LaBRI Research Report No. Compiled on April 11, 2001
The final version of this paper has been published in *Fundamenta Informaticae* **27-1** (1996), 1–26.

EXPANDING GRAPH RELABELING SYSTEMS HAVE THE POWER OF RECURSIVE ENUMERABILITY

Éric SOPENA[1]
LaBRI, Univ. Bordeaux I, 351 cours de la Libération, 33405 Talence, France.

Abstract. Graph relabeling systems (GRS's) have been introduced as a suitable tool for coding and proving sequential or distributed algorithms on graphs or networks. These systems do not change the underlying structure of the graph on which they work, but only the labeling of its components (edges or vertices). Each relabeling step is fully determined by the knowledge of a fixed size subgraph, the relabeled occurrence. We introduce an extension of that model, the so-called expanding graph relabeling systems (e-GRS's), which allows the generation of sets of graphs by means of component relabeling. We study the generating power of these systems and prove that they enable us to generate any recursively enumerable set of graphs. We first show how the "from left to right" natural orientation of a string-graph, that is a graph representation of a string, can be translated by means of vertex labels in such a way that any local transformation of the string can be simulated by a local relabeling of the string-graph vertices. Using this translation, we show that any phrase-structure string grammar can be simulated by an e-GRS. Finally, we provide a way of encoding graphs as strings and an e-GRS, called the decoder, which can convert any string representation of the encoding of a graph into the graph itself.

Keywords. Graph relabeling systems, Graph grammars, Recursive enumerability.

1 Introduction

The theory of graph rewriting, or graph grammars, is an active field in computer science for more than twenty years (see for instance the survey by Nagl [21]). In a *graph rewriting system* the basic operation consists in a subgraph replacement operation, usually done in three steps : we first locate in the graph to be rewritten an image of the left-hand side of a *rewriting rule*, and delete it. We then put in its place the right-hand side of the rewriting rule and finally "connect" it to the host graph. This third step is clearly the difficult one and is usually carried out by means of some special *embedding mechanism* depending on the considered model.

Various models have been considered in the literature. The replacement operation may concerns single vertices, as in the Node Label Controlled approach [8, 12, 13], single edges, or more generally hyperedges [1, 2, 9, 10, 19, 20], or any kind of connected subgraphs as in [18, 26] or in the well-known algebraic approach of the Berlin school [6, 7]. In that formalism, rewriting steps are defined in terms of double-pushouts (or simple-pushouts in [17, 22]) in a given category.

Graph relabeling systems have been introduced in [3] as a suitable tool for coding and proving sequential or distributed algorithms on graphs or networks. These systems deal with connected labeled graphs (given as a graph G together with a labeling function λ) and satisfy the following requirements:

(i) they do not change the underlying graph but only the labeling of its components (edges and/or vertices), the final labeling being the result of the computation,

[1]With the support of the European Basic Research Action ESPRIT n° 3166 "ASMICS" and the ESPRIT-Basic Research Working Group n° 7183 "COMPUGRAPH II".

(*ii*) they are *local*, that is, each relabeling step changes only a connected subgraph of a fixed size in the underlying graph,

(*iii*) they are *locally generated*, that is, the applicability of a relabeling rule only depends on the *local context* of the relabeled subgraph.

In order to relate the relabeling approach to classical models of graph rewriting systems, we extend these relabeling systems and obtain graph generating devices based on a relabeling operation, on the contrary to the usual replacement operation. The *expanding graph relabeling systems* thus obtained will also satisfy the three previous constraints. In order to keep the principle of "component relabeling", we introduce the notion of an *expanded graph* which can be viewed as an infinite, complete, simple, loopless graph (called the *universal* graph) in which a finite subset of components (vertices and edges) are selected thanks to an adequate labeling. More precisely, we will consider an infinite countable set of vertices V_∞ with associated edge set $E(V_\infty)$ and two labeling functions $\mu_v : V_\infty \longrightarrow \mathcal{L}_v$ and $\mu_e : E(V_\infty) \longrightarrow \mathcal{L}_e$ where $\mathcal{L}_v$ and $\mathcal{L}_e$ are two finite sets of labels such that $\perp \in \mathcal{L}_v \cap \mathcal{L}_e$. $\perp$ is a special symbol used to indicate that the components thus labeled do not have to be considered as part of the (underlying) graph. In order to deal with finite graphs we require both the sets $\{x \in V_\infty / \ \mu_v(x) \neq \perp\}$ and $\{\{x,y\} \in E(V_\infty) / \ \mu_e(\{x,y\}) \neq \perp\}$ to be finite. Note that expanded graphs are simply a way of viewing classical graphs as subgraphs of a universal graph and that we do not really deal with infinite graphs (an analogy can here be made with the "potentially infinite" tape of a Turing machine).

An expanding relabeling rule will then consist in the relabeling of a fixed connected expanded subgraph, that is a subgraph whose underlying graph is connected in the usual way. This leads to the notion of expanding graph relabeling systems, namely *e-GRS*'s. The basic operation is then a relabeling operation which can be done in two steps as follows : we first locate in the graph to be relabeled an image of the left-hand side of a relabeling rule and then relabel it according to the right-hand side. Hence, the third step of the usual graph-replacement approach is bypassed, leading to a formalism without any explicit embedding mechanism, as in the set-theoretic approach of Raoult [23].

Moreover, we naturally obtain systems which are "context-preserving" in the following sense : if a graph G derives a graph G' by the relabeling of a subgraph K of G, then the context of K in G (that is all the components of G that do not belong to K, but also the edges linking a vertex of K and a vertex of $G \setminus K$) is preserved in G'. Under some specific constraints, such a property is also satisfied by some existing models (see section 2 for a more detailed discussion).

In this paper we study the generating power of *e-GRS*'s and show that they enable us to generate any recursively enumerable set of graphs. To our knowledge, only four models of graph grammars [2, 19, 20, 26] have been shown to have a similar power. Our result on relabeling systems seems to indicate that the generating power of a graph rewriting system is not a consequence of its embedding mechanism as suggested by Main and Rozenberg (see discussion in [19]).

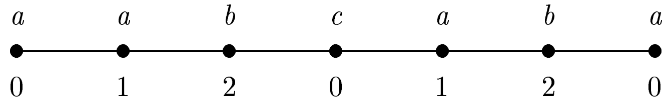
In order to prove that a graph rewriting model has the power of recursive enumerability, the following method is generally used:

(*i*) prove that any phrase-structure string grammar [11] can be simulated by a graph rewriting system,

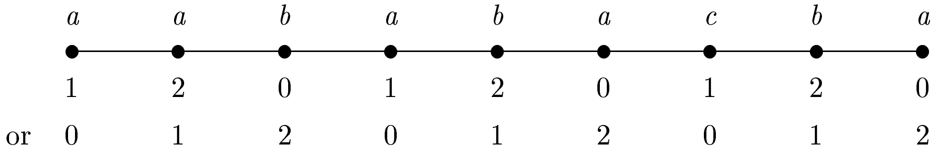
(*ii*) find a linear encoding of graphs as strings such that the decoding process can be handled by a graph rewriting system,

(*iii*) "merge" the two above-defined systems in order to generate any family of graphs whose corresponding set of encodings is recursively enumerable.

Usual representations of strings by graphs make use of the notion of string-graph. Since we deal with undirected labeled graphs, we have to handle the natural orientation of strings (that is from left to right) by means of vertex (or edge) labels. It is folklore that any string-graph can be "oriented" by using the vertex label set $\{0, 1, 2\}$ and considering an edge as directed from an i-labeled vertex to a j-labeled one whenever $j = i + 1 \ (mod\ 3)$. For example, the orientation of the string-graph representation of *aabcaba* may be encoded as follows:



However, such an encoding does not allow an easy treatment of string productions: if we want to replace the substring *bca* by *babac* we must relabel all the vertices of the left (or right) part of the rewritten substring as follows:



In order to overcome that drawback, we provide a new encoding of string orientation using 7 labels, which allows us to handle any string production by changing only a fixed part of the rewritten string (namely the rewritten substring and eventually its immediately left and/or right neighbours).

The paper is organised as follows. In section 2, we introduce and illustrate the notions of expanded graphs and expanding graph relabeling systems. In section 3, we turn to the combinatorial problem of string orientation encoding and show in section 4 how any phrase-structure string grammar can be simulated by an *e-GRS*. In section 5, we provide a linear encoding of graphs as strings and prove in section 6 that the corresponding decoding operation can be handled by an *e-GRS*, which allows us to obtain our main result (section 7). Finally, section 8 outlines some directions for future work.

2 Expanding Graph Relabeling Systems and Graph Languages

Graph relabeling systems [3, 15, 16] have been essentially introduced as a suitable tool for describing local computations on graphs. These systems are essentially "static" in the sense that they only modify the labeling of edges and vertices of the rewritten graph, and not the underlying structure of the graph itself. In this section, we want to extend these systems in order to allow the generation of sets of graphs, as for classical graph grammars. However, our approach will preserve the main characteristics of graph relabeling systems: the rewriting rules will only be relabeling rules, that is they will not modify the "underlying structure" of the rewritten graph. In order to capture the generative concept, we will work on *expanded graphs* which can be viewed as infinite complete graphs in which a finite subset of components (vertices and edges) are selected thanks to an adequate labeling.

In this paper, we will consider simple, loopless, undirected labeled graphs. Note that all the definitions we will use can easily be extended to other types of graphs. For any set V (finite or not) we will denote by $E(V)$ the set $E(V) = \{\{x,y\}/\ x \in V,\ y \in V,\ x \neq y\}$. Let $\mathcal{L} = (\mathcal{L}_v, \mathcal{L}_e)$ be a pair of two finite sets of labels (the vertex and edge labels respectively). A *labeled graph* G is defined as a triple (V, E, λ) where V is a finite set of vertices, $E \subset E(V)$ a set of edges, $\lambda = (\lambda_v, \lambda_e)$ the labeling function with $\lambda_v : V \longrightarrow \mathcal{L}_v$ and $\lambda_e : E \longrightarrow \mathcal{L}_e$.

Suppose now that $\perp \in \mathcal{L}_v \cap \mathcal{L}_e$, where $\perp$ is a special label used to indicate that some components do not have to be considered. An *expanded labeled graph* is a pair $H = (V_\infty, \mu)$ where V_∞ is an infinite countable set of vertices and $\mu = (\mu_v, \mu_e)$ a pair of mappings with $\mu_v : V_\infty \longrightarrow \mathcal{L}_v,\ \mu_e : E(V_\infty) \longrightarrow \mathcal{L}_e$ such that both the sets $\{x \in V_\infty /\ \mu_v(x) \neq \perp\}$ and $\{\{x,y\} \in E(V_\infty)/\ \mu_e(\{x,y\}) \neq \perp\}$ are finite. $E(V_\infty)$ is the set of edges and μ_v (resp. μ_e) the vertex (resp. edge) labeling function. Two expanded labeled graphs $H = (V_\infty, \mu)$ and $H' = (V_\infty, \mu')$ are said to be *isomorphic* if there exists a one-to-one mapping φ over V_∞ such that $\forall\ x \in V_\infty,\ \mu_v(x) = \mu'_v(\varphi(x))$ and $\forall\ x, y \in V_\infty,\ \mu_e(\{x,y\}) = \mu'_e(\{\varphi(x), \varphi(y)\})$.

