respect to the number of nodes in the graph) of edges. Given a rewriting step $G \to_{R,\mu} G'$, we can consider a less fine-grained decomposition of edges of G into $\mathcal{E}_G = \mathcal{C}^\mu \oplus \mathcal{Q}^\mu$ where, of course, $\mathcal{Q}^\mu = \mu(\mathcal{E}_P) \oplus \mathcal{K}^\mu \oplus \mathcal{H}^\mu$. By definition of the context, edges in $\mathcal{C}^\mu$ are unchanged during the reduction and edges of G' can also be split in two sets: $\mathcal{E}_{G'} = \mathcal{C}^\mu \oplus \mathcal{Q}'^\mu$.

Lemma 4.6 (Linear modification). Given a GRS $\mathcal{G}$, there is a constant $C > 0$ such that, for any rewriting step $G \rightarrow_{R,\mu} G'$ the two canonical corresponding edge decompositions $\mathcal{E}_G = \mathcal{C}^\mu \oplus \mathcal{Q}^\mu$ and $\mathcal{E}_{G'} = \mathcal{C}^\mu \oplus \mathcal{Q}'^\mu$ satisfy:

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

Proof. Let $C = \max\{2 \times |P|^2 \times |\Sigma_E| \mid \langle P, \vec{c} \rangle \in \mathcal{G}\}$. 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, $|\mathcal{Q}^\mu| \leq C \times (|G| + 1)$ and $|\mathcal{Q}'^\mu| \leq C \times (|G| + 1)$. □

5. Termination

We recall that a GRS is said to be (strongly) terminating whenever there is no infinite sequence $G_1 \rightarrow G_2 \rightarrow \cdots$. Actually, for non-size increasing GRS as presented above, we have immediately the decidability of non-uniform termination. That is, given some GRS $\mathcal{G}$ and some graph G, one may decide whether there is an infinite sequence $G_1 \rightarrow G_2 \rightarrow G_3 \rightarrow \cdots$. 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.

Uniform termination— given a GRS, is it terminating on all inputs?— is undecidable (Plump, 1998). For the particular case of non-size increasing graph rewriting, the problem remains open.

There is a need to define some termination method pertaining to non-size increasing GRS. Compared to standard work in termination (Plump, 1995; Godard et al., 2002), 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 subgraph of the resulting graph.

5.1. *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 (Knuth

and Bendix, 1970) 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 5.1 (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 $w(G) = w(\mathcal{E}_G)$. A *node weight* is a mapping $\eta : \Sigma_N \to \mathbb{Z}$. For a graph $G = (\mathcal{N}_G, \ell_G, \mathcal{E}_G)$, we define $\eta(G) = \sum_{n \in \mathcal{N}_G} \eta(\ell_G(n))$.

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 5.1. 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 $E \subset \mathcal{E}_G$ 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 5.2 (Weight compatibility). A weight ω is said to be *compatible* with a rule $R = \langle P, \vec{c} \rangle$ whenever $\vec{c}$ contains a node deletion command or when $\vec{c}$ preserves nodes and the three following conditions hold:

1 R is uniform,
2 $\omega(P \cdot \vec{c}) < \omega(P)$,
3 for all $e \in \Sigma_E$ such that $w(e) < 0$, for all $n \in \Phi_{\vec{c}}(\mathcal{N}_P)$, let $M_n = \Phi_{\vec{c}}^{-1}(n)$; then for all $\alpha \in \Sigma_N$, for all but possibly one element q in M_n, the set of negative condition contains $\underset{m}{\alpha} \xrightarrow{e} \underset{q}{\ell_P(q)} \in \vec{\nu}$ and for all $\alpha \in \Sigma_N$, for all but possibly one element p in M_n, the set of negative conditions contains $\underset{q}{\ell_P(q)} \xleftarrow{e} \underset{m}{\alpha} \in \vec{\nu}$.

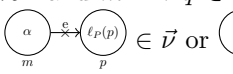
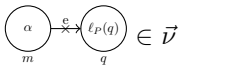
A weight is said to be compatible with a GRS $\mathcal{G}$ when it is compatible with all its rules. In that case, we say that $\mathcal{G}$ has a compatible weight. A weighted GRS is a pair $(\mathcal{G}, w)$ of a GRS and a compatible weight.

Hypothesis (1) will serve to manage glued edges in the pattern images, Hypothesis (2) will serve edges in the pattern image and Hypothesis (3) for the crown edges. One may note that when there is no `shift` commands in the rule, the Hypothesis (3) holds whatever ω is. Indeed, in that case, Φ is the identity function and all the sets M_n are singletons.

Lemma 5.2 (weight context closure). Let $(\mathcal{G}, w)$ be a weighted GRS. Suppose that $G \to G'$ is a rewrite step of $\mathcal{G}$, either $|G| > |G'|$ or $|G| = |G'|$ and $w(G) > w(G')$.

Proof. The property holds for the pattern, we prove that it is close by context. If the rule applied on G contains a `del_node` command, then, $|G| > |G'|$ and the result

holds. We suppose now that the rule contains no `del_node` command. Then $|G| = |G'|$. We prove that $\omega(G) > \omega(G')$. By Definition 3.7, $\mathcal{E}_G = \mathcal{P}^\mu \oplus \mathcal{K}^\mu \oplus \mathcal{C}^\mu \oplus \mathcal{H}^\mu$. Since the rule is uniform (Hyp. (1)), by Theorem 4.1, we can state that edges of G' are $\mathcal{E}_{G'} = \mu(P \cdot_{\mathbb{1}} \vec{c}) \oplus \mathcal{K}'^\mu \oplus \mathcal{C}^\mu \oplus \mathcal{H}^\mu$. As $\omega(P \cdot \vec{c}) < \omega(P)$ (Hyp. (2) in the definition), we only have to prove that the weight of the crown edges decrease: $w(\mathcal{K}'^\mu) \leq w(\mathcal{K}^\mu)$. Since $w(\mathcal{K}'^\mu) = \sum_{e \in \Sigma_E} \omega(e) \times |\mathcal{K}'^\mu|_e$ and $w(\mathcal{K}^\mu) = \sum_{e \in \Sigma_E} \omega(e) \times |\mathcal{K}^\mu|_e$, it is enough to prove that, for each $e \in \Sigma_E$, that $\omega(e) \times |\mathcal{K}'^\mu|_e \leq \omega(e) \times |\mathcal{K}^\mu|_e$. For a label e such that $\omega(e) \geq 0$, this follows from Lemma 4.5. Now let e such that $\omega(e) < 0$, we show that $|\mathcal{K}'^\mu|_e = |\mathcal{K}^\mu|_e$.

Ad absurdum, suppose $|\mathcal{K}'^\mu|_e < |\mathcal{K}^\mu|_e$, this implies that there are two edges in $\mathcal{K}^\mu$ with the same image in $\mathcal{K}'^\mu$. Suppose that these two edges are oriented from crown nodes to pattern nodes (the other case is similar): $m \xrightarrow{e} p \in \mathcal{K}^\mu$ and $m \xrightarrow{e} q \in \mathcal{K}^\mu$ such that $r = \Phi(p) = \Phi(q)$. But this is not possible since either  $\in \vec{\nu}$ or  $\in \vec{\nu}$ and thus, one of the two edges cannot exist. □

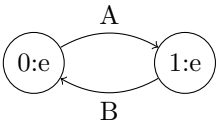
Theorem 5.1. For any weighted GRS $(\mathcal{G}, w)$, $\mathcal{G}$ is strongly terminating.

