LaBRI Research Report No. Compiled on April 11, 2001
The final version of this paper has been published in *Math. System Theory* **28** (1995), 41–65.

DIFFERENT LOCAL CONTROLS FOR GRAPH RELABELLING SYSTEMS[1]

Igor LITOVSKY, Yves MÉTIVIER and Éric SOPENA
Laboratoire Bordelais de Recherche en Informatique, Unité associée C.N.R.S. 1304, 351, cours de la Libération, 33405 Talence, France.

Abstract. We are interested in models to encode and to prove decentralized and distributed computations on graphs. In this paper, we define and compare six models of graph relabelling systems. These systems do not change the underlying structure of the graph on which they work, but only the labelling of its components (edges or vertices). Each relabelling step is fully determined by the knowledge of a fixed size subgraph, the local context of the relabelled occurrence. The studied families are based on the relabelling of partial or induced subgraphs and we use two kinds of mechanisms to locally control the applicability of rules : a priority relation on the set of rules or a set of forbidden contexts associated with each rule. We show that these two basic (i.e. without local control) families of graph relabelling systems are distinct, but whenever we consider the local controls of the relabelling, the four so-obtained families are equivalent.

Keywords. Graph relabelling systems, Graph grammars.

1 Introduction

We are interested in models to encode and to prove decentralized and distributed computations on graphs. The presented models are graph relabelling systems satisfying the following constraints which seem to be natural when describing distributed computations with a decentralized control:

($C1$) they do not change the underlying graph but only the labelling of its components (edges and/or vertices), the final labelling being the result of the computation,

($C2$) they are *local*, that is, each relabelling step changes only a connected subgraph of a fixed size in the underlying graph,

($C3$) they are *locally generated*, that is, the application condition of the relabelling only depends on the *local context* of the relabelled subgraph.

For such systems, the distributed aspect comes from the fact that several relabelling steps can be performed simultaneously on "far enough" subgraphs, giving the same result as a sequential realisation of them, in any order.

In this paper, we define and compare six types of graph relabelling systems. Any such system, say $\mathcal{R}$, is defined by a finite set L of labels (labels used in the relabelled graphs), a set $I \subseteq L$ of initial labels (every graph starting a relabelling process has only labels in I) and a finite set of relabelling rules ; it may be equipped with a mechanism which locally controls the application of the relabelling rules. A relabelling rule r consists of the relabelling of a fixed connected subgraph G_r :

$$r : (G_r, \lambda) \longrightarrow (G_r, \lambda')$$

[1]With the support of the PRC Mathématiques et Informatique, the European Basic Research Action ESPRIT No 3166 ASMICS and the ESPRIT-Basic Research Working Group "COMPUGRAPH II".

We say that a labelled graph (G, l) is relabelled into (G, l') by $\mathcal{R}$ if there exists a finite sequence of *allowed applications* (in a sense specified below) of relabellings in $\mathcal{R}$ leading from (G, l) to (G, l'). Given a noetherian graph relabelling system $\mathcal{R}$, we are interested in the function $Irred_{\mathcal{R}}$ which, with each initial graph (G, l), associates the set of *irreducible* graphs (i.e. where no allowed application of a rule is possible) obtained from (G, l). We say that two noetherian graph relabelling systems $\mathcal{R}$ and $\mathcal{R}$' are *equivalent* when they have the same set of initial labels and when $Irred_{\mathcal{R}} = Irred_{\mathcal{R}'}$. A family $\mathcal{F}_1$ of graph relabelling systems is *less powerful* than a family $\mathcal{F}_2$ if every noetherian graph relabelling system in $\mathcal{F}_1$ is equivalent to a graph relabelling system in $\mathcal{F}_2$. The families $\mathcal{F}_1$ and $\mathcal{F}_2$ are equivalent if each one is less powerful than the other one.

We now present the six types of graph relabelling systems we will consider in this paper. For each of them we have to specify the notion of *allowed application* of a rule r in a graph (G, l). The first criterium characterizing the applicability of r is given by the definition of an *occurrence* of the graph (G_r, λ) in (G, l). Such an occurrence may be :

- a partial subgraph of (G, l) isomorphic to (G_r, λ),
- an induced subgraph of (G, l) isomorphic to (G_r, λ).