Remark 1 Every labeled graph $G = (V, E, \lambda)$ with $V \subset V_\infty$ can naturally be viewed as an expanded labeled graph $H = (V_\infty, \mu)$ with (i) if $x \in V$ then $\mu_v(x) = \lambda_v(x)$ else $\mu_v(x) = \perp$ and (ii) if $\{x,y\} \in E$ then $\mu_e(\{x,y\}) = \lambda_e(\{x,y\})$ else $\mu_e(\{x,y\}) = \perp$. Hence, expanded labeled graphs will be simply

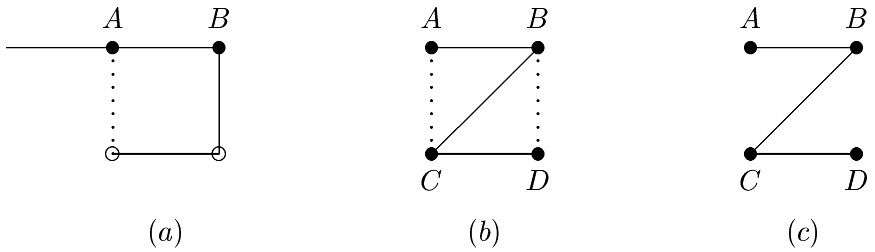


Figure 1: Sample partial graphs.

referred to as *graphs* in the rest of this paper. When we want to explicitly distinguish these two notions, we will speak about *usual* or *expanded* graphs.

Let V be a finite subset of V_∞ and $\nu = (\nu_v, \nu_e)$ be a pair of *partial* mappings $\nu_v : V \longrightarrow \mathcal{L}_v$, $\nu_e : E(V) \longrightarrow \mathcal{L}_e$. The pair $K = (V, \nu)$ is said to be a *partially finitely expanded graph* (or simply *partial graph*). Hence, a partial graph (V, ν) can be viewed as a partial labeling of the complete graph K_n where $n = \#V$.

Note that the labeling of a graph in our sense does not necessarily induce a usual graph: some existing edges, that is edges which are not $\perp$-labeled, may have non-existing, or $\perp$-labeled, end-points. We will say that a graph is a *real* graph if the vertex labeling function is "well-defined" in the following sense :

$$\forall\ \{x,y\} \in E(V_\infty),\ \ \mu_e(\{x,y\}) \neq \perp \implies \mu_v(x) \neq \perp \text{ and } \mu_v(y) \neq \perp.$$

In the same way, a partial graph will be said to be a *real* partial graph if :

$$\forall\ \{x,y\} \in Dom(\nu_e),\ \ \nu_e(\{x,y\}) \neq \perp \implies x, y \in Dom(\nu_v),\ \nu_v(x) \neq \perp \text{ and } \nu_v(y) \neq \perp,$$

where $Dom(\nu_v)$ (resp. $Dom(\nu_e)$) denotes the set of vertices (resp. edges) for which ν_v (resp. ν_e) is defined.

Remark 2 When we have to deal with unlabeled graphs, we will use the label sets $\mathcal{L}_v = \mathcal{L}_e = \{\perp, \varepsilon\}$, where ε stands for the "empty" label.

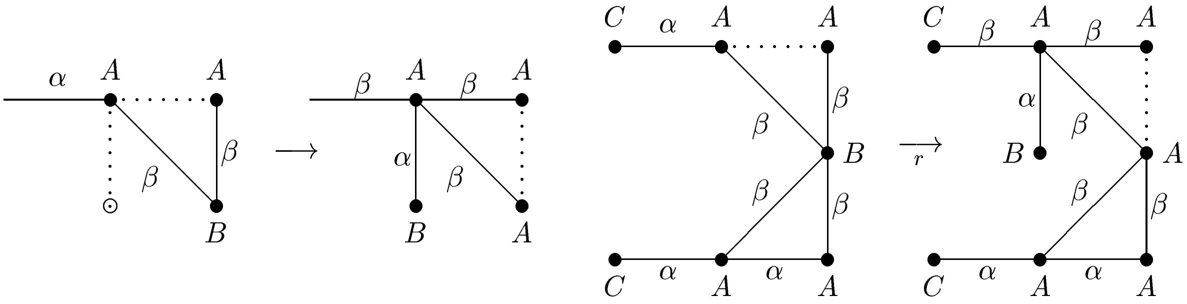
Drawing conventions. When we draw graphs (or partial graphs), we represent all the components which are not $\perp$-labeled and some of the $\perp$-labeled ones when it is necessary (e.g. when the graph is not a real graph). $\perp$-labeled vertices will be drawn as circles, other vertices as full circles, $\perp$-labeled edges as dotted lines and other edges as thin lines. Some edges of partial graphs may appear as "hanging" edges when one (or two) of their end-points do not belong to $Dom(\nu_v)$. ε-labeled components will be drawn as unlabeled ones.

Example 3 Figure 1(a) shows a partial graph which is not a real partial graph : there is one edge with only one end-point and two edges with $\perp$-labeled end-points. Figure 1(b) shows a real partial graph and figure 1(c) the usual graph it corresponds to.

We now extend classical definitions on usual graphs to expanded graphs. Let $H = (V_\infty, \mu)$ be a graph. A vertex x and an edge of the form $\{x, y\}$ are said to be *incident*. A *generalized path*, or simply *path*, is a sequence $(c_1, c_2, \ldots, c_k)$ of not $\perp$-labeled components (that is edges or vertices) such that for any i, $1 \leq i < k$, c_i and c_{i+1} are incident. Hence, such a path is alternatively made of edges and vertices. It may indifferently start or end with an edge or a vertex. We will say that the two components c_1 and c_k are *linked* by this path. A graph, or a partial graph, is said to be *connected* when any two of its not $\perp$-labeled components are linked by a path.

Let $H = (V_\infty, \mu)$ be a graph and $K = (V, \nu)$ be a partial graph. We will say that K is a *subgraph* of H if

$$\begin{array}{rl} (i) & V \subset V_\infty \ \text{ (which implies } E(V) \subset E(V_\infty)), \\ (ii) & \forall\ x \in Dom(\nu_v),\ \nu_v(x) = \mu_v(x), \\ (iii) & \forall\ \{x,y\} \in Dom(\nu_e),\ \nu_e(\{x,y\}) = \mu_e(\{x,y\}). \end{array}$$



(a) The rule r (b) A relabeling step $H \underset{r}{\longrightarrow} H'$

Figure 2: The expanded relabeling mechanism.

Let $K = (V, \nu)$ be a partial graph and θ be an injective mapping from V to V'. The partial graph $\theta(K) = (V', \lambda)$ is defined by

$$\begin{array}{rl}
(i) & V' = \theta(V) = \{\theta(x),\ x \in V\}, \\
(ii) & Dom(\lambda_v) = \theta(Dom(\nu_v)),\ Dom(\lambda_e) = \theta(Dom(\nu_e)), \\
(iii) & \forall\ x \in Dom(\nu_v),\ \lambda_v(\theta(x)) = \nu_v(x), \\
(iv) & \forall\ \{x, y\} \in Dom(\nu_e),\ \lambda_e(\{\theta(x), \theta(y)\}) = \nu_e(\{x, y\}).
\end{array}$$

If $\theta(K)$ is a subgraph of H, we will say that θ is an *occurrence* of K in H. In the following, for any subset A of $E(V)$, we will denote by $\theta(A)$ the set $\{\{\theta(x), \theta(y)\},\ \{x, y\} \in A\}$.

In an expanded graph, any two vertices are linked by an edge (maybe a virtual edge). If we want to rewrite (i.e. relabel) a given graph in a "local" way, we must restrict the structure of the left-hand sides of the rules in order to avoid the relabeling of "distant" vertices (i.e. joined by a $\perp$-labeled edge) by a unique rule.

An *expanding relabeling rule*, or simply *rule*, is a triple $r = (V^r, \nu^r, \nu'^r)$, also denoted $(V^r, \nu^r) \longrightarrow (V^r, \nu'^r)$, such that (V^r, ν^r) is a connected non-empty partial graph, (V^r, ν'^r) is a partial graph, $Dom(\nu_v^r) = Dom(\nu_v'^r)$ and $Dom(\nu_e^r) = Dom(\nu_e'^r)$. Let r be a rule; the relabeling relation $\underset{r}{\longrightarrow}$ is defined in the following way: let $H = (V_\infty, \mu)$ and $H' = (V_\infty, \mu')$ be two graphs, $H \underset{r}{\longrightarrow} H'$ if there exists an occurrence θ of (V^r, ν^r) in H such that :

$$\begin{array}{rl}
(i) & \theta \text{ is an occurrence of } (V^r, \nu'^r) \text{ in } H', \\
(ii) & \forall\ x \in V_\infty \setminus \theta(Dom(\nu_v^r)),\ \mu_v(x) = \mu_v'(x), \\
(iii) & \forall\ \{x, y\} \in E(V_\infty) \setminus \theta(Dom(\nu_e^r)),\ \mu_e(\{x, y\}) = \mu_e'(\{x, y\}), \\
(iv) & \forall\ x \in Dom(\nu_v^r),\ (\nu_v^r(x) = \perp \text{ and } \forall\ \{x, y\} \in Dom(\nu_e^r),\ \nu_e^r(\{x, y\}) = \perp) \\
& \implies (\forall\ z \in V_\infty,\ \mu_e(\{x, z\}) = \perp).
\end{array}$$

Note that condition *(iv)* will ensure that the creation of a new vertex by a rule will involve a "free" vertex, that is a $\perp$-labeled vertex which is not linked to any "hanging" edge. Note that depending on the choice of that free vertex, we may obtain several graphs which are all isomorphic. In the following, such an occurrence of (V^r, ν^r) will be called an occurrence of r.

Example 4 Figure 2(a) shows a sample rule r and figure 2(b) a sample application of r. Note that there is no occurrence of r in the right bottom part of H since the two corresponding A-labeled vertices are joined by an edge which is not $\perp$-labeled.

An *expanding graph relabeling system* (or *e-GRS*) is a pair $\mathcal{R} = (L, P)$ where L is a pair (L_v, L_e) of finite label sets (respectively of vertex and edge labels) containing $\perp$ and P a finite set of rules. The relabeling relation $\xrightarrow[\mathcal{R}]{}$ is defined by: $H \xrightarrow[\mathcal{R}]{} H'$ if and only if there exists a rule $r \in P$ such that $H \xrightarrow[r]{} H'$. Its transitive closure will be denoted by $\xrightarrow[\mathcal{R}]{*}$.

Starting from a graph Z (the *axiom*), we are now able to generate some (finite or infinite) sets of graphs. Let $T = (T_v, T_e)$ be a pair of *terminal* label sets. The *terminal language* of $\mathcal{R}$ is defined as :

$$\mathbf{L}_T(\mathcal{R}, Z) = \{\ H\ /\ H \text{ is a real graph with labels in } T,\ Z \xrightarrow[\mathcal{R}]{*} H\}$$

For any *e-GRS* $\mathcal{R} = (L, P)$ we will simply denote by $\mathbf{L}(\mathcal{R}, Z)$ the language $\mathbf{L}_L(\mathcal{R}, Z)$ obtained by taken the whole set L as terminal label set.