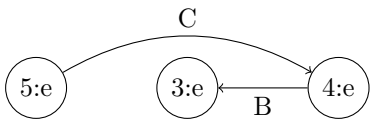
Proof. Let $\succ$ be the lexicographic ordering on $\mathbb{N} \times \mathbb{Z}$, that is $(n, v) \succ (n', v')$ iff $n > n'$ or $n = n'$ and $v > v'$. The ordering $\succ$ is well-founded on any subset $\mathbb{N} \times I$ with I a finite subset of $\mathbb{Z}$. We define $\omega(G) = (|G|, w(G))$. Lemma 5.2 shows that $G \to G'$ implies that $\omega(G) \succ \omega(G')$.

Consider a sequence $G_1 \to G_2 \to \cdots$. Since the system is non-size increasing, $|G_i| \leq |G_1|$ and consequently $K_E \times |G_i|^2 \leq K_E \times |G_1|^2$. But, by lemma 5.1(b), $-K_E \times |G_i|^2 \leq w(G_i) \leq K_E \times |G_i|$, the weights $w(G_i)$ range actually in the set $[-K_E \times |G_1|^2, \ldots, K_E \times |G_1|^2]$. Then, $(\omega(G_i))_i$ is a decreasing sequence on $\mathbb{N} \times [-K_E \times |G_1|^2, \ldots, K_E \times |G_1|^2]$, thus finite. □

Condition (3) of Definition 5.2 is necessary. Here is a counter-example of a non-terminating system with a compatible weight up to this condition.

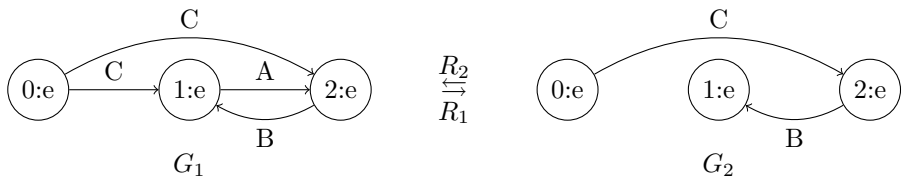
Example 5.1. Consider the two positive patterns:

$Q_1 =$ 

$Q_2 =$ 

$\vec{c_1} = [\texttt{del_edge}(0, A, 1); \texttt{shift}(0, 1)]$ $\qquad$ $\vec{c_2} = [\texttt{add_edge}(3, A, 4); \texttt{add_edge}(5, C, 3)]$

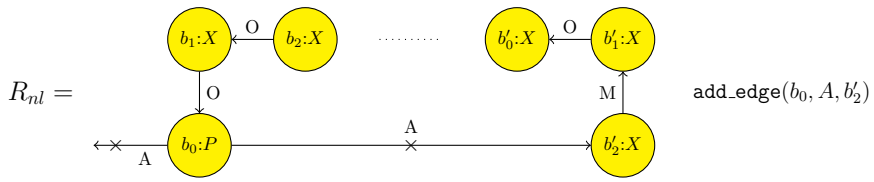
We define the GRS defined by the rules $R_1 = \langle (Q_1, ()), \vec{c_1} \rangle$ and $R_2 = \langle (Q_2, ()), \vec{c_2} \rangle$. Set $w(A) = w(B) = 1$ and $w(C) = -2$. Then, one may observe that $w(Q_1 \cdot_{\mathbb{1}} \vec{c_1}) = 1 < 2 = w(Q_1)$ and $w(Q_2 \cdot \vec{c_2}) = -2 < -1 = w(Q_2)$. However, there is an infinite sequence $G_1 \to_{R_1} G_2 \to_{R_2} G_1 \to_{R_1} G_2 \to \cdots$ with G_1 and G_2 being:



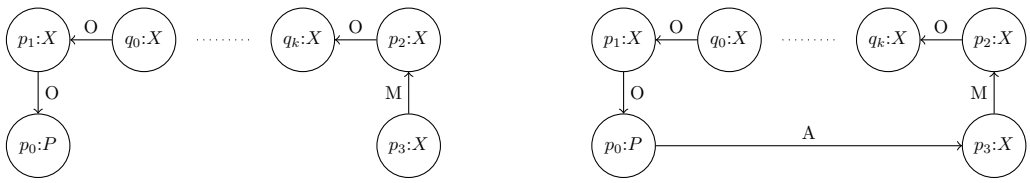
5.2. *Termination by lexicographic weight*

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 2.4. In a first step, we prove that such systems are strongly terminating but have no compatible weight. In a second step, we provide a conciliable extension of this termination proof method.

With a little abstraction, the linguistic example of Section 2.4 about long distance dependencies looks like R_{nl}:



For $k \in \mathbb{N}$, let us consider the two graphs G_k (on the left) and G'_k (on the right):



Let us suppose that a GRS $\mathcal{G}$ implements the graph rewriting form G_k to G'_k. It is clear that the shortest derivation $G_k \to H_1 \to \ldots \to H_{d_k} = G'_k$ has length $d_k \geq \Omega(k)$.

Let us observe that it is impossible to find a weight compatible with all rules of $\mathcal{G}$. Ad absurdum, suppose that w is such a weight. Then, as there is no node deletion, by Lemma 5.2, for all $k > 0$, $w(G'_k) \leq w(G_k) - d_k$. Since $w(G'_k) - w(G_k) = w(A)$, $d_k \leq -w(A)$. This means that the derivation height is bounded by a constant and this is contradictory with $d_k \geq \Omega(k)$.

In Figure 3, we propose a GRS $\mathcal{G}_{nl}$ with 4 rules which implements the non-local rule R_{nl}. With this GRS, we have: $G_k \to_{\text{Init}} \to^k_{\text{Rec}} \to_{\text{Stop}} \to^{k+1}_{\text{Clean}} G'_k$. To prove termination of this GRS, we have to use the notion of contextual weight defined below.