Hence, we respectively obtain the families of $pGRS$'s and $iGRS$'s. We prove that the family of $pGRS$'s is strictly less powerful than the family of $iGRS$'s.

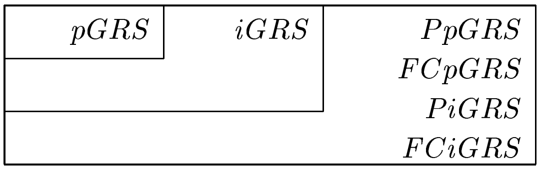
On the other hand, to increase the computational power of these basic graph relabelling systems, we use two kinds of local control on the applicability of rules :

- The first one has been introduced in [4] and consists of adding a partial order relation, called *priority*, on the set of relabelling rules. In such systems, the application of a rule r is *allowed* on an occurrence θ of (G_r, λ) if no rule with a greater priority has an occurrence overlapping θ. Note that the effect of these priorities is strictly local (constraint $(C3)$ is respected).
- The second one, inspired by [5], consists of adding to each relabelling rule r a set of *forbidden contexts*, where a context is a graph having (G_r, λ) as subgraph. For such systems, an application of r is *allowed* on an occurrence θ of (G_r, λ) if θ is not a subgraph of a forbidden context in (G, l).

Remark 1 Partial order and forbidden context conditions are known in formal language theory [6]. However, partial order of rules is used here in a strictly local way.

These relabelling systems are respectively called $PxGRS$'s and $FCxGRS$'s (for $x \in \{p, i\}$). It is easy to see that the so-defined families are strictly more powerful than the previous ones, and that the family of $FCxGRS$'s is more powerful than the family of $PxGRS$'s (for $x \in \{p, i\}$). The main part of this paper is devoted to proving the equivalence of the $FCpGRS$'s and the $PpGRS$'s. This result is not immediate : for example, it is easy to give a one-rule $FCpGRS$ "recognizing" the class of complete graphs, but no "simple" $PpGRS$ can do it. The main difficulty comes from the fact that a $FCpGRS$ forbids the application of a relabelling rule by only considering the forbidden contexts associated with this rule, since a $PpGRS$ only forbids such an application when another rule (with a greater priority) is applicable on an *overlapping* occurrence. Assuming first that one works on graphs having a distinguished vertex, depth-first traversals can be sequentially processed by using a $PpGRS$ (see example 1.4). In this case, every $FCpGRS$ can be simulated by a $PpGRS$ in the following way : each depth-first traversal attempts to apply a fc-rule ; when it has found one or more such rules, it "chooses" one of them and applies it ; when no fc-rule is applicable, the $PpGRS$ stops (see $\mathcal{R}_{locsim}$ in Section 4). But it is known that the problem of distinguishing one vertex (known as the *election problem*) is not solvable for arbitrary graphs (see [1, 2, 12]). Hence, the main idea of this paper is to construct, using a $PpGRS$, a partition of the graph into subgraphs (called *countries*) of k-bounded diameter (where k is the maximal diameter of the graphs in the rules of the $FCpGRS$), each country having an elected vertex (the *capital*). This "k-election" mechanism, used together with the $PpGRS$ $\mathcal{R}_{locsim}$, enables us to simulate every $FCpGRS$ by a $PpGRS$ (Proposition 6.1).

We also prove that with a local control Y (Priority or Forbidden Contexts), the $YpGRS$'s and the $YiGRS$'s are equivalent. The following scheme summarizes the relative powers of these different families.



This paper is organized as follows. Section 1 contains the definitions. Sections 2 to 5 describe the different steps used for proving Proposition 6.1 : the k-election problem is solved in Section 2, a $PpGRS$ enumerating m-tuples of vertices is given in Section 3 and used for the local simulation of a $FCpGRS$ by a $PpGRS$ in Section 4, Section 5 realizes the global simulation. In Section 6, the equivalence between $PpGRS$'s and $FCpGRS$'s is proved. In Section 7, we finally compare the other relabelling families we have introduced.