Expanding graph relabeling systems can be understood as classical graph rewriting systems which are "context-preserving" in the following sense: if $H = (V_\infty, \mu)$ and $H' = (V_\infty, \mu')$ are two graphs such that $H \xrightarrow[r]{} H'$ by an application of r to an occurrence θ, then we have :

$$\begin{array}{ll} (i) & \forall\ x \in V_\infty \setminus \theta(Dom(\nu_v^r)),\ \mu_v(x) = \mu'_v(x), \\ (ii) & \forall\ \{x, y\} \in E(V_\infty) \setminus \theta(Dom(\nu_e^r)),\ \mu_e(\{x, y\}) = \mu'_e(\{x, y\}). \end{array}$$

Some of the existing graph rewriting models do not satisfy this condition: in the Node-Label-Controlled approach [8, 12, 13] or in the Pfaltz and Rosenfeld approach [18], edges incident to nodes of the "context" of the rewritten nodes can be created or removed, according to some connection relation. In the Hyperedge Replacement approach [1, 10] or in the algebraic approach [6] this condition is not satisfied as soon as we are allowed to identify vertices in the context-graph. By using a priority mechanism [3], we show in [24], how such systems can be simulated by means of relabeling systems : every rewriting step is encoded by a sequential application of relabeling steps, each one being context-preserving.

The relabeling approach we have introduced is very similar to the double-pushout construction in the Algebraic Approach where the left-hand sides are connected graphs, the interface graph is discrete and the morphisms are injective. One difference is that the context preservation in the algebraic approach is ensured thanks to some *gluing condition* (see [6]) which prevents a rule from being applied when some "hanging" edges may appear. Another difference is that in our general model we may use as left-hand sides some partial graphs which are not real graphs (as an edge without end-points for instance). However, since we will not need to use these possibilities in proving our main result, we obtain as a consequence that this particular case of the algebraic approach also has the power of recursive enumerability.

Our model may also be related to the model introduced by Uesu [26] : in that model, context preservation is ensured by means of some applicability condition (using the notion of *graph partition*) which is equivalent to the gluing condition in the algebraic approach. But the main difference between our model and Uesu's one is that our relabeling rules always have connected left-hand sides. In particular, the set of rewriting rules used by Uesu in the proof of the recursive enumerability power of his model contains some rules with non-connected left-hand sides. However, the set of relabeling rules we will use in the following can easily be expressed within Uesu's approach. We thus obtain a new proof of Uesu's result by means of rewriting rules with connected left-hand sides.

We now illustrate the notions of *e-GRS* by giving systems which generate the set of all unlabeled trees and the set of all unlabeled graphs. Since for any rule the left- and right-hand sides have the same underlying graph, the correspondance between left- and right-components is established according to their graphical position.

Example 5 *(unlabeled trees)* Let $\mathcal{R}_1$ be the *e-GRS* defined by $L_v = L_e = \{\varepsilon, \perp\}$ and $P_1 = \{r_1\}$ where r_1 is given as:

r_1 : ●·······⊙ $\longrightarrow$ ●———●

Rule r_1 allows us to attach to any existing vertex a (new) one-degree vertex. Hence, if Z stands for the one vertex (unlabeled) graph, $\mathbf{L}(\mathcal{R}_1, Z)$ is exactly the set of unlabeled trees.

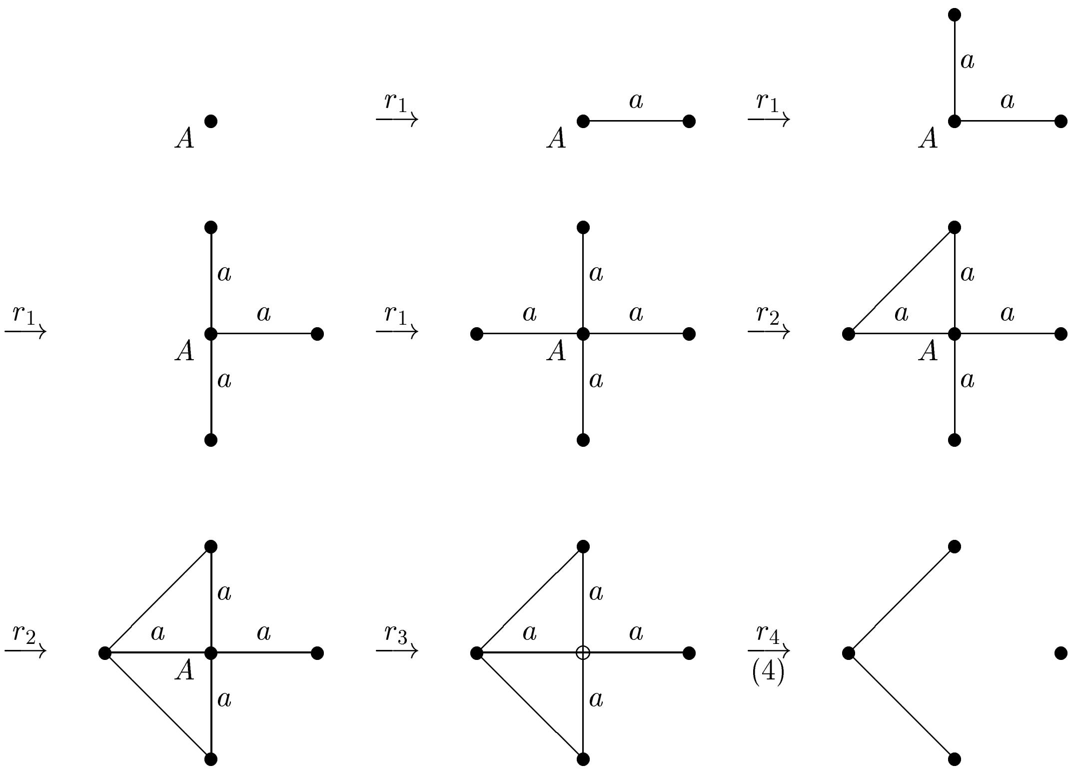
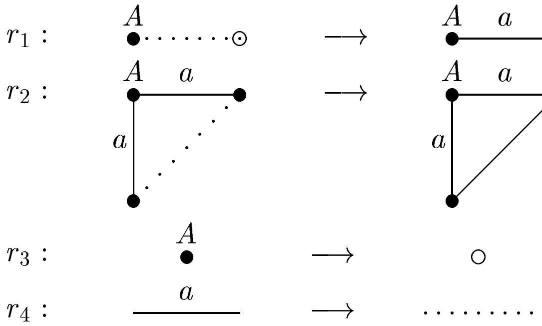


Figure 3: A sample derivation sequence in $\mathcal{R}_2$

Example 6 *(the set of all unlabeled graphs)* In order to generate the set of all graphs, we will use a distinguished vertex (the "administrator" of the generation) which will be linked by a special edge (with label a) to all the newly created vertices (rule r_1). This administrator will then be able to link together any two vertices of the graph thus constructed (rule r_2). At the end of that construction, the administrator vertex and all special edges will be deleted by adequate erasing rules (r_3 and r_4). More formally, let $\mathcal{R}_2$ be the *e-GRS* defined by $L_v = \{A, \varepsilon, \perp\}$, $L_e = \{a, \varepsilon, \perp\}$ and $P_2 = \{r_1, r_2, r_3, r_4\}$ where the rules are given as:



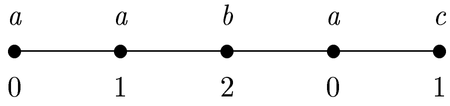
If Z_A stands for the graph with one A-labeled vertex, the *e-GRS* $\mathcal{R}_2$ thus obtained is such that $\mathbf{L}_T(\mathcal{R}_2, Z_A)$, where $T = (\{\varepsilon, \perp\}, \{\varepsilon, \perp\})$, is the set of all unlabeled graphs. Figure 3 shows a sample derivation sequence. Note that all special edges can be deleted as soon as their corresponding vertex is no longer used in the generation process.

Other examples of *e-GRS*'s generating different families of graphs can be found in [25].

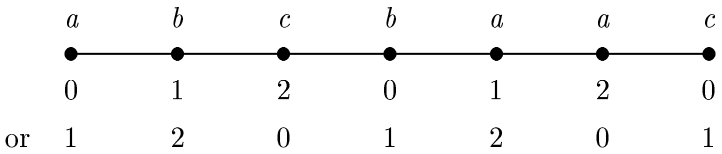
3 Encoding the orientation of string-graphs

In this section, we turn to a combinatorial problem concerning the encoding of the natural orientation (from left to right) of string-graphs by using vertex labels in such a way that any "substring replacement" can be realised by a *local* relabeling of the string-graph. We show that such an encoding can be obtained by using a set of 7 vertex labels. This encoding will allow us to simulate any phrase-structure string grammar by an e-GRS as shown in the next section.

When we represent words (or strings) as undirected string-graphs, it is necessary to use additional information allowing us to retrieve their natural "from left to right" orientation. This can easily be done by using vertex labels taken in the set $\{0, 1, 2\}$ according to the following rule: an edge $\{x, y\}$ is directed from x to y iff $\lambda(y) = \lambda(x) + 1 \ (mod\ 3)$. Hence, any string on an alphabet X can be encoded by a string-graph whose vertices are labeled in $X \times \{0, 1, 2\}$. For example, the string $aabac$ can be encoded as follows (the orientation component is written separately for clarity):



If we want to simulate a phrase-structure string grammar by a graph rewriting system, we must be able to handle applications of string productions, that is substring replacements, by a local modification of the corresponding string-graph. For instance, an application of the string production $ab \longrightarrow bcba$ should change the string-graph encoding $aabac$ into a string-graph encoding $abcbaac$. By using the set $\{0, 1, 2\}$ as orientation components, we have to relabel all the vertices on the left (or right) side of the rewritten part, thus having:



As we want to "locally" relabel the corresponding string-graphs, we will have to use a more elaborate encoding for string orientation.

Let us now introduce more formally the notion of orientation encoding.

Definition 7 Let $G = (V, A)$ and $H = (W, B)$ be two loopless antisymetric directed graphs. We will say that H is an *encoder* of the orientation of G if there exists an *encoding mapping* ξ from V to W such that :

$$\forall\ (x, y) \in A,\ \ (\xi(x), \xi(y)) \in B.$$

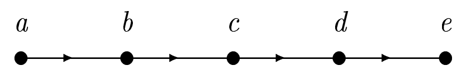
The orientation of G can then be encoded by using W as set of vertex labels and labeling any vertex x in V by $\xi(x)$.

For any string u, we will denote by $D(u)$ its associated directed vertex-labeled string-graph representation. If $H = (W, B)$ is an encoder for the orientation of $D(u)$ and ξ an encoding mapping, we will denote by $U_H(u, \xi)$ the encoded version of $D(u)$ given by ξ.