Definition 5.3 (Contextual weight). An *edge contextual weight* is a map $\omega : \Sigma_N \times$

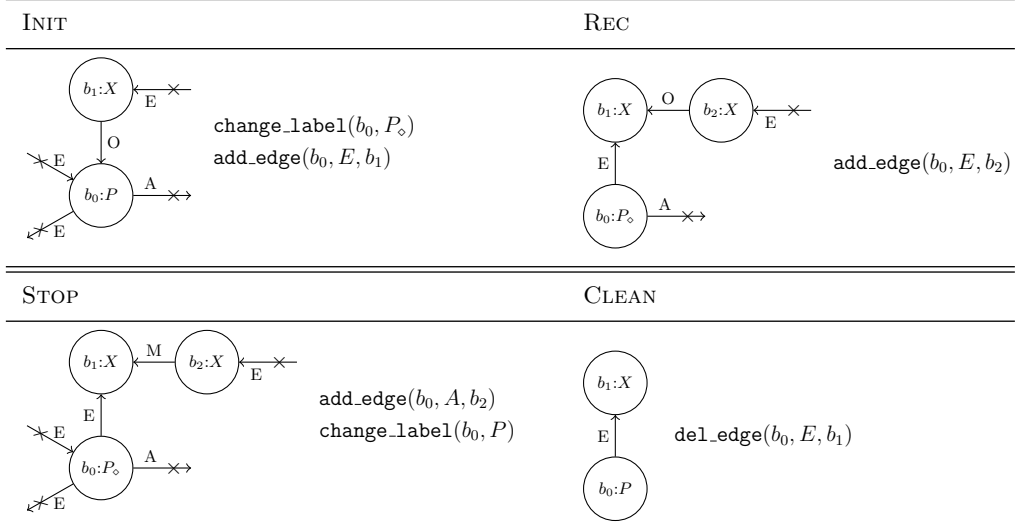


Fig. 3. Local implementation of the non-local rule

$\Sigma_E \times \Sigma_N \to \mathbb{Z}$. As for weights, for some graph $G = (\mathcal{N}, \ell, \mathcal{E})$ it extends to any set $E \subseteq \mathcal{E}$ by: $\omega(E) = \sum_{n \xrightarrow{e} m \in E} \omega(\ell(n), e, \ell(m))$. And the weight of a graph is $\omega(G) = \omega(\mathcal{E})$.

A *contextual weight* is a pair $\pi = (\omega, \eta)$ with ω an edge contextual weight and η a node weight. We define $\pi(G) = \omega(G) + \eta(G)$.

An edge contextual weight is called (α,α')-decreasing if

$$\forall e \in \Sigma_E, \forall \beta \in \Sigma_N, \qquad \begin{aligned} &\omega(\alpha, e, \beta) \geq \omega(\alpha', e, \beta) \text{ and} \\ &\omega(\beta, e, \alpha) \geq \omega(\beta, e, \alpha') \end{aligned}$$

We say that a rule R is label-compatible with a context weight ω if ω is (α, α')-decreasing whenever R contains a rule `change_label`(α, α').

Lemma 5.3 (Contextual closure). Given a contextual weight $\pi = (\omega, \eta)$, a rule $R = \langle P, \vec{c} \rangle$ which is (a) uniform, (b) contains no `shift` and (c) is *label-compatible* with ω, consider a rule application $G \to_R G'$. Then $\pi(P) > \pi(P')$ implies $\pi(G) > \pi(G')$ where $P' = P \cdot_{\mathbb{1}} \vec{c}$. If $\pi(P) \geq \pi(P')$, then $\pi(G) \geq \pi(G')$.

Proof. We do the proof for the strict inequality, the other one is analogous. To stress that the node labeling functions, ℓ in G and ℓ' in G', may differ, we write $\omega_\ell(E) = \sum_{n \xrightarrow{e} m \in E} \omega(\ell(n), e, \ell(m))$ for E in G and $\omega_{\ell'}(E) = \sum_{n \xrightarrow{e} m \in E} \omega(\ell'(n), e, \ell'(m))$ for E in G'.

We recall that $\mathcal{E}_G = \mathcal{P}^\mu \oplus \mathcal{K}^\mu \oplus \mathcal{C}^\mu \oplus \mathcal{H}^\mu$ and $\mathcal{E}_{G'} = \mathcal{P}'^\mu \oplus \mathcal{K}'^\mu \oplus \mathcal{C}'^\mu \oplus \mathcal{H}'^\mu$.

Since R contains no `shift` $\mathcal{K}^\mu = \mathcal{K}'^\mu$ by Corollary 4.1. Since R is w-compatible,

$$\omega_\ell(\mathcal{K}^\mu) = \sum_{n \xrightarrow{e} m \in \mathcal{K}^\mu} \omega(\ell(n), e, \ell(m)) \geq \sum_{n \xrightarrow{e} m \in \mathcal{K}'^\mu} \omega(\ell'(n), e, \ell'(m)) = \omega_{\ell'}(\mathcal{K}'^\mu) \qquad (1)$$

Since R is uniform, by Lemma 4.1, $\mathcal{H}^\mu = \mathcal{H}'^\mu$; again, by label-compatibility:

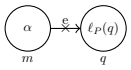
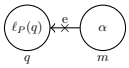
$$\omega_\ell(\mathcal{H}^\mu) \geq \omega_{\ell'}(\mathcal{H}'^\mu) \tag{2}$$

Now,

$$\begin{array}{rcll}
\pi(G) & = & \pi(P) + \omega_\ell(\mathcal{K}^\mu) + \omega_\ell(\mathcal{C}^\mu) + \omega_\ell(\mathcal{H}^\mu) + \eta(\mathcal{K}_\mu) + \eta(\mathcal{C}_\mu) & \\
& > & \pi(P') + \omega_\ell(\mathcal{K}^\mu) + \omega_\ell(\mathcal{C}^\mu) + \omega_\ell(\mathcal{H}^\mu) + \eta(\mathcal{K}_\mu) + \eta(\mathcal{C}_\mu) & \text{by hypothesis} \\
& > & \pi(P') + \omega_{\ell'}(\mathcal{K}'^\mu) + \omega_\ell(\mathcal{C}^\mu) + \omega_{\ell'}(\mathcal{H}'^\mu) + \eta(\mathcal{K}_\mu) + \eta(\mathcal{C}_\mu) & \text{by (1) and (2)} \\
& > & \pi(G') &
\end{array}$$

where the last inequality is due to the fact that the labeling functions ℓ and ℓ' are equal on $\mathcal{K}_\mu$ and $\mathcal{C}_\mu$. □

Definition 5.4 (lexicographical weight). Given an edge weight $w_0 : \Sigma_E \to \mathbb{Z}$, given k contextual weights $\pi_1 = (\omega_1, \eta_1), \ldots, \pi_k = (\omega_k, \eta_k)$ 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 $(w_0, \pi_1, \ldots, \pi_k)$ iff: either $\vec{c}$ contains a `del_node` command or R is a node-preserving rule such that:

1 R is uniform and
2 either
  (i) $w_0(P') < w_0(P)$ and
  (ii) for all $e \in \Sigma_E$ such that $w(e) < 0$, for all $n \in \Phi_{\vec{c}}(\mathcal{N}_P)$, let $M_n = \Phi_{\vec{c}}^{-1}(n)$; then for all but possibly one element q in M_n, the set of negative condition contains  $\in \vec{\nu}$ and for all but possibly one element p in M_n, the set of negative conditions contains  $\in \vec{\nu}$.
3 or $\exists j \leq k$ such that
  (i) $w_0(P') = w_0(P)$ and
  (ii) $\forall i < j$, $\pi_i(P') = \pi_i(P)$ and
  (iii) $\pi_j(P') < \pi_j(P)$ and
  (iv) $\forall i \leq j$, R is label-compatible with w_i and
  (v) $\vec{c}$ does not contain any `shift` commands.

When a weight w_0 and k contextual weights $\pi_1, \ldots, \pi_k$ are compatible with all the rules of some GRS $\mathcal{G}$, we say that $\mathcal{G}$ is *lexicographically weighted* by $(w_0, \pi_1, \ldots, \pi_k)$. Observe that Definition 5.2 corresponds to the current definition with $k = 0$. Clauses (1) and (2.a) of Definition 5.4 play the role of clauses (1) and (2) in Definition 5.2.