2 Definitions and notation

2.1 Graphs

A simple, loopless, undirected *graph* G [3, 7] is defined as a pair $(\boldsymbol{v}(G), \boldsymbol{e}(G))$ where $\boldsymbol{v}(G)$ is a finite set of vertices and $\boldsymbol{e}(G)$ a set of edges, an edge being a subset of two distinct vertices in $\boldsymbol{v}(G)$. Let v_1 and v_2 be two vertices in $\boldsymbol{v}$(G) ; a *path* p from v_1 to v_2 in G is a sequence $x_0, \ldots, x_n$ of vertices in $\boldsymbol{v}(G)$ such that for $0 \leq i < n$, $\{x_i, x_{i+1}\} \in \boldsymbol{e}(G)$, $x_0 = v_1$ and $x_n = v_2$; n is said to be the *length* of p. The graph G is *connected* if any two vertices in $\boldsymbol{v}(G)$ are linked by a path. The *distance* between two vertices v_1 and v_2 in G, denoted $d(v_1, v_2)$, is the length of the shortest path from v_1 to v_2. The maximal distance between any two vertices in $\boldsymbol{v}(G)$ is called the *diameter* of G.

Let $L = (L_v, L_e)$ be a pair of two finite sets of *labels* (L_v (resp. L_e) stands for the set of vertex (resp. edge) labels). A *labelled graph* is a pair (G, λ) where G is a graph and $\lambda = (\lambda_v, \lambda_e)$ where λ_v (resp. λ_e) is a mapping from $\boldsymbol{v}(G)$ (resp. $\boldsymbol{e}(G)$) to L_v (resp. L_e). We will denote by $|A|_{(G,\lambda)}$, or simply $|A|$, the number of A-labelled vertices (or edges) in (G, λ). Let (G, λ) and (G', λ') be two labelled graphs. (G, λ) is a *(partial) subgraph* of (G', λ') if

$$\begin{cases} \boldsymbol{v}(G) \subseteq \boldsymbol{v}(G'), \\ \boldsymbol{e}(G) \subseteq \boldsymbol{e}(G'), \\ \lambda = \lambda'|_G = (\lambda'_v|_G, \lambda'_e|_G). \end{cases}$$

where $\lambda'_v|_G$ (resp. $\lambda'_e|_G$) denotes the restriction of λ_v (resp. λ_e) to $\boldsymbol{v}(G)$ (resp. $\boldsymbol{e}(G)$).

Remark 2 From now on, we will simply use λ' instead of $\lambda'|_G$ whenever G is clearly given by the context.

An *injective* mapping θ from $\boldsymbol{v}(G)$ into $\boldsymbol{v}(G')$ is an *occurrence* of (G, λ) in (G', λ') if for any x, y in $\boldsymbol{v}(G)$, we have :

$$\begin{cases} \{x, y\} \in \boldsymbol{e}(G) \Longrightarrow \{\theta(x), \theta(y)\} \in \boldsymbol{e}(G') \\ \lambda_v(x) = \lambda'_v(\theta(x)) \\ \lambda_e(\{x, y\}) = \lambda'_e(\{\theta(x), \theta(y)\}) \end{cases}$$

Let θ be an occurrence of (G, λ) in (G', λ'); we will denote by $\theta(G)$ the graph $(\theta(\boldsymbol{v}(G)), \mathcal{E})$ where $\mathcal{E} = \{\{\theta(x), \theta(y)\} \ / \ \{x, y\} \in \boldsymbol{e}(G)\}$. Note that $(\theta(G), \lambda')$ is a subgraph of (G', λ').

Let (G, λ) be a subgraph of (G', λ'). We say that (G, λ) is an *induced* subgraph of (G', λ') iff for all x, y in $\boldsymbol{v}(G)$, $\{x, y\} \in \boldsymbol{e}(G') \Longleftrightarrow \{x, y\} \in \boldsymbol{e}(G)$. An occurrence θ of (G'', λ'') in (G', λ') is said to be an *induced occurrence* if $(\theta(G''), \lambda')$ is an induced subgraph of (G', λ').