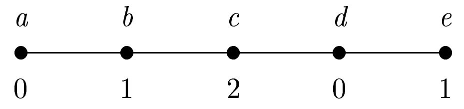
Example 8 As we have seen before, the directed cycle C_3 (see figure 4(c)) is an encoder of any string-graph. For example, the string-graph $D(abcde)$ (see figure 4(a)) can be oriented as shown in figure 4(b), thanks to the mapping ξ given by $\xi(a) = \xi(d) = 0,\ \xi(b) = \xi(e) = 1,\ \xi(c) = 2$. Moreover, it is not difficult to check that C_3 is the smallest graph which can encode any string-graph.

In order to handle an arbitrary substring replacement in a string-graph by a local relabeling of its vertices, we will use the directed tournament F_7, associated with the Fano plane of order 7. The Fano plane (see figure 5) is given as 7 vertices and 7 lines of three vertices each (the "internal" line is drawn as a triangle) such that any two lines have a unique common vertex. If we number the vertices from 0 to 6, it is then possible to number the lines from 0 to 6 in such a way that:

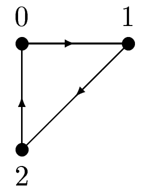
$$\forall\ i, j \in \{0, 1, \ldots, 6\},\ \ i \notin L_i\ \text{ and }\ j \in L_i \Rightarrow i \notin L_j$$



(a) the string-graph $D(abcde)$

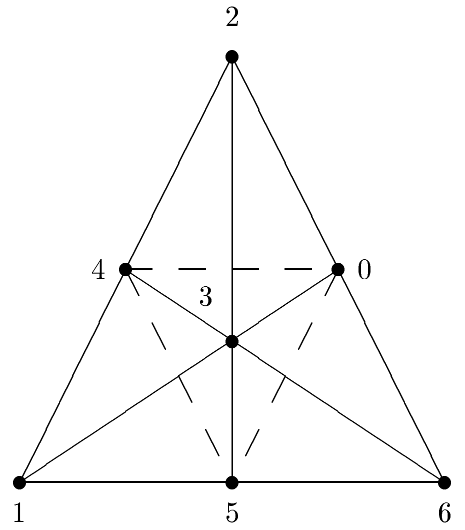


(b) the encoded string-graph $U_{C_3}(abcde, \xi)$



(c) the encoder C_3

Figure 4: Orientation of string-graphs with C_3.



$L_0 = \{1, 2, 4\}$ $L_4 = \{5, 6, 1\}$

$L_1 = \{2, 3, 5\}$ $L_5 = \{6, 0, 2\}$

$L_2 = \{3, 4, 6\}$ $L_6 = \{0, 1, 3\}$

$L_3 = \{4, 5, 0\}$

Figure 5: Numbering of the Fano plane.

This numbering is obtained by setting $L_i = \{i+1, i+2, i+4\}$ (these values are taken modulo 7). The associated directed graph $F_7 = (V_7, E_7)$ is then defined by $V_7 = \{0, 1, \dots, 6\}$ and $\forall\, i, j \in V_7,\ (i, j) \in E_7$ iff $j \in L_i$. Hence, each line L_i corresponds to the set of successors of vertex i.

It is not difficult to check that the graph F_7 satisfies the following property :

$$(P)\ \ \forall\, x, y \in V_7,\ x \neq y,\ \exists\, z \in V_7,\ s.t.\ \ (x, z) \in E_7 \text{ and } (z, y) \in E_7$$

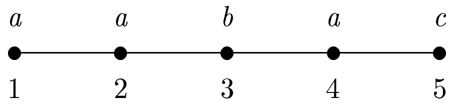
For instance, if we consider the vertices 0 and 1, the vertex 4 is such that $(0, 4)$ and $(4, 1)$ belong to E_7, the vertex 3 is such that $(1, 3)$ and $(3, 0)$ belong to E_7. This property will ensure that the encoder F_7 can be efficiently used as an encoder, as shown by the following proposition :

Proposition 9 *Let $\alpha \longrightarrow \beta$, be a production, $u = u_1 \alpha u_2$ and ξ be an encoding mapping of $D(u)$ on F_7. Then, there exists an encoding mapping ξ' of $D(u_1 \beta u_2)$ on F_7 such that :*

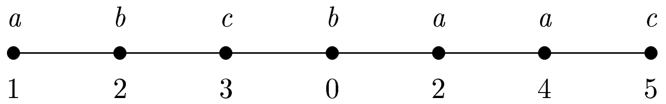
$$\begin{array}{ll} (i) & \textit{if } u_1 = u_1' x_1 \textit{ then } \forall\, x \in u_1',\ \xi'(x) = \xi(x), \\ (ii) & \textit{if } u_2 = x_2 u_2' \textit{ then } \forall\, x \in u_2',\ \xi'(x) = \xi(x). \end{array}$$

Proof. Note first that if $|u_1| = 0$ (resp. $|u_2| = 0$) one can always find a mapping ξ' satisfying the conditions : the mapping ξ on u_2 (resp. on u_1) can be extended on β since any vertex in F_7 has a predecessor (resp. a successor). This is also the case when $u_1 = x_1$ or $u_2 = x_2$. Suppose now that $u_1 = u_1'' y_1 x_1$, $u_2 = x_2 y_2 u_2''$. Let $\alpha = \alpha_1 \alpha_2 \ldots \alpha_m$ and $\beta = \beta_1 \beta_2 \ldots \beta_n$. If $m = n$, the result is obvious since we can simply take $\xi' = \xi$. Suppose now that $n = m + 1$. We first take $\xi'(\beta_i) = \xi(\alpha_i)$ for any i, $1 \leq i \leq m$. Since $u = u_1 \alpha x_2 u_2'$, we necessarily have $\xi(\alpha_m) \neq \xi(x_2)$ and property (P) tells us that there exists an adequate label for β_n. In this case, the mapping ξ' is identical to ξ on u_1 and u_2. If $n = m+k$, $k > 1$, we simply iterate k times the previous construction and the result follows. Suppose now that $n = 0$. If $(\xi(x_1), \xi(x_2))$ is an arc in F_7 we can simply take ξ' identical to ξ on u_1 and u_2. Otherwise, since F_7 is antisymetric, we cannot have at the same time $\xi(y_1) = \xi(x_2)$ and $\xi(x_1) = \xi(y_2)$. If $\xi(y_1) \neq \xi(x_2)$ (resp. $\xi(x_1) \neq \xi(y_2)$), the property (P) tells us that there exists an adequate label for x_1 (resp. x_2) and the result follows. Finally, productions such that $0 < n < m$ can be handled by first erasing α and then inserting β as shown in the above constructions. □

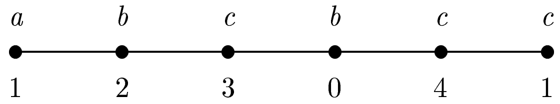
Example 10 Let $U(aabac, \xi)$ be the following graph:



By applying the string production $ab \longrightarrow bcba$, we obtain the graph $U(abcbaac, \xi')$ defined as follows:



By applying now the string production $aa \longrightarrow c$, we obtain the graph $U(abcbcc, \xi')$ defined as follows:

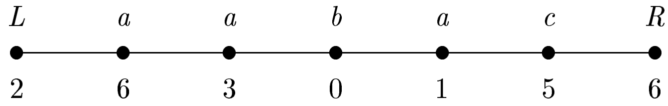


Hence, any string production can be handled in a local way by using a set of 7 orientation labels. The problem of encoding graph orientation by means of vertex labels has also been considered in [5] for other families of graphs but in a static way (that is with no "evolution" of the graph thus encoded).

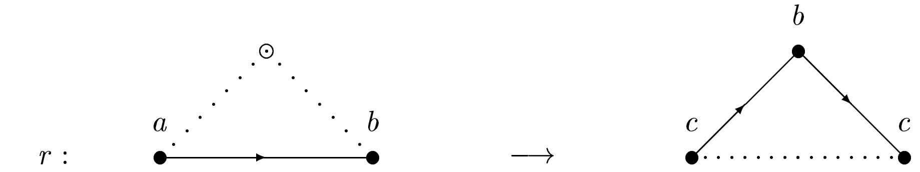
4 e-GRS's and Phrase-Structure String Grammars

A phrase-structure string grammar [11] is given as a 4-tuple $G =< N, T, P, Z >$ where N is a finite set of non-terminal symbols, T a finite set of terminal symbols such that $N \cap T = \emptyset$, $Z \in N$ the axiom symbol and P a finite set of string productions $p : \alpha \longrightarrow \beta$ with $\alpha \in (T \cup N)^+ \setminus T^+$ and $\beta \in (T \cup N)^*$. Let u and v be two strings over $T \cup N$; we say that u derives v, denoted $u \stackrel{*}{\longrightarrow} v$, if there exists a sequence of strings $w_1, w_2, \ldots, w_n$ such that $w_1 = u,\ w_n = v$ and $\forall\ i,\ 1 \leq i < n,\ w_i = w_i' \alpha_i w_i'',\ w_{i+1} = w_i' \beta_i w_i''$ with $\alpha_i \longrightarrow \beta_i$ is a production in P. As usual, we will say that a string production p is *increasing* when $|\beta| \geq |\alpha|$ and *decreasing* otherwise. A decreasing string production with $|\beta| = 0$ will be said to be *erasing*. The *string language* generated by G is defined as $L(G) = \{w \in T^* \ / \ Z \stackrel{*}{\longrightarrow} w\}$. Such a set of strings is said to be *recursively enumerable*.

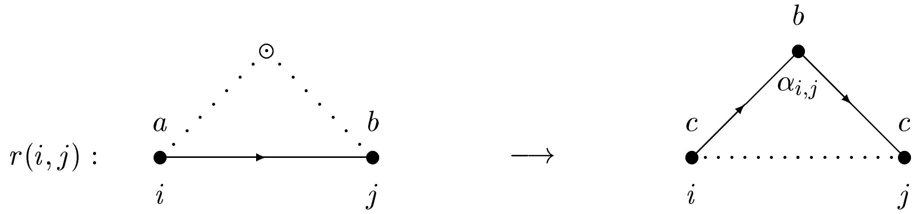
In this section, we will show how any phrase-structure string grammar can be simulated by an *e-GRS*. We have previously seen that the "from left to right" orientation of strings can be captured by using special vertex labels, and have shown that any string production $\alpha \longrightarrow \beta$ can be locally handled. Some special cases require particular attention, namely when the substring to be replaced is located at the beginning or at the end of the rewritten string. Since our graph relabeling rules cannot detect whether a given vertex is an "end-point" or not, we will use a slightly different representation of strings as undirected string-graphs, by adding two special end-point vertices, respectively labeled with L (for Left) and R (for Right). Hence, the string *aabac* will be encoded as (for example):



If H stands for a string-graph of the above form, we will denote by $word(H)$ the string it encodes. Using the results of the previous section, we will not indicate the orientation labels in the relabeling rules but simply draw them with directed edges. Hence, each relabeling rule will in fact be a "meta-rule" corresponding to a (finite) set of standard relabeling rules. For example, the following (meta-) rule:



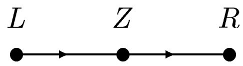
corresponds to the set of rules $\{r(i,j),\ (i,j) \in E_7\}$ defined as:



where $\alpha_{i,j}$ stands for one chosen vertex of V_7 such that $(i, \alpha_{i,j}) \in E_7$ and $(\alpha_{i,j}, j) \in E_7$ (see property (P) in section 3).