Example 5.2. Let us come back to Figure 3, 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 contextual weight $\pi = (\omega, \mathbf{0}_{\Sigma_N})$.

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 decrease by (2.b): one edge labeled E starting from P disappears.

Theorem 5.2. Whenever a program $\mathcal{G}$ is lexicographically weighted by $(w_0, \pi_1, \ldots, \pi_k)$, it is strongly terminating.

Proof. Given $1 \leq i \leq k$, let $\pi_i = (\omega_i, \eta_i)$, define

$$\begin{aligned} K_{\omega_i} &= \max_{(\alpha,e,\beta)\in\Sigma_N\times\Sigma_E\times\Sigma_N} |\omega_i(\alpha, e, \beta)| \\ K_{\eta_i} &= \max_{\alpha\in\Sigma_N} |\eta_i(\alpha)| \\ K_{\pi_i} &= |\Sigma_E| \times K_{\omega_i} + K_{\eta_i} \end{aligned}$$

Then, adapting Lemma 5.1(b) to the present context, we can state that $|\omega_i(G)| \leq K_{\omega_i} \times |\Sigma_E| \times |G|^2$. With Lemma 5.1(c), we have $|\eta_i(G)| \leq K_{\eta_i} \times |G|$ and finally

$$|\pi_i(G)| \leq K_{\omega_i} \times |\Sigma_E| \times |G|^2 + K_{\eta_i} \times |G| \leq K_{\pi_i} \times |G|^2$$

Let $K_0 = \max_{i\in[1,k]}(K_{\pi_i})$. Finally, let K_E be the constant as given by Lemma 5.1 for w_0, we define $K = \max(K_0, K_E)$. Then, for all $i \leq k$, we have:

$$|\pi_i(G)| \leq K \times |G|^2 \quad (3)$$

$$|w_0(G)| \leq K \times |G|^2 \quad (4)$$

Consider a sequence $G_1 \to G_2 \to \cdots$, let $I = \{1, 2, \ldots\}$ be the set of indices of this sequence. Since for all $n \in I$, $|G_n| \leq |G_1|$, due to Equations (3) and (4), any element of the sequence $(|G_n|, w_0(G_n), \pi_1(G_n), \ldots, \pi_k(G_n))_{n\in I}$ is ranging in $L = [0, |G_1|] \times [-K \times |G_1|^2, K \times |G_1|^2]^{k+1}$. On the set L, the lexicographic ordering is well-founded. Thus, to prove the finiteness of the sequence $G_1 \to G_2 \to \cdots$, it is sufficient to prove that the sequence $(|G_n|, w_0(G_n), \pi_1(G_n), \ldots, \pi_k(G_n))_{n\in I}$ is strictly decreasing for the lexicographic ordering.

Consider a step $G \to_{R,\mu} G'$ of the sequence. The rule R is compatible with the weights, so that we work by case on Definition 5.4. The Lemma 5.2 deals with case (1) and case (2.a) of the definition. It remains to look at case (2.b). So, the commands are neither node deletion nor shifts. Let edges $\mathcal{E}_G = \mu(\mathcal{E}_P) \oplus \mathcal{K}^\mu \oplus \mathcal{C}^\mu \oplus \mathcal{H}^\mu$ and $\mathcal{E}_{G'} = \mu(\mathcal{E}'_P) \oplus \mathcal{K}'^\mu \oplus \mathcal{C}^\mu \oplus \mathcal{H}^\mu$. Since there is no `shift` commands (Hyp. (2.b.v)), $\mathcal{K}^\mu = \mathcal{K}'^\mu$, by Corollary 4.1. Since the rule is uniform, $\mathcal{H}^\mu = \mathcal{H}'^\mu$ (by Lemma 4.1). Thus, $w_0(G_i) = w_0(\mathcal{K}^\mu) + w_0(\mathcal{C}^\mu) + w_0(\mathcal{H}^\mu) + w_0(\mu(P)) = w_0(\mathcal{K}^\mu) + w_0(\mathcal{C}^\mu) + w_0(\mathcal{H}'^\mu) + w_0(\mu(P')) = w_0(G_{i+1})$ where the second equality is due to Hypothesis (2.b.i).

Now, let us consider $j \in \mathbb{N}$ given by definition (2.b). If $i \leq j$, applying Lemma 5.3 (with Hyp (2.b.ii) and (2.b.iv)), we get $\pi_i(G) \geq \pi_i(G')$. With Hyp (2.b.iii) and (2.b.iv), the same lemma leads to $\pi_j(G) > \pi_j(G')$. □

6. Properties of weighted GRS

We consider two properties of weighted GRS. First, we show that we can decide whether or not a GRS has a compatible weight. Second, we give a precise bound on the length of derivation of weighted GRS.

6.1. *The synthesis of weights*

The problem of the synthesis is the following. Given a GRS $\mathcal{G}$, is there a weight w compatible with $\mathcal{G}$? The next theorem gives an answer to this question.

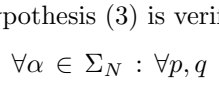
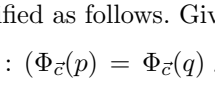
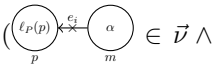
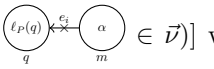
Theorem 6.1. Given a GRS $\mathcal{G}$, one may decide whether or not it has a compatible weight.

The remaining of the section is devoted to the proof of the Theorem. To sum up, we have to compute a weight that is compatible with all rules. First we can verify that either the rule preserves node or not. Since this is a syntactical question, it can be checked within the rules. If a rule is node preserving, we have to check its uniformity. Again, this is a syntactical problem. The remaining is to find a weight that respects conditions (2) and (3) for node preserving rules. Let $R_1, \ldots, R_n$ be the set of node preserving rules.

Without loss of generality, we suppose $\Sigma_E = \{e_1, \ldots, e_k\}$. Given some node-preserving rule $R = \langle\langle P, \vec{\nu}\rangle, \vec{c}\rangle$, let $P' = P \cdot \vec{c}$, finally let the Equation $E(R)$ on variables $x_1, \ldots, x_k$ be:

$$(|P|_{e_1} - |P'|_{e_1}) \times x_1 + \cdots + (|P|_{e_k} - |P'|_{e_k}) \times x_k > 0 \qquad E(R).$$

A weight w verifies Hypothesis (2) of Definition 5.2 iff $(x_i \mapsto w(e_i))_{i \leq k}$ is a solution to Equation $E(R)$ for each R. In other words, to get Hypothesis (2), we have to check the existence of $x_1, \ldots, x_k$ such that $W(x_1, \ldots x_k) \triangleq E(R_1) \wedge E(R_2) \wedge \cdots \wedge E(R_n)$.