Let r be an integer and x be a vertex in $\boldsymbol{v}(G)$; the *ball subgraph* $B(x, r)$ is the induced subgraph of (G, λ) whose vertices are all the vertices in $\boldsymbol{v}(G)$ whose distance to vertex x is at most r.

Remark 3 From now on, as we will only deal with connected labelled graphs, we will simply use *graph* to denote connected labelled graphs. By using a special "empty label", denoted by ε, we will be able to consider unlabelled graphs as labelled ones.

2.2 Relabelling of partial or induced subgraphs

A *partial-Graph Relabelling rule* is a triple $(G_r, \lambda_r, \lambda'_r)$, also denoted $(G_r, \lambda_r) \longrightarrow (G_r, \lambda'_r)$. (G_r, λ_r) (resp. (G_r, λ'_r)) is called the *left-hand side* (resp. *right-hand side*) graph of the rule r. The relabelling relation $\xrightarrow[r]{}$ is defined by : $(G, \lambda) \xrightarrow[r]{} (G, \lambda')$ if there exists an occurrence θ of (G_r, λ_r) in (G, λ) such that θ is an occurrence of (G_r, λ'_r) in (G, λ'), for any $x \in \boldsymbol{v}(G) \setminus \boldsymbol{v}(\theta(G_r))$, $\lambda_v(x) = \lambda'_v(x)$ and for any $e \in \boldsymbol{e}(G) \setminus \boldsymbol{e}(\theta(G_r))$, $\lambda_e(e) = \lambda'_e(e)$. We say that θ is the *relabelled* occurrence.

A *partial-Graph Relabelling System (pGRS)* is a triple $\mathcal{R} = (L, I, P)$ where $L = (L_v, L_e)$ is the set of labels, $I = (I_v, I_e)$, with $I_v \subseteq L_v$ and $I_e \subseteq L_e$, is the set of initial labels and P is a finite set of partial-graph relabelling rules, such that the graphs in these rules have labels in L. The relabelling relation $\xrightarrow[\mathcal{R}]{}$ is defined by : $(G, \lambda) \xrightarrow[\mathcal{R}]{} (G, \lambda')$ if and only if there exists a rule $r \in \mathcal{R}$ such that $(G, \lambda) \xrightarrow[r]{} (G, \lambda')$.

A *partial-Graph Relabelling System with Priorities (PpGRS)* is a triple $\mathcal{R} = (L, I, P)$ where L and I are defined as before and P is a finite set of relabelling rules equipped with a partial ordering relation $>$ called *priority* which works as follows : let θ be an occurrence of a rule $r \in \mathcal{R}$ in a graph (G, λ). The rule r is *applicable* to θ if there is no occurrence θ' of a rule $r' > r$ such that $\boldsymbol{v}(\theta(G_r)) \cap \boldsymbol{v}(\theta'(G_{r'})) \neq \emptyset$. If two or more rules are simultaneously applicable in (G, λ), one of them (nondeterministically chosen) is applied. We write $(G, \lambda) \xrightarrow[\mathcal{R}]{} (G, \lambda')$ if there exists a rule $r \in R$ such that $(G, \lambda) \xrightarrow[r]{} (G, \lambda')$ and r is applicable in (G, λ) to the relabelled occurrence. In the sequel, the priority of each rule may be specified by using an integer to indicate the ordering (whenever two rules have the same integer as priority, they are not comparable).

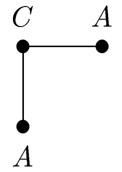
A *partial-Graph Relabelling rule with forbidden contexts (fc-rule for short)* is a pair $(r, \mathcal{H}_r)$ where r is a relabelling rule $(G_r, \lambda_r, \lambda'_r)$ and $\mathcal{H}_r$ is a finite family of pairs $\{((G_i, \lambda_i), \theta_i)\}_{i \in I_r}$ where (G_i, λ_i) is a graph (called *forbidden context*) and θ_i is an occurrence of (G_r, λ_r) in (G_i, λ_i). The forbidden contexts of the fc-rule are used as follows : let θ be an occurrence of (G_r, λ_r) in (G, λ); the fc-rule $(r, \mathcal{H}_r)$ is *applicable* to θ if for no i, there exists an occurrence φ_i of (G_i, λ_i) in (G, λ) such that $\varphi_i \theta_i = \theta$.