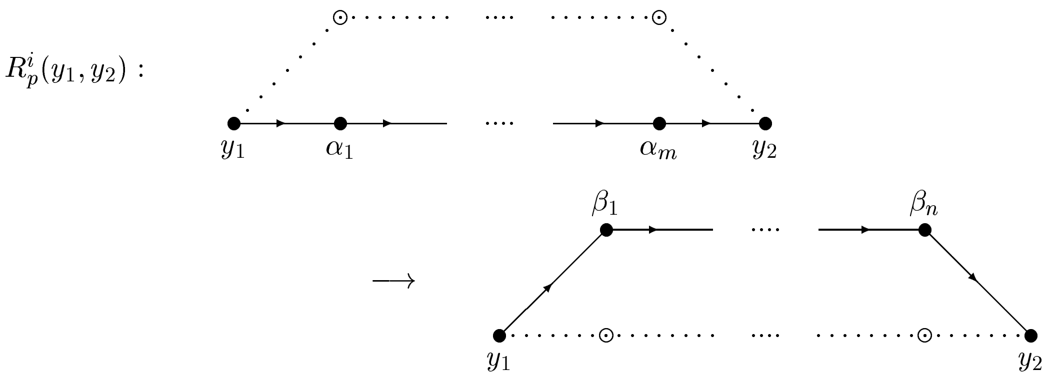
We can now state the main result of this section:

Theorem 11 *For any phrase-structure string grammar $G =< N, T, P, Z >$, there exists an e-GRS $\mathcal{R}_G$ and a set T' of terminal labels such that $L(G) = \{word(H),\ H \in \mathbf{L}_{T'}(\mathcal{R}_G, Z_s)\}$, where Z_s stands for the string-graph encoding Z, that is:*



Proof. Let G be any phrase-structure string grammar. The *e-GRS* $\mathcal{R}_G = (L, P)$ will then be defined as $L_v = (T \cup N \cup \{L, R\}) \times V_7$, $L_e = \{\perp, \varepsilon\}$ and P is obtained by associating with each string production p a set of relabeling rules in the following way:

Case 1: with any increasing string production $\alpha_1 \ldots \alpha_m \longrightarrow \beta_1 \ldots \beta_n$ $(n \geq m)$ we associate the set of rules $\{R_p^i(y_1, y_2),\ y_1, y_2 \in T \cup N \cup \{L, R\}\}$ defined as:

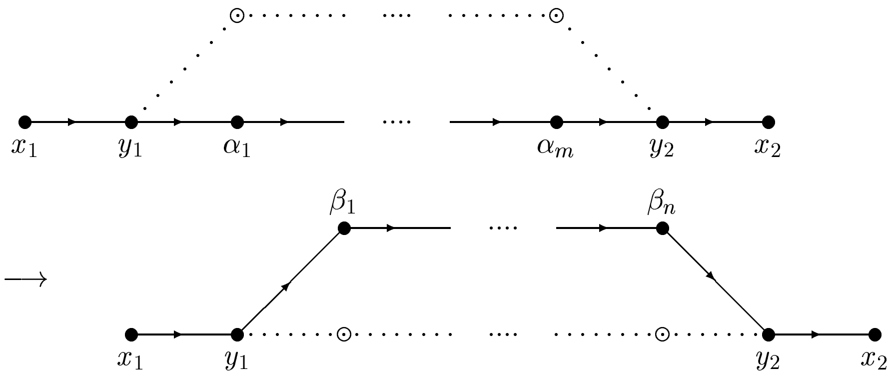


Recall that our encoding of strings as undirected string-graphs ensures that any "substring" α has a left (resp. right) neighbour y_1 (resp. y_2), thanks to the two additional end-point vertices.

Case 2: with any decreasing non-erasing string production $\alpha_1 \ldots \alpha_m \longrightarrow \beta_1 \ldots \beta_n$ $(0 < n < m)$ we associate

(i) the set of rules $\{R_p^{d_1}(y_1, y_2, x_1, x_2),\ y_1, y_2 \in T \cup N,\ x_1, x_2 \in T \cup N \cup \{L, R\}\}$ defined as:

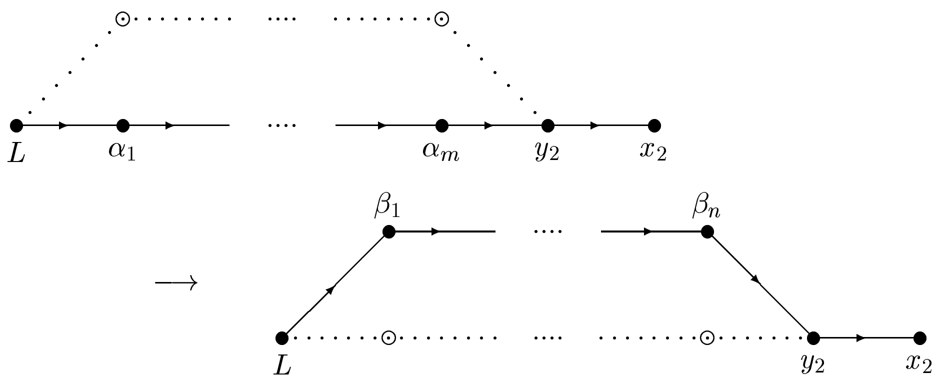
$R_p^{d_1}(y_1, y_2, x_1, x_2)$:



These rules will be used whenever the substring α to be replaced is neither the leftmost nor the rightmost substring of the rewritten string.

(ii) the set of rules $\{R_p^{d_2}(y_2, x_2),\ y_2 \in T \cup N,\ x_2 \in T \cup N \cup \{L, R\}\}$ defined as:

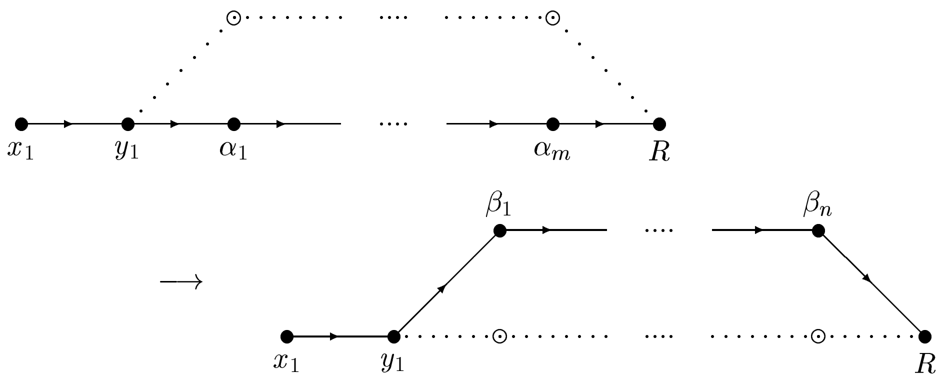
$R_p^{d_2}(y_2, x_2)$:



These rules will be used whenever the substring α to be replaced is the leftmost substring of the rewritten string.

(iii) the set of rules $\{R_p^{d_3}(y_1, x_1),\ y_1 \in T \cup N,\ x_1 \in T \cup N \cup \{L, R\}\}$ defined as:

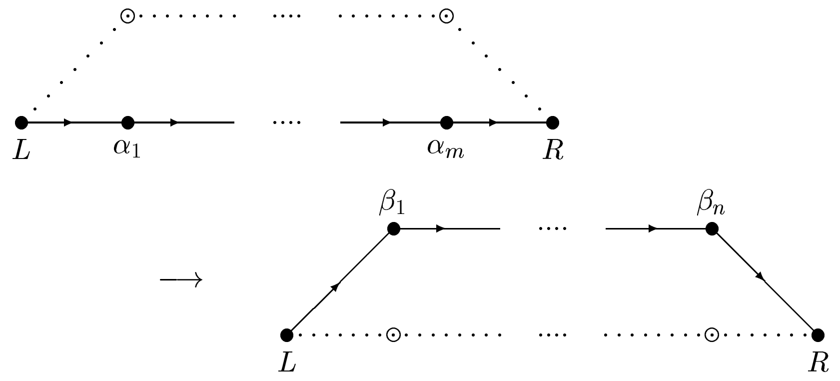
$R_p^{d_3}(y_1, x_1)$:



These rules will be used whenever the substring α to be replaced is the rightmost substring of the rewritten string.

(iv) the rule $R_p^{d_4}$ defined as:

$R_p^{d_4}$:

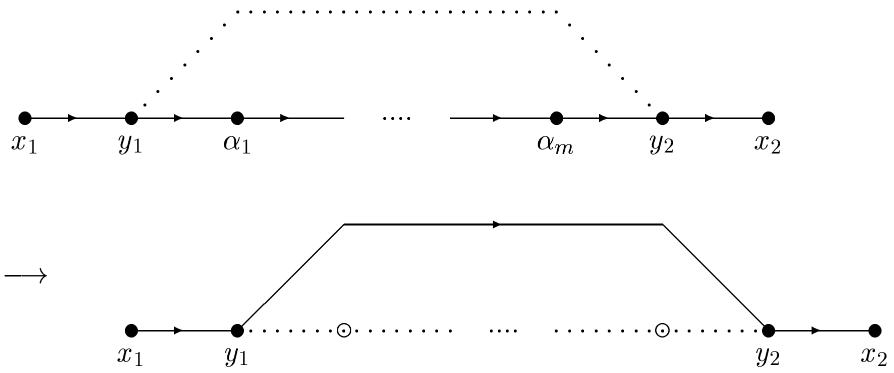


These rules will be used whenever the substring α to be replaced is equal to the full rewritten string.

Case 3: with any erasing string production $\alpha_1 \dots \alpha_m \longrightarrow \varepsilon$ $(0 < m)$ we associate

(i) the set of rules $\{R_p^{e_1}(y_1, y_2, x_1, x_2),\ y_1, y_2 \in T \cup N,\ x_1, x_2 \in T \cup N \cup \{L, R\}\}$ defined as:

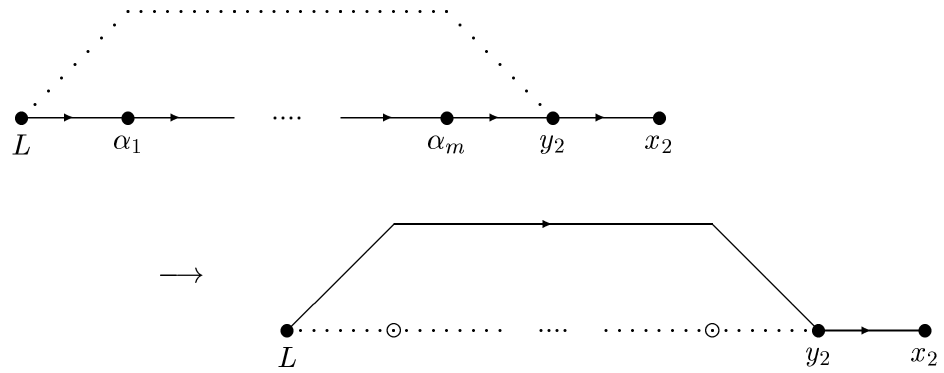
$R_p^{e_1}(y_1, y_2, x_1, x_2)$:



These rules will be used whenever the substring α to be replaced is neither the leftmost nor the rightmost substring of the rewritten string.