Hypothesis (3) is verified as follows. Given $e_i \in \Sigma_E$, let $H_{e_i} \triangleq \forall R \in R_1, \ldots, R_k : \forall n \in \mathcal{N}_P : \forall \alpha \in \Sigma_N : \forall p, q : (\Phi_{\vec{c}}(p) = \Phi_{\vec{c}}(q) \wedge p \neq q) \Rightarrow [($  $\in \vec{\nu} \wedge$  $\in \vec{\nu}) \vee ($  $\in \vec{\nu} \wedge$  $\in \vec{\nu})]$ where in the formula, $\mathcal{N}_P$ refers to the nodes of the positive pattern of R, $\vec{c}$ to its commands and ν to the negative conditions of its pattern. As we can see, this formula is actually purely syntactic and its validity can be checked. The Hypothesis (3) is equivalent to $U(x_1, \ldots, x_k) \triangleq \forall i \leq k : x_i < 0 \Rightarrow H_{e_i}$.

So, a weight is compatible with $R_1, \ldots, R_n$ whenever $\exists x_1, \ldots, x_k : W(x_1, \ldots, x_k) \wedge U(x_1, \ldots, x_k)$ is satisfiable. One may observe that this formula is actually defined in Presburger's arithmetic. Thus, the existence of a weight is decidable. It is clear that the proof, although tedious, can be extended to contextual weights. Thus,

Theorem 6.2. Given a GRS $\mathcal{G}$, one may decide whether or not it has a compatible lexicographic weight.

6.2. *The derivation height of weighted GRS*

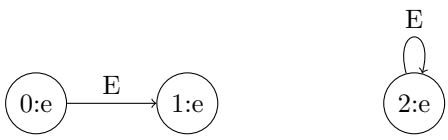
Definition 6.1. Given a strongly terminating GRS $\mathcal{G}$ and a graph G, the derivation height for G, next denoted $h_{\mathcal{G}}(G)$, is the length of the longest derivation $G \to G_1 \to \cdots \to G_k$ starting from G. The derivation height of $\mathcal{G}$, next denoted $h_{\mathcal{G}}(n)$, is defined by: $h_{\mathcal{G}}(n) = \max\{h_{\mathcal{G}}(G) \mid |G| \leq n\}$.

The proof of Theorem 5.1 actually suggests a cubic upper bound on derivation height.

We establish in this section that the exact—both an upper and a lower—bound on derivation height is quadratic.

Theorem 6.3. Given a weighted GRS $(\mathcal{G}, w)$, then the derivation height of $\mathcal{G}$ is at most quadratic. Furthermore, this quadratic bound is a lower bound, that is there is a GRS $\mathcal{G}$ with a compatible weight such that $h_{\mathcal{G}}(n) \geq \Omega(n^2)$.

Proof. We begin to show the lower bound. Let $\Sigma_E = \{E\}$, $\Sigma_N = \{e\}$. Consider the two rules GRS $\mathcal{G}$ defined by the two positive patterns:



The rules are $(\langle P_0, ()\rangle, [\texttt{del_edge}(0, E, 1)])$ and $(\langle P_\epsilon, ()\rangle, [\texttt{del_edge}(2, E, 2)])$. Then, since the clique C_n of size n has n^2 edges, the derivation height $h_{\mathcal{G}}(C_n) = n^2$. The lower bound follows from the definition.

We prove now the upper bound, that is given a GRS $\mathcal{G}$ and a compatible weight w, then $h_{\mathcal{G}}(n) \leq O(n^2)$. Let C be the constant as defined by Lemma 4.6, let $K = \max(1, K_w)$ (we recall that $K_w = \max_{e \in \Sigma_E}(|w(e)|)$). Finally, let $H = \max\{n \mid (P, c_1, \ldots, c_n) \in \mathcal{G}\}$. Let $A = 2 \times K \times C \times (H + 1) + 1$. Let Ω be the 'energy function' defined on graphs $\Omega(G) = w(G) + A \times |G|^2$.

Suppose that $G \rightarrow_{R,\mu} G'$, we prove that $\Omega(G) > \Omega(G')$. First, if there is no node deletion, then, $w(G) > w(G')$, $|G| = |G'|$ and the result follows. Suppose now the there is a node deletion command. From the definition of H and the fact that there is a `del_node` command, it is clear that

$$|G'| < |G| \leq |G'| + H. \tag{5}$$

From Lemma 4.6, we have the decompositions $\mathcal{E}_G = \mathcal{C}^\mu \oplus \mathcal{Q}^\mu$ and $\mathcal{E}_{G'} = \mathcal{C}^\mu \oplus \mathcal{Q}'^\mu$. Then $w(G) = w(\mathcal{E}_G) = w(\mathcal{C}^\mu) + w(\mathcal{Q}^\mu)$ and $w(G') = w(\mathcal{E}_{G'}) = w(\mathcal{C}^\mu) + w(\mathcal{Q}'^\mu)$.

$$w(G') - w(G) = w(\mathcal{Q}'^\mu) - w(\mathcal{Q}^\mu). \tag{6}$$

Due to Lemma 4.6, both inequality $|\mathcal{Q}^\mu| \leq C \times (|G| + 1)$ and $|\mathcal{Q}'^\mu| \leq C \times (|G| + 1)$ hold. According to Lemma 5.1(a), $w(\mathcal{Q}'^\mu) \leq K_w \times |\mathcal{Q}'^\mu| \leq K \times C \times (|G| + 1)$ and $w(\mathcal{Q}^\mu) \leq K_w \times |\mathcal{Q}^\mu| \leq K \times C \times (|G| + 1)$ and so $w(\mathcal{Q}'^\mu) - w(\mathcal{Q}^\mu) \leq 2 \times K \times C \times (|G| + 1)$. Injecting the inequality in Equation 6, we get:

$$\begin{aligned} w(G') - w(G) &\leq 2 \times K \times C \times (|G| + 1) && (7)\\ &\leq 2 \times K \times C \times |G'| + 2 \times K \times C \times (H + 1) \qquad \text{due to Eq 5} && (8)\\ &< 2 \times A \times |G'| + A && (9) \end{aligned}$$

To conclude, if there is a deletion rule,

$$\begin{array}{rcl}\Omega(G) & = & w(G) + A \times |G|^2 \\ & \geq & w(G) + A \times (|G'|+1)^2 \text{ due to Eq 5} \\ & \geq & w(G') + A \times |G'|^2 + A \times (2 \times |G'|+1) + w(G) - w(G') \\ & \geq & \Omega(G') + 2 \times A \times |G'| + A + w(G) - w(G') \\ & > & \Omega(G') \qquad \text{due to Eq 9.}\end{array}$$

Now, for all graph G, due to Lemma 5.1, we have: $-K \times |\Sigma_E| \times |G|^2 \leq w(G) \leq K \times |\Sigma_E| \times |G|^2$ and then $(A - K \times |\Sigma_E|) \times |G|^2 \leq \Omega(G) \leq (A + K \times |\Sigma_E|) \times |G|^2$. Since $\Omega(G) > \Omega(G')$ for all $G \to G'$, given some derivation $G_0 \to G_1 \to \cdots \to G_k$, we can state that $k \leq \Omega(G_0) - \Omega(G_k)$. After injection of the inequations above, we get $k \leq (K \times |\Sigma_E| + A) \times |G_0|^2 + (K \times |\Sigma_E| - A) \times |G_k|^2$. But, $|G_k| \leq |G_0|$ implies $k \leq 2 \times K \times |\Sigma_E| \times |G_0|^2$ which itself establishes the quadratic bound. □

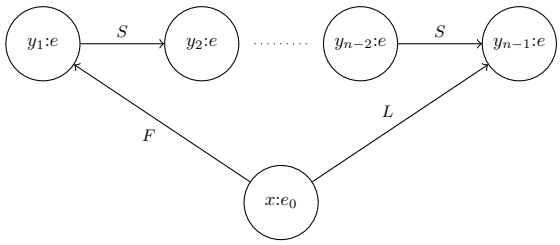
6.3. *The derivation height of lexicographically weighted GRS*

We prove the following theorem:

Theorem 6.4. If a program $\mathcal{G}$ is compatible with the lexicographic weight $(w_0, \pi_1, \ldots, \pi_k)$, then its derivation height is at most polynomial. The bound is tight, that is for all $k > 0$, there is a GRS whose derivation height is $O(n^k)$.