A *partial-Graph Relabelling System with Forbidden Context (FCpGRS)* is a triple $\mathcal{R} = (L, I, P)$ where L and I are defined as before and P is a finite set of fc-rules. We write $(G, \lambda) \xrightarrow[\mathcal{R}]{} (G, \lambda')$ if there exists a rule $r \in \mathcal{R}$ such that $(G, \lambda) \xrightarrow[r]{} (G, \lambda')$ and r is applicable in (G, λ) to the relabelled occurrence.

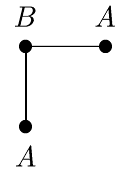
The same notions can be defined by using induced occurrences instead of partial ones, leading respectively to *i-*, *Pi-* and *FCiGRS*. We shall use $\xrightarrow[\mathcal{R}]{i}$ to denote the corresponding relabelling relations.

Example 4 Consider the graph (G, λ) of Figure 1(f) and the graph relabelling rules $r = (G_r, \lambda_r, \lambda'_r)$, $s = (G_s, \lambda_s, \lambda'_s)$, where (G_r, λ_r), (G_r, λ'_r), (G_s, λ_s) and (G_s, λ'_s) are given by Figure 1(a,b,d,e) respectively. Recall that unlabelled edges are considered as labelled with the empty label.

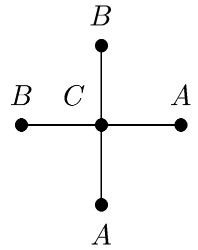
- As rule of a *pGRS*, r can be applied to the four corners of graph (G, λ) (vertices marked as $\circ$).
- As rule of a *iGRS*, r can be applied to each corner of graph (G, λ) except to the upper-right one, since there is a forbidden edge linking two vertices of the occurrence.
- As rule of a *FCpGRS*, with graph (G_1, λ_1) of Figure 1(c) as forbidden context, r can only be applied to the two upper corners of graph (G, λ).
- As rule of a *FCiGRS*, r can be applied to the upper-left corner of (G, λ) and to its bottom-right corner, since the forbidden context of r does not appear as an induced subgraph.
- As rule of a *PpGRS*, with $s > r$, rule r can only be applied to the two upper corners of (G, λ).



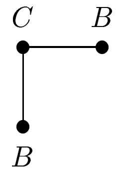
(a) The graph (G_r, λ_r)



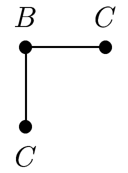
(b) The graph (G_r, λ'_r)



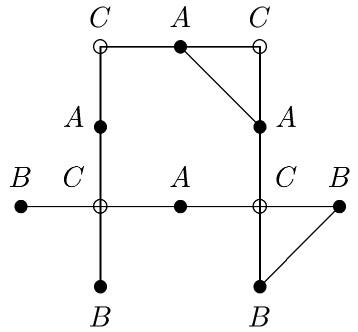
(c) The graph (G_1, λ_1)



(d) The graph (G_s, λ_s)



(e) The graph (G_s, λ'_s)



(f) The graph (G, λ)

Figure 1: Applicability of rewriting rules.

- As rule of a $PiGRS$, with $s > r$, rule r can be applied to the upper left corner, and the bottom-right corner of (G, λ) (since s cannot be applied to occurrences overlapping these corners).

2.3 Relabelling system behaviour

Any computation on a graph (by means of a relabelling system) must give a result in a finite time. Thus (unless explicitly stated otherwise) we will only consider *noetherian* graph relabelling systems, which means that from any initial graph, there exists no infinite relabelling chain.