(ii) the set of rules $\{R_p^{e_2}(y_2, x_2),\ y_2 \in T \cup N,\ x_2 \in T \cup N \cup \{L, R\}\}$ defined as:

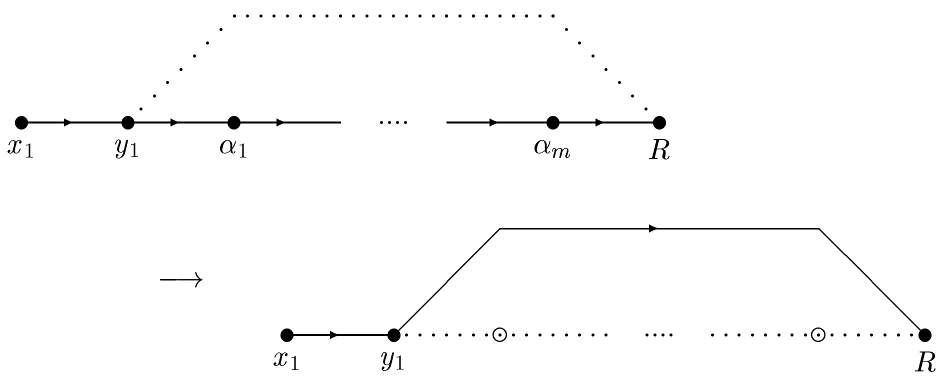
$R_p^{e_2}(y_2, x_2)$:



These rules will be used whenever the substring α to be replaced is the leftmost substring of the rewritten string.

(iii) the set of rules $\{R_p^{e_3}(y_1, x_1),\ y_1 \in T \cup N,\ x_1 \in T \cup N \cup \{L, R\}\}$ defined as:

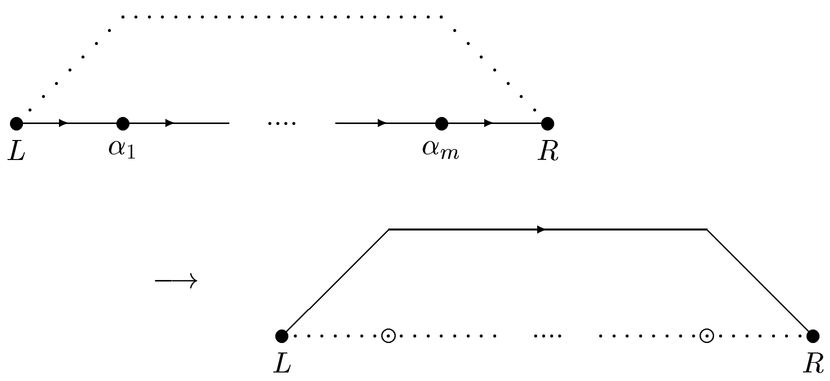
$R_p^{e3}(y_1, x_1)$:



These rules will be used whenever the substring α to be replaced is the rightmost substring of the rewritten string.

(iv) the rule R_p^{e4} defined as:

R_p^{e4} :



These rules will be used whenever the substring α to be replaced is equal to the full rewritten string.

Note that all of the above rules have been constructed in such a way that the orientation label of the leftmost and righmost vertex (excepted if they are L- or R-labeled) of the left-hand side has not to be modified (proposition 9). Any string production $p : \alpha \longrightarrow \beta$ applicable to a given string $u\alpha v$ can always be simulated by an application of a unique relabeling rule on the string-graph encoding $u\alpha v$. Hence, to any derivation sequence of length n in G there is a corresponding derivation sequence of the same length in $\mathcal{R}_G$. It is not difficult to see that the converse also holds and that the *e-GRS* $\mathcal{R}_G$ thus constructed is such that $L(G) = \{word(H),\ H \in \mathbf{L}_{T'}(\mathcal{R}_G, Z_s)\}$ where T' is the set of terminal labels defined as $T' = (\ \{\perp\} \cup (\{L, R\} \cup T) \times V_7\ ,\ \{\perp, \varepsilon\}\ )$. $\square$

5 A linear encoding of undirected graphs

In this section we introduce a way of encoding graphs as strings, inspired by the one used in [19], and will show in the next section how the corresponding decoding operation can be handled by an *e-GRS*. For simplicity we will only present the case of unlabeled graphs, the general case being an easy generalisation of it (see remark 14).

In order to encode an undirected unlabeled graph G as a string, we first assume that a linear ordering of its vertices is given. The encoding $\gamma(G)$ of G will then be given as a string over the alphabet $I = \{a, l, r, c, e\}$ and can be viewed as a sequence of *instructions* allowing a "step by step" construction of G. These instructions will refer to a sequence of vertices, called the *vertex sequence* (the vertices which have already been created) having one distinguished vertex, called the *current vertex*. More precisely, these instructions work as follows:

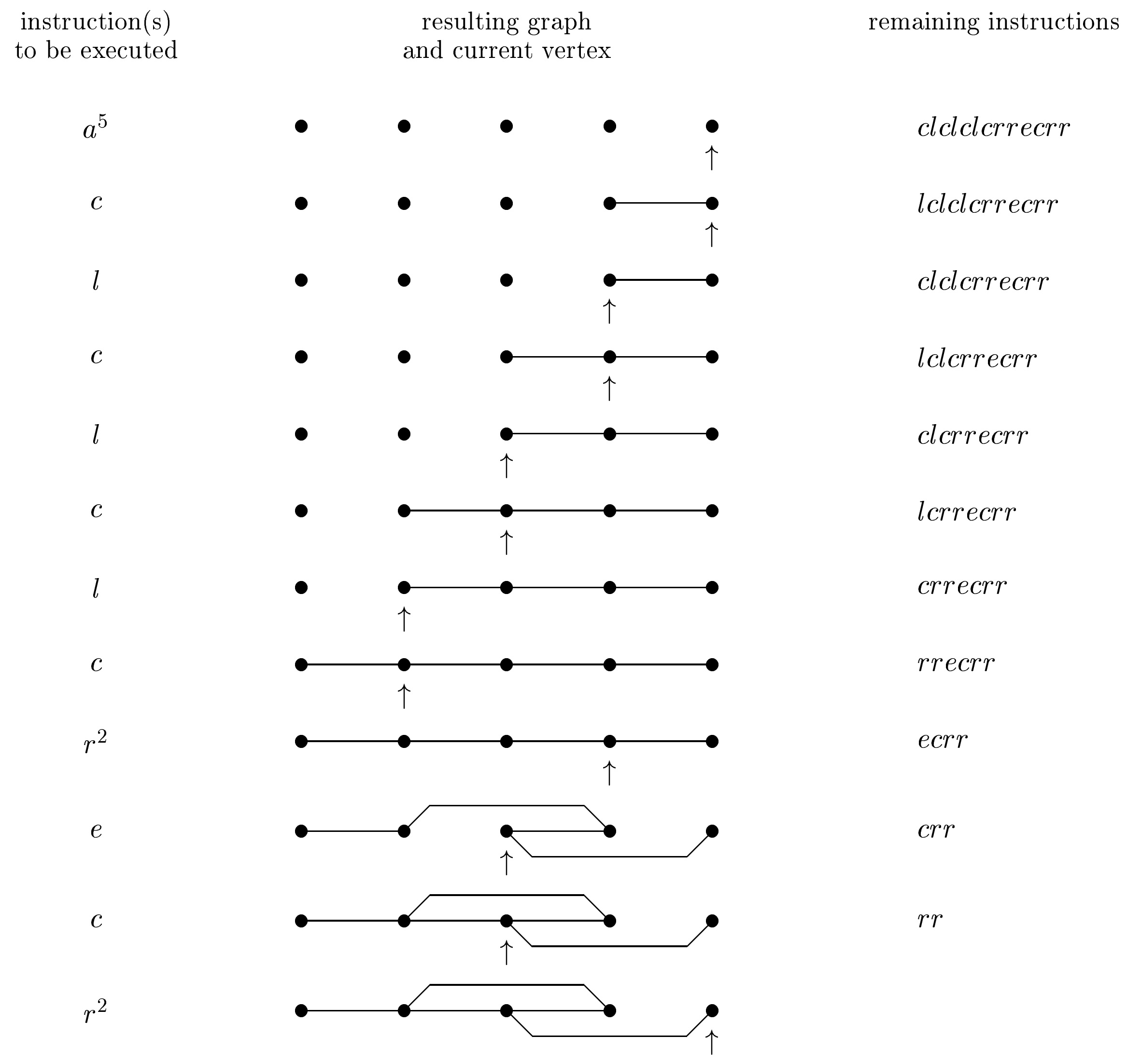
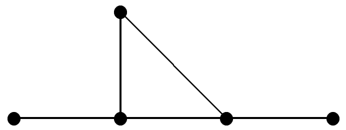


Figure 6: Encoding an unlabeled graph as a string.

a : add a new vertex to the sequence and this new vertex becomes current,
l : the new current vertex is now the left neighbour of the old one,
r : the new current vertex is now the right neighbour of the old one,
c : connect the current vertex to its left neighbour by an unlabeled edge,
e : exchange the current vertex and its left neighbour in the sequence.

Without loss of generality, we can assume that the string $\gamma(G)$ is such that $\gamma(G) \in a^*\{l, r, c, e\}^*$ (we first create all the vertices of G) and that at the end of the computation the current vertex is the rightmost vertex in the sequence. Note that different encoding functions may yield different encodings for the same graph but we do not need the unicity of the encoding.

Example 12 The graph G at the right can be encoded by the string $\gamma(G) = aaaaaclclclcrrecrr$ as shown in Figure 6 (the current vertex is marked as $\uparrow$).



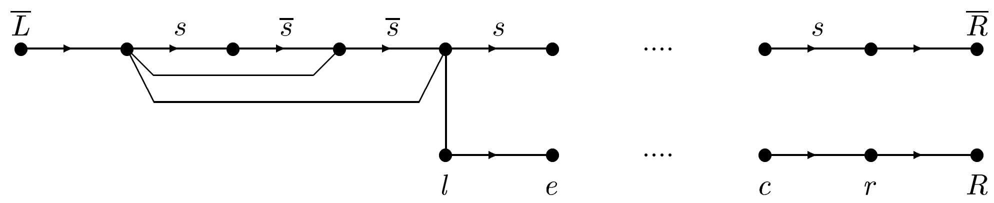
Any set $\mathcal{S}$ of unlabeled undirected graphs can be associated with the set of strings $St(\mathcal{S})$ defined as $St(\mathcal{S}) = \{\gamma(G) \ / \ G \in \mathcal{S}\}$. We will say that the set $\mathcal{S}$ is *recursively enumerable* (see section 7) when its associated set of strings $St(\mathcal{S})$ is recursively enumerable. Note that our definition of recursive enumerability of a set of graphs makes reference to the special encoding γ we have defined. However, any other method allowing us to encode graphs as strings leads to the same notion of recursive enumerability since it is always possible to construct a Turing machine translating one encoding into any other one.