In the proof of the termination of lexicographically weighted programs, we have seen that any sequence $G_1 \to \cdots \to G_n$ is such that $(|G_i|, w_0(G_i), \pi_1(G_i), \ldots, \pi_k(G_i))_{i \in [1,n]}$ range in $L = [0, |G_1|] \times [-K \times |G_1|^2, K \times |G_1|^2]^{k+1}$. Thus, $n \leq |L| = (|G_1|+1) \times (2 \times |G_1|^2 + 1)^{k+1}$ which is a polynomial with respect to $|G_1|$. The upper bound follows. Contrarily to the previous section, the bound given here is not precise, but sufficient with respect to the theorem statement.

For the lower bound, we build a graph rewriting system which simulates a k-counters machine. In the graphs we consider, edges are used to encode a k digits representation in base n. The starting graph correponds to the representation $\underbrace{(n-1) \cdot (n-1) \cdots (n-1)}_{k}$ and rules are drawn to decrease the given encoding one by one downto $\underbrace{0 \cdots 0}_{k}$; this ensures that the rewriting process uses at least n^k steps. All graphs we will consider are built from the skeleton graph (below) by adding edges going from node x to one of the nodes $\{y_1, \ldots, y_{n-1}\}$ and labeled by an integer between 1 and k.



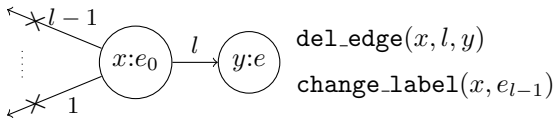
Such a graph encodes a k digits representation in base n in the following way: the digit of rank j is the number of edges labelled j in the graph. The skeleton graph corresponds to the encoding of $0 \cdots 0$. The graph encoding $(n-1) \cdots (n-1)$ is obtained by adding to the skeleton graph the edges $(x \xrightarrow{\ell} y_m)$ for all $1 \leq \ell \leq k$ and $1 \leq m \leq n-1$.

This drives us to the definitions below (where F stands for "first", S for "successor" and "L" for last). Let $\Sigma_E = \{1, \ldots, k, F, S, L\}$, let $\Sigma_N = \{e, e_0, \ldots, e_{k-1}\}$. There are four kinds of rules intended to implement the predecessor operation. That is, in terms of counters, to transform the graph corresponding to $a_k \cdots a_l \cdot 0 \cdots 0$ into $a_k \cdots (a_l - 1) \cdot (n-1) \cdots (n-1)$, one applies rules:

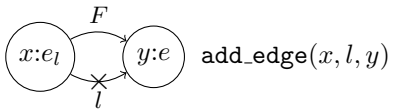
$$D_l, F_{l-1}, (S_{l-1})^{n-2}, L_{l-1}, F_{l-2}, (S_{l-2})^{n-2}, L_{l-2}, \ldots, F_1, (S_1)^{n-2}, L_1$$

D_l decreases by one digit l while the further rules update digits $1 \ldots (l-1)$ from 0 to $n-1$. More precisely, rules F_j, S_j increase by one counter number j and L_j gives hand to the update of the lower bit.

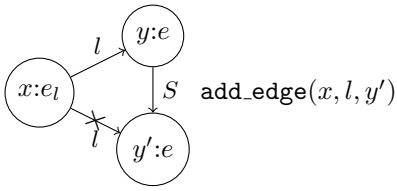
— For all l such that $1 \leq l \leq k$, the rule D_l (given below) removes a l-edge provided there is no edge with a smaller label starting from x. The node label of x is changed to enforce the construction of the full set of edges with labels 1, ..., $l-1$.



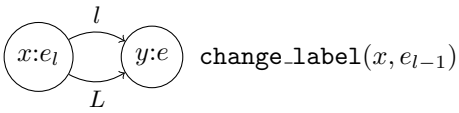
— The F-edge of the pattern ensures that y_1 is the first node to receive an l-edge. For all l such that $1 \leq l \leq k-1$, the rule F_l is given by:



— For all l such that $1 \leq l \leq k-1$, the rule S_l is given by:



— For all l such that $1 \leq l \leq k-1$, the rule L_l is given by:



The graph rewriting system is lexicographically weighted by $(\mathbf{0}, \pi_k, \ldots, \pi_1)$, the contextual weights π_j being defined by $\pi_j = (\omega_j, \eta_j)$ with

— $\omega_j = \mathbf{0}_{\Sigma_N \times \Sigma_E \times \Sigma_N}[(e_j, j, e) \mapsto -2, (e_{j-1}, j, e), \mapsto 2, \ldots, (e_0, j, e) \mapsto 2]$;
— $\eta_j = \mathbf{0}_{\Sigma_E}[e_{j-1} \mapsto 1]$.

The verification of the compatibily of the weights with the graph rewriting system is given in the appendix.

6.4. *A characterization of polynomial time*

We have defined in Section 3, Definition 3.12 a notion of semantics, that is which relations are computed by our notion of rewriting. To evaluate the strength of the lexicographic order, we propose to describe the set of relations computed by such GRS through (implicit) computational complexity. To make the relation between computations on graphs and functions on words (as done in complexity theory), we introduce the following notion of computation. Given an alphabet Σ, a word $w = a_1, \ldots, a_k$ in Σ^* is encoded as a graph G_w:

on the vertex labels $\Sigma \cup \{\texttt{nil}\}$, the latter label serves as an endmarker. The edge label is suc. It is clear that the function $w \mapsto G_w$ is injective, thus, given such a graph G, we can define $\overline{G}$ to be the unique (if it exists) word w such that $G = G_w$. One may observe that $\overline{G_w} = w$ for all words in Σ^*.

Consider for a while a GRS $\mathcal{G}$ working on the set $W = \{G_w \mid w \in \Sigma^*\}$. Suppose that $(G_w, G_u) \in [\![\mathcal{G}]\!]$. Due to the non-size increasing property of our notion of rewriting, the size $|u|$ of u is smaller than the size $|w|$ of w. This notion of computation–though interesting–restricts to non-size increasing relation. Such relations arise quite usually in complexity theory. Maybe the most remarkable result comes from Hofmann in the context of linear logic (Hofmann, 1998) where he describes non-size increasing functions computable in polynomial time. In some way, with the size restriction, the best we can offer is a characterization of functions computable within linear space. However, a machine working in linear space may take exponential time, so that a direct simulation would involve exponential derivation height. Since we have a polynomial derivation length on GRS with lexicographic weight, there is no hope for such a simulation.