6 Decoding by means of an e-GRS

In this section, we will prove that we can construct an *e-GRS* which will be able to produce any graph when starting from the string-graph representation of its encoding as defined in the previous section. This special *e-GRS* uses relabeling rules which have been designed in order to simulate any instruction of an encoding as shown before. More formally, we obtain:

Proposition 13 *There exists an e-GRS $\mathcal{D}$ (the decoder) such that for any undirected unlabeled graph G, if Z_G stands for the string-graph representation of $\gamma(G)$ then $\mathbf{L}_T(\mathcal{D}, Z_G) = \{G\}$, where $T = (\{\varepsilon, \perp\}, \{\varepsilon, \perp\})$.*

Proof. We start from an axiom Z_G which is a string-graph representation of the string $\gamma(G)$ and whose orientation is encoded by F_7 (see section 3). The basic idea is to provide relabeling rules which will be able to execute (from left to right) the instructions given by $\gamma(G)$ in a way very similar to the one illustrated in Figure 6. During the decoding process, the sequence of generated vertices will be represented as a string-graph (the *sequence string*) whose edges are labeled s or $\overline{s}$ and whose vertices are unlabeled (except an adequate orientation label described below) with two special end-point vertices respectively labeled by $\overline{L}$ and $\overline{R}$. The string-graph corresponding to the remaining instructions, called the *instructions string*, will be attached to the sequence string by an edge linking the current vertex and the next instruction to be executed. Hence, each intermediate graph in a derivation sequence will be of the following form:



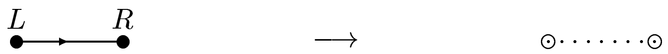
Labels s and $\overline{s}$ are used in order to distinguish the edges of the sequence string (linking any two consecutive vertices) from the edges which have been created between two non consecutive vertices of that sequence. Label s (resp. $\overline{s}$) is used for a sequence edge which has to be (resp. does not have to be) preserved in the final generated graph. Hence, the above configuration corresponds in fact to the following (intermediate) generated graph:

Recall that the creation of the vertices is always made first and that each new vertex must be added to the right end of the sequence string. This allows us to encode the orientation of the sequence string by using only the label set $\{0,1,2\}$ (see the encoder C_3 in section 3). When all the instructions have been executed the generated graph has to be "cleaned": the s- and $\overline{s}$-labeled edges of the sequence string must be modified into $\perp$- or ε-labeled edges and the two additional end-point vertices must be removed.

More formally, we define $\mathcal{D} = (L, P)$ with $\mathcal{L}_v = \{\perp, \varepsilon\} \cup (I \cup \{L, R\}) \times V_7 \cup \{C, \overline{L}, \overline{R}, 0, 1, 2\}$, $\mathcal{L}_e = \{\perp, \varepsilon, s, \overline{s}\}$ and P is the set of relabeling rules defined below :

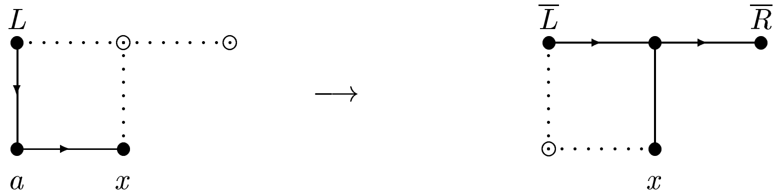
(i) (the empty case) The first rule R_ε will only be used when the axiom is the string-graph representation of the empty string (the corresponding graph is the empty graph) and is defined by:

R_ε :

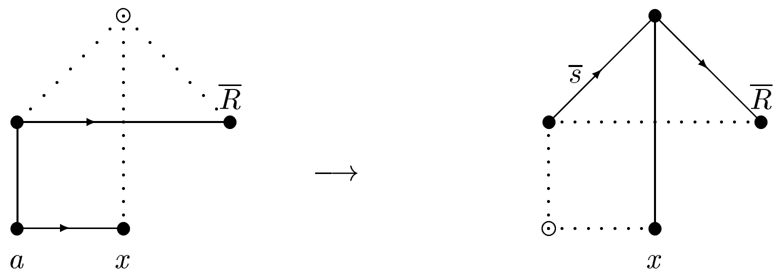


(ii) (execution of an 'a' instruction) We know that all the a instructions are located at the beginning of $\gamma(G)$. The first a will create the sequence string (rules $R_a^1(x)$) while the next ones will add a new vertex to the actual sequence (rules $R_a^2(x)$). The vertex thus created becomes current and the executed a instruction is erased from the instruction string. Hence, we define the two families of rules $R_a^1(x)$ and $R_a^2(x)$ for any $x \in I \cup \{R\}$ as:

$R_a^1(x)$:

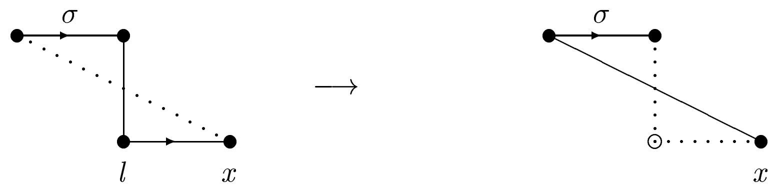


$R_a^2(x)$:



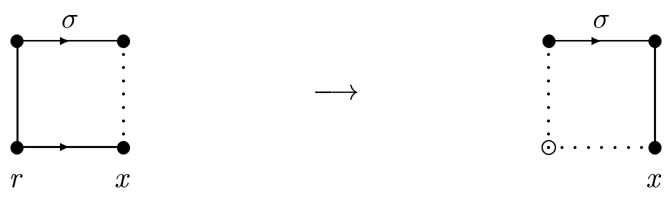
(iii) (execution of a 'l' instruction) The following rules will erase the l symbol from the instruction string and link the remaining instructions to the left neighbour of the current vertex. For any $x \in I \cup \{R\}$ and any $\sigma \in \{s, \overline{s}\}$ we then define the rules $R_l(\sigma, x)$ as:

$R_l(\sigma, x)$:



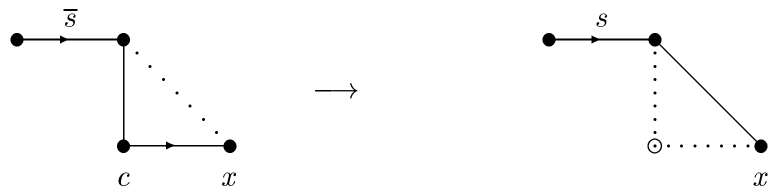
(iv) (execution of an 'r' instruction) The following rules will erase the l symbol from the instruction string and link the remaining instructions to the right neighbour of the current vertex. For any $x \in I \cup \{R\}$ and any $\sigma \in \{s, \overline{s}\}$ we then define the rules $R_l(\sigma, x)$ as:

$R_r(\sigma, x)$:



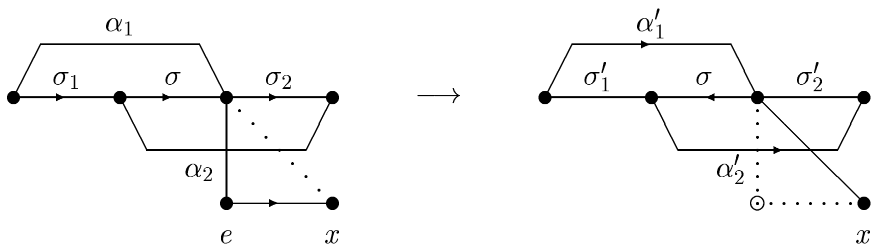
(v) (execution of a 'c' instruction) The following rules will connect the current vertex to its left neighbour, that is will relabel the corresponding $\overline{s}$-labeled edge into a s-labeled one. The instruction string is updated as before (the current vertex remains current). For any $x \in I \cup \{R\}$, the rules $R_c(x)$ are then defined as:

$R_c(x)$:



(vi) (execution of an 'e' instruction) In order to execute an e instruction, it is necessary to reorder the sequence string: if the actual sequence string has the form uz_1xyz_2v, with y as the current vertex, then it must become uz_1yxz_2v (y remains current). This can be done by linking y and z_1, as well as x and z_2, by an s-labeled or $\overline{s}$-labeled edge depending on whether they were initially linked by an edge or not. On the other hand, the edges $\{z_1, x\}$, $\{x, y\}$ and $\{y, z_2\}$ must be relabeled by ε or $\perp$ depending on whether they were s- or $\overline{s}$-labeled, and the orientation of the edge $\{x, y\}$ must be reversed. The instruction string is updated as usual and the rules $R_e(\sigma, \sigma_1, \sigma_2, \alpha_1, \alpha_2, x)$ for any $x \in I \cup \{R\}$, $\sigma, \sigma_1, \sigma_2 \in \{s, \overline{s}\}$ and $\alpha_1, \alpha_2 \in \{\perp, \varepsilon\}$ thus described are defined as:

$R_e(\sigma, \sigma_1, \sigma_2, \alpha_1, \alpha_2, x)$:

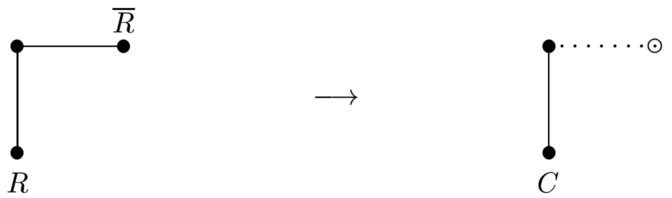


where for $1 \leq i \leq 2$, $\sigma'_i = \varepsilon$ if $\sigma_i = s$ and $\perp$ otherwise,

and for $1 \leq i \leq 2$, $\alpha'_i = s$ if $\alpha_i = \varepsilon$ and $\overline{s}$ otherwise.

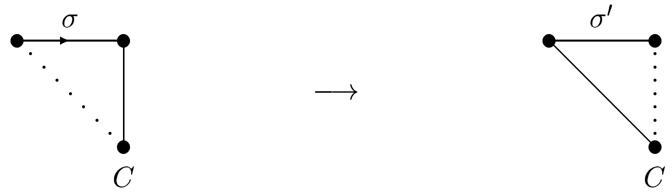
(vii) (cleaning the generated graph) When the instruction string has been completely executed (the current vertex is the rightmost vertex in the sequence string) we must traverse the sequence string from right to left in order to clean up the s- and $\overline{s}$-labeled edges as well as the two end-point vertices (with labels $\overline{L}$ and $\overline{R}$). The first of these rules to be applied is the rule R_{sc} (starting the cleaning process) which lead to a special C-labeled vertex (the "cleaner"):

R_{sc} :



Then the cleaner vertex will update all the s- and $\overline{s}$-labeled edges by using the rules $R_c(\sigma)$, $\sigma \in \{s, \overline{s}\}$ defined as:

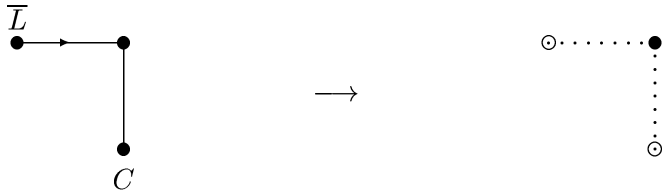
$R_c(\sigma)$:



where $\sigma' = \varepsilon$ if $\sigma = s$ and $\perp$ otherwise.