To decouple size issues from time issues, we propose a little bit more liberal notion of computation, by padding inputs by a polynomial number of blank symbols. Let us extend the alphabet Σ by an extra blank letter $\Box$. Let F be a vertex label. Given some word w, $F[w]$ denote the unique graph made of one node F pointing to G_w:

Definition 6.2. A function $\phi : \Sigma^* \to \Sigma^*$ is computed up to padding by a *confluent* GRS $\mathcal{G}$ iff there is a polynomial P such that for all words $w \in \Sigma^*$:

$$F[G_{w \cdot \Box^{P(|w|)}}] \to^! G_{\phi(w)}.$$

We state that:

Theorem 6.5. Functions computed by a polynomial-time Turing Machine are exactly functions computed up to padding by GRS compatible with a lexicographical weight.

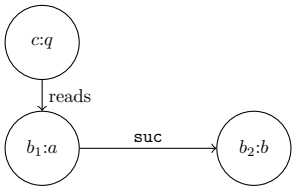
Proof. First, we prove that any function computed (up to padding) by a GRS is computable in polynomial time. So, let us suppose $w \in \Sigma^*$ has size n. The initial graph $F[w]$ has size $Q(n) = P(n)+n+2$ where P is the padding polynomial. Thus, all graphs along a computation have size bounded by $Q(n)$, that is are polynomially bounded with respect to the size of the input word w. By Theorem 6.4, we can state that the derivation height of $\mathcal{G}$ on $F[w]$ is bounded by $R(P(n) + n + 2)$ for some polynomial R.

Since the GRS is supposed to be confluent, to compute the normal form of $F[w]$, it is sufficient to simulate the rewriting process. Finding a redex in a graph takes polynomial time (this is the sub-isomorphism problem for some fixed patterns) in the size of the current graph. The graph transformations take linear time (each command taking linear time) in the size of the current graph. Now, we conclude: we apply polynomially many steps with respect to n, each of which takes polynomial time with respect to a the size of a graph which itself has polynomial size with respect to the input words. Thus, the process takes polynomial time.

In the other direction, the key idea is to use the GRS of the preceding section as clocks to simulate the computation of a Turing Machine.

First, without loss of generality, we may suppose that a Turing Machine working in polynomial time uses only a right-lateral tape. That is, it never goes to the left of its input (see for instance the book (Jones, 1997)). Consider such a machine M working in polynomial time for some polynomial P. If the $\square$ letter corresponds to the blank symbol of the Turing Machines, we can suppose that a run of M on the initial tape $w_{pad} = w \cdot \square^{P(|w|)}$ never reaches the last symbol of w_{pad}. In other words, such a machine runs in constant space on its initial configuration.

This being said, any step of computation of a Turing Machine $M = \langle Q, \Sigma, q_0, F, \delta \rangle$ may be simulated by a graph rule. For instance, consider the transition $\delta(q, a) = (q', a', \text{Right})$, that is in state q, if the current letter is an 'a', rewrite it by a b, go to the right, and update state to q'. This is expressed by the rules:



$\texttt{del_edge}(c, \text{reads}, b_1)$
$\texttt{add_edge}(c, \text{reads}, b_2)$
$\texttt{change_label}(c, q')$
$\texttt{change_label}(b_1, a')$.

with $b \in \Sigma$. With the precaution we took at the beginning on the size of inputs, such a step by step simulation will eventually output the result of the computation of the Turing Machine after at most $P(|w|)$ steps.

Moreover, for any node weight being constant on $\Sigma \cup Q$, for any edge weight contant on $\{\text{reads}, \text{suc}\}$, for any edge contextual weight constant on $Q \times \{\text{reads}, \texttt{suc}\} \times \Sigma$, the weight of the left hand side of these rules equals their right hand side. Moreover, these

rules do not contain any `shift` commands. It remains to glue clock rules (which will be lexicographically ordered) to the preceding ones to get both:

— the termination of the system in time $P(|w|)$ by a lexicographic weight,
— the simulation of the Turing Machine.

□

7. Conclusion

We have implemented a software —called GREW (`grew.loria.fr`)— based on the Graph Rewriting definition presented in this article. In (Guillaume and Perrier, 2012), 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.

Acknowledgement. The authors thank the referees for their helpful comments. Some of them lead to substantial improvements.

References

Bédaride, P. and Gardent, C. (2009). Semantic Normalisation : a Framework and an Experiment. In *8th International Conference on Computational Semantics - IWCS 2009*, Tilburg Netherlands.

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

Bonfante, G., Guillaume, B., Morey, M., and Perrier, G. (2011). Modular graph rewriting to compute semantics. In *IWCS 2011*, pages 65–74, Oxford, UK.

Chaumartin, F.-R. and Kahane, S. (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. (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)*, pages 1–9, Athens, Greece. Association for Computational Linguistics.

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

de Groote, P. (2001). Towards Abstract Categorial Grammars. In *39th Annual Meeting of the Association for Computational Linguistics and 10th Conference of the European Chapter of the Association for Computational Linguistics,Toulouse, France*, pages 148–155.

Echahed, R. (2008). Inductively sequential term-graph rewrite systems. In *Proceedings of the 4th international conference on Graph Transformations*, ICGT '08, pages 84–98, Berlin, Heidelberg. Springer-Verlag.

Godard, E., Métivier, Y., Mosbah, M., and Sellami, A. (2002). Termination detection of distributed algorithms by graph relabelling systems. In Corradini, A., Ehrig, H., Kreowski, H.-J., and Rozenberg, G., editors, *ICGT*, volume 2505 of *Lecture Notes in Computer Science*, pages 106–119. Springer.

Guillaume, B. and Perrier, G. (2012). Annotation sémantique du French Treebank à l'aide de la réécriture modulaire de graphes. In *Traitement Automatique des Langues Naturelles (TALN)*, Grenoble, France.

Hofmann, M. (1998). Linear types and non-size-increasing polynomial time computation. In *Information and Computation*, pages 464–473.

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

Janssens, D. and Rozenberg, G. (1982). Graph grammars with neighbourhood-controlled embedding. *Theoretical Computer Science*, 21(1):55 – 74.

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

Jones, N. D. (1997). *Computability and complexity: from a programming perspective*. MIT Press, Cambridge, MA, USA.

Kaplan, R. M. and Bresnan, J. (1982). Lexical-functional grammar: A formal system for grammatical representation. In Bresnan, J., editor, *The Mental Representation of Grammatical Relations*, pages 173–281. MIT Press, Cambridge, MA.

Knuth, D. and Bendix, P. (1970). Simple word problems in universal algebras. In Leech, J., editor, *Computational problems in abstract algebra*, Pergamon.

Montague, R. (1970). Universal grammar. *Theoria*, 36(3):373–398.

Montague, R. (1973). The proper treatment of quantification in ordinary english. *Formal Semantics*, pages 221–242.

Newman, M. (1942). On theories with a combinatorial definition of "equivalence". *Annals of Math.*, 43(2):223–243.

Plump, D. (1995). On termination of graph rewriting. In *Proceedings of the 21st International Workshop on Graph-Theoretic Concepts in Computer Science*, WG '95, pages 88–100, London, UK. Springer-Verlag.

Plump, D. (1998). Termination of graph rewriting is undecidable. *Fundamenta Informaticae*, 33(2):201–209.

Pollard, C. and Sag, I. A. (1994). *Head-Driven Phrase Structure Grammar*. The University of Chicago Press, Chicago.

Roche, E. and Schabes, Y., editors (1997). *Finite-State Language Processing*. Bradford Book. MIT Press, Cambridge, Massachusetts, USA.

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

Schürr, A. (1997). Programmed graph replacement systems. In *In Rozenberg, G. (Ed.), Handbook on Graph Grammars: Foundations*, pages 479–546. World Scientific.

Stabler, E. (1997). Derivational minimalism. In Retoré, C., editor, *Logical Aspects of Computational Linguistics, LACL'96*, volume 1328 of *LNCS/LNAI*, pages 68–95. Springer-Verlag.

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

Uchida, H., Zhu, M., Senta, T., and Foundation, U. (2006). *UNL: Universal Networking Language*. UNDL Foundation, International Environment House.

Van den Eynde, K. and Mertens, P. (2003). La valence : l'approche pronominale et son application au lexique verbal. *French Language Studies*, 13:63–104.

Vijay-Shanker, K. and Joshi, A. K. (1988). Feature structures based tree adjoining grammars. In *COLING*, pages 714–719.

Appendix A. Proof details

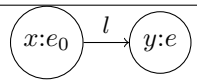
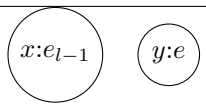
We give here the details of the end of the proof of the theorem 6.4. We have to prove that the graph rewriting system described in the proof is lexicographically weighted by $(\mathbf{0}, \pi_k, \ldots, \pi_1)$, the contextual weights π_j being defined by $\pi_j = (\omega_j, \eta_j)$ with

— $\omega_j = \mathbf{0}_{\Sigma_N \times \Sigma_E \times \Sigma_N}[(e_j, j, e) \mapsto -2, (e_{j-1}, j, e), \mapsto 2, \ldots, (e_0, j, e) \mapsto 2]$;

— $\eta_j = \mathbf{0}_{\Sigma_E}[e_{j-1} \mapsto 1]$.

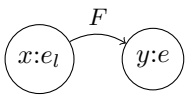
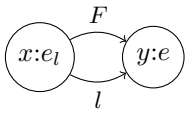
It is suffisant to prove that each rule is compatible (definition 5.1) with $(\mathbf{0}, \pi_k, \ldots, \pi_1)$. No rule contains a `del_node` command and the node weight is $\mathbf{0}$, so none of the hypothesis (1) and (2.a) apply. Les us prove that all rules verify hypothesis (2.b). As node weight is $\mathbf{0}$, (2.b.i) is verified for all rules; as there is no `shift` commands, (2.b.v) is also verified for all rules. It remains to check clauses (2.b.ii), (2.b.iii) and (2.b.iv). Let us consider each rule in turn.

— Rule D_l: the clause (2.b) is verified with $j = l$. We have to check that $\pi_l(D_l) < \pi_l(D'_l)$ and that for $i > l$, $\pi_i(D_l) = \pi_i(D'_l)$. The different cases are given in the table below and so, (2.b.ii) and (2.b.iii) are verified for rule D_l.

	(2.b.ii)		(2.b.iii)			
	$i > l$		$i = l > 1$		$i = l = 1$	
	ω_i	η_i	ω_i	η_i	ω_i	η_i
$D_l =$ 	0	0	2	0	2	1
$D'_l =$ 	0	0	0	1	0	1

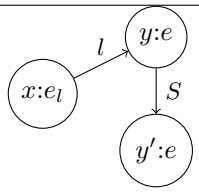
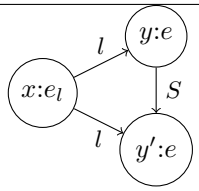
For the clause (2.b.iv), we have to check that for all $i \geq l > 1$, w_i is (e_0, e_{l-1})-decreasing. The only case where a crown-edge changes of weight during the operation `change_label`(e_0, e_{l-1}) is the case of an edge (e_0, m, e) which becomes (e_{l-1}, m, e). In this case, $w_i(e_0, m, e) = 2$ and $w_i(e_0, m, e) \geq w_i(e_{l-1}, m, e)$ because 2 is the maximal weight value.

— Rule F_l: the clause (2.b) is verified with $j = l$. We have to check that $\pi_l(F_l) < \pi_l(F'_l)$ and that for $i > l$, $\pi_i(F_l) = \pi_i(F'_l)$. The different cases are given in the table below and so, (2.b.ii) and (2.b.iii) are verified for rule F_l.

	(2.b.*ii*)				(2.b.*iii*)	
	$i > l+1$		$i = l+1$		$i = l$	
	ω_i	η_i	ω_i	η_i	ω_i	η_i
$F_l =$ 	0	0	0	1	0	0
$F'_l =$ 	0	0	0	1	-2	0

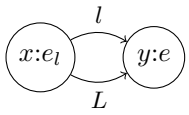
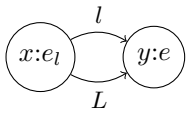
There is no `change_label` in F_l and so (2.b.iv) is necessarily verified.

— Rule S_l: the clause (2.b) is verified with $j = l$. We have to check that $\pi_l(S_l) < \pi_l(S'_l)$ and that for $i > l$, $\pi_i(S_l) = \pi_i(S'_l)$. The different cases are given in the table below and so, (2.b.ii) and (2.b.iii) are verified for rule S_l.

	(2.b.*ii*)				(2.b.*iii*)	
	$i > l+1$		$i = l+1$		$i = l$	
	ω_i	η_i	ω_i	η_i	ω_i	η_i
	0	0	0	1	-2	0
	0	0	0	1	-4	0

There is no `change_label` in S_l and so (2.b.iv) is necessarily verified.

— Rule L_l: the clause (2.b) is verified with $j = l+1$. We have to check that $\pi_{l+1}(L_l) < \pi_{l+1}(S'_l)$ and that for $i > l+1$, $\pi_i(L_l) = \pi_i(S'_l)$. The different cases are given in the table below and so, (2.b.ii) and (2.b.iii) are verified for rule L_l.

	(2.b.ii)		(2.b.iii)	
	$i > l+1$		$i = l+1$	
	ω_i	η_i	ω_i	η_i
	0	0	0	1
	0	0	0	0

For the clause (2.b.iv), we have to check that for all $i \geq l+1$, w_i is (e_l, e_{l-1})-decreasing. The only case where a crown-edge changes of weight during the operation `change_label`(e_l, e_{l-1}) is the case of an edge (e_l, m, e) which becomes (e_{l-1}, m, e). Now, if $m = i$, $w_i(e_l, m, e) = 2 = w_i(e_{l-1}, m, e)$ and if $m \neq i$, $w_i(e_l, m, e) = 0 = w_i(e_{l-1}, m, e)$.

This achieves to prove the Theorem 6.4.