When the cleaner vertex reaches the leftmost vertex of the sequence string, it deletes the $\overline{L}$-labeled vertex by the following rule R_{lc} (last cleaning):

R_{lc} :



Note that all of the above rules have been designed in such a way that the orientation component of the vertices of the sequence string can always be updated in an adequate way whenever it is necessary (rules $R_a^1(x)$, $R_a^2(x)$ and $R_e(\sigma, \sigma_1, \sigma_2, \alpha_1, \alpha_2, x)$). Note also that the cleaning rules $R_c(\sigma)$ erase that orientation component when traversing the sequence string.

Starting from an axiom Z_G (the string-graph representation of $\gamma(G)$), it is not difficult to see that the system $\mathcal{D}$ is such that:

(i) If $\gamma(G) = \varepsilon$ the only rule which can be applied is R_ε, leading to the (irreducible) empty graph.
(ii) If $\gamma(G) \neq \varepsilon$, the only rule which can be applied first is $R_a^1(x)$, which initializes the sequence string.
(iii) At any time, there is only one of the above rules which can be applied and that application leads to an intermediate sequence string which exactly reflects the effect of the corresponding executed instruction.

Hence, with any axiom Z_G, exactly one derivation sequence in $\mathcal{D}$ can be associated that leads to a T-labeled graph isomorphic to G itself. □

Remark 14 For simplicity, we have restricted ourselves to the case of unlabeled graphs. It is not difficult to modify our encoding (and the corresponding decoder) in order to handle labeled graphs: to encode an undirected labeled graph with vertex label set L_v and edge label set L_e we simply replace the instruction a by the instructions (a, x) for any $x \in L_v$ and the instruction c by the instructions (c, y) for any $y \in L_e$. In the same way, by using the directed version of expanded graphs (see [24]) we can also prove that there exists an *e-GRS* which can decode any string encoding a directed labeled graph (for such directed graphs we no longer need to use orientation labels).

7 e-GRS's and Recursively Enumerable Sets of Graphs

In order to prove that any recursively enumerable set of graphs can be generated by an *e-GRS* the basic idea is to merge the system $\mathcal{R}_G$ (where G stands for the phrase-structure string grammar generating the desired set of strings) defined in section 4 and the system $\mathcal{D}$ defined in the previous section. However, we must ensure that the decoding process (realised by the rules of $\mathcal{D}$) does not start before

the complete generation of a terminal string-graph by $\mathcal{R}_G$ is achieved. This can be done by taking a phrase-structure string grammar G in a normal form given in [14] (and also used in [20]) which has the following properties:
(i) every string production in G is either a context-free production $B \longrightarrow \alpha$ with $B \in N,\ \alpha \in (T \cup N)^*$ or a production of the form $BC \longrightarrow BD$ with $B, C, D \in N$,
(ii) in any derivation of G the leftmost symbol remains a non-terminal symbol until the last production is applied.
Since the first applicable production in $\mathcal{D}$ always uses the leftmost (terminal) symbol and its auxiliary L-labeled left neighbour this normal form will obviously give us the required behaviour of the global system.

Hence, we finally obtain:

Theorem 15 *The sets of graphs generated by e-GRS's are exactly the recursively enumerable sets of labeled graphs. Moreover, for any recursively enumerable set $\mathcal{S}$ of unlabeled graphs, there exists an e-GRS which generates $\mathcal{S}$.*

Proof. For any *e-GRS*, it is not difficult to build a Turing machine which enumerates the set of strings encoding the graphs it generates. This set of graphs is thus recursively enumerable. Conversely, let $\mathcal{S}$ be a recursively enumerable set of graphs. There exists a phrase-structure string grammar $G = (N, T, P, A)$, given in the normal form discussed above, which generates the set $St(\mathcal{S}) = \{\gamma(H)\ /\ H \in \mathcal{S}\}$. By Theorem 11 we can construct an *e-GRS* $\mathcal{R}_G$ such that $\mathbf{L}_{T_1}(\mathcal{R}_G, Z_A) = \{Z_H,\ Z_H$ is a string representation of $\gamma(H),\ H \in \mathcal{S}\}$. By renaming some labels if necessary, we can assume that $\overline{L}, \overline{R}, C, s$ and $\overline{s}$ are not used in $\mathcal{R}_G$ and then define the system $\mathcal{R}_S$ as the union (in an obvious way) of $\mathcal{R}_G$ and $\mathcal{D}$. Due to the properties of the string grammar G, every derivation in $\mathcal{R}_S$ will be of the form $Z_A \xrightarrow[\mathcal{R}_G]{*} Z_H \xrightarrow[\mathcal{D}]{*} H$ and the result follows by taking as terminal labels set $T_2 = (\{\perp\} \cup L_v, \{\perp\} \cup L_e)$ where L_v (resp. L_e) stands for the vertex (resp. edge) label set of the considered labeled graphs. Note that this construction also holds for any recursively enumerable set $\mathcal{S}$ of unlabeled graphs. □

8 Discussion

In this paper, we have introduced and illustrated a new graph grammar model motivated from the previously studied graph relabeling systems. The main characteristics of this model is that it is not based on a classical graph replacement operation but on the relabeling of vertices and edges. Hence, no explicit embedding mechanism is needed : each relabeling rule application consists in modifying some labels of the relabeled occurrence (which allows some kind of *logical erasing*) and possibly adding some new components which are not linked to the context-graph of the relabeled occurrence.

We have shown that this model has the power of recursive enumerability. This clearly indicates that general *e-GRS*'s are too powerful to have "nice" properties : most of the non-trivial questions about the generated languages must be undecidable. Thus, it would be interesting to define "good" restrictions of the global model. In particular, it would be useful to obtain a characterization of *context-free e-GRS*'s (see [4]) and to see whether classical context-free graph grammars remain expressible in that new subclass or not.

It would be interesting to examine the effect on the generative power of some restrictions such as bounding the number of components in the left-hand sides of the relabeling rules (e.g. using "handles" as left-hand sides would provide a model which is exactly the edge-replacement model of Habel and Kreowski [9]), which seems to determine a strict hierarchy of *e-GRS*'s. Another possibility would be to restrict the "erasing" capabilities of *e-GRS*'s by means of some adequate constraints defined on the right-hand sides of the relabeling rules.

The notions of priority and of forbidden contexts, used in graph relabeling systems [3, 15] as control mechanisms for the applicability of relabeling rules, can also be extended to *e-GRS*'s [24, 25]. Their respective influence on the above-stated restrictions is not immediate and their study would extend the results obtained in [15].

References

[1] M. Bauderon, B. Courcelle, *Graph expressions and graph rewriting*, Math. System Theory **20** (1987) 83–127.

[2] F. Berman, G. Shannon, *Edge grammars: decidability results and formal language issues*, Proc. 22nd Allerton Conference on Communication, Control and Computing, Urbana, IL (1984).

[3] M. Billaud, P. Lafon, Y. Métivier and E. Sopena, *Graph rewriting systems with priorities*, Lecture Notes in Comput. Sci. **411** (1989) 94–106.

[4] B. Courcelle, *An axiomatic definition of context-free rewriting and its application to NLC grammars*, Theoret. Comput. Sci. **55** (1987) 141–181.

[5] B. Courcelle, *The monadic second-order logic of graphs VI: on several representations of graphs by relational structures*, Discrete Applied Math. **54** (1994) 117–149.

[6] H. Ehrig, *A tutorial introduction to the algebraic approach of graph grammars*, Third Int. Workshop on Graph Grammars and their Applications to Computer Science, Lecture Notes in Comput. Sci. **291** (1987) 3–14.

[7] H. Ehrig, M. Pfender, H.-J. Schneider, *Graph grammars: an algebraic approach*, Proc. 14th Annual IEEE Symp. Switch. Automat. Theory, Iowa City (1973) 167–180.

[8] J. Engelfriet, G. Rozenberg, *Graph grammars based on node rewriting: an introduction to NLC graph grammars*, Lecture Notes in Comput. Sci. **532** (1991) 12–23.

[9] A. Habel, H.-J. Kreowski, *Characteristics of graph languages generated by edge replacement*, Theoret. Comput. Sci. **51** (1987) 81–115.

[10] A. Habel, H.-J. Kreowski, *May we introduce to you: hyperedge replacement*, Lecture Notes in Comput. Sci. **291** (1987) 15–26.

[11] M. Harrison, *Introduction to formal language theory*, Addison-Wesley (1978).

[12] D. Janssens, G. Rozenberg, *On the structure of node label controlled graph languages*, Inform. Sci. **20** (1980) 191–216.

[13] D. Janssens, G. Rozenberg, *Restrictions, extensions and variations of NLC grammars*, Inform. Sci. **20** (1980) 217–244.

[14] H.-C.-M. Kleijn, M. Penttonen, G. Rozenberg, K. Salomaa, *Direction independent context-sensitive grammars*, Inform. and Control **63** (1984) 113–117.

[15] I. Litovsky, Y. Métivier, E. Sopena, *Different local controls for graph relabeling systems*, to appear in Math. System Theory (1994).

[16] I. Litovsky, Y. Métivier, W. Zielonka, *The power and limitations of local computations on graphs and networks*, Proceedings of Graph-Theoretic Concepts in Computer Science (WG'92), Lecture Notes in Comput. Sci. **657** (1993) 333–345.

[17] M. Löwe, *Algebraic approach to single-pushout graph transformation*, Theoret. Comput. Sci. **109** (1993) 181–224.

[18] J.-L. Pfaltz, A. Rosenfeld, *Web grammars*, Proc. 1st Int. Joint Conf. on Artificial Intelligence, Washington (1969) 609–619.

[19] M.-G. Main, G. Rozenberg, *Handle NLC grammars and R.E. Languages*, J. Comput. System Sci. **35** (1987) 192–205.

[20] M.-G. Main, G. Rozenberg, *Edge-label controlled graph grammars*, J. Comput. System Sci. **40** (1990) 188–228.

[21] M. Nagl, *A tutorial and bibliographical survey on graph grammars*, in Proc. 1st Int. Workshop on Graph Grammars and their Applications to Computer Science and Biology, Ehrig, Claus & Rozenberg (Eds), Lecture Notes in Comput. Sci. **79** (1979).

[22] J.-C. Raoult, *On graph rewritings*, Theoret. Comput. Sci. **32** (1984) 1–24.

[23] J.-C. Raoult, *Set-theoretic graph rewriting*, Lecture Notes in Comput. Sci **776** (1994), 312–325.

[24] E. Sopena, *Expanding graph relabeling systems*, Research Report 92-76, University Bordeaux I (1992).

[25] E. Sopena, *A generative approach of graph relabeling systems*, Research Report 93-7, University Bordeaux I (1993).

[26] T. Uesu, *A system of graph grammars which generates all recursively enumerable sets of graphs*, Tsukuba J. Math. **2** (1978) 11–26.