Given a graph relabelling system $\mathcal{R}$, we consider the reflexive transitive closure $\xrightarrow[\mathcal{R}]{*}$ of $\xrightarrow[\mathcal{R}]{}$ (or $\xrightarrow[\mathcal{R}]{i}$). A graph (G, λ') is said to be *irreducible* with respect to $\mathcal{R}$ if no rule of $\mathcal{R}$ is applicable to (G, λ'). For every graph (G, λ) with labels in I, we denote by $Irred_{\mathcal{R}}((G, \lambda))$ the set of *irreducible* graphs obtained from (G, λ):

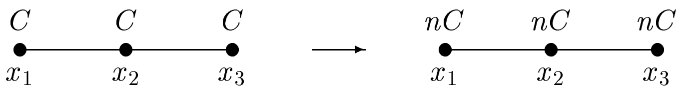
$$Irred_{\mathcal{R}}((G,\lambda)) = \{(G,\lambda') \ / \ (G,\lambda) \xrightarrow[\mathcal{R}]{*} (G,\lambda') \text{ and no rule of } \mathcal{R} \text{ is applicable to } (G,\lambda')\}$$

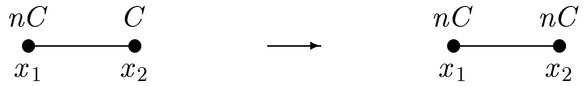
Let $\mathcal{R}$ and $\mathcal{R}'$ be two relabelling systems. The systems $\mathcal{R}$ and $\mathcal{R}'$ are said to be *equivalent* if $I = I'$ and for every graph (G, λ) with labels in I we have $Irred_{\mathcal{R}}((G, \lambda)) = Irred_{\mathcal{R}'}((G, \lambda))$. We will say that a family $\mathcal{F}_1$ of graph relabelling systems is *less powerful* than a family $\mathcal{F}_2$, if every graph relabelling system in $\mathcal{F}_1$ is equivalent to a graph relabelling system in $\mathcal{F}_2$. The families $\mathcal{F}_1$ and $\mathcal{F}_2$ are *equivalent* if each one is less powerful than the other one.

We now give some examples of various $GRS's$. To avoid any ambiguity between the left and right-hand sides of the relabelling rules (which correspond to the same underlying unlabelled graph) we will number the vertices by x_1, x_2,...

Example 5 The following $iGRS$ is such that $L = (\{C, nC\}, \{\varepsilon\})$, $I = (\{C\}, \{\varepsilon\})$ and P has two rules given below. We have the property that a graph $(G, \lambda') \in Irred((G, \lambda))$ has only C-labels if G is a complete graph, it has only nC-labels if G is not complete.

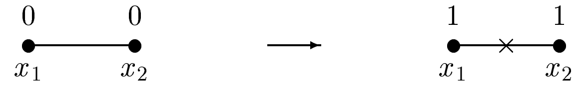
R_1

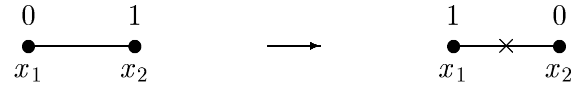


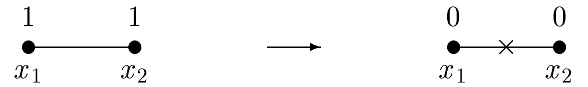
R_2


Rule R_1 is applied whenever the two vertices associated with x_1 and x_3 are not linked by an edge. In such a case, the graph G is not a complete graph, and rule R_2 broadcasts the nC-label to all other vertices. Whenever the graph G is complete, no rule can be applied.

Example 6 The following $pGRS$ is such that $L = (\{0, 1\}, \{\varepsilon, \times\})$, $I = (\{0\}, \{\varepsilon\})$ and P has three rules given below. An irreducible graph will be such that all its vertices are labelled 0 or 1 according to the parity of their degree.

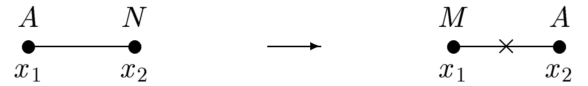
R_{00}


R_{01}


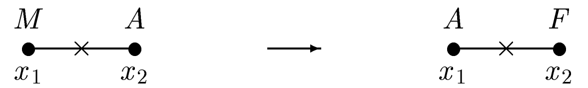
R_{11}


Every rule marks an edge of the graph and updates the labels of its end-points. When all the edges are marked, the so-obtained graph is irreducible.

Example 7 The following $PpGRS$ is such that $L = (\{N, A, M, F\}, \{\varepsilon, \times\})$, $I = (\{A, N\}, \{\varepsilon\})$ and P has two rules given below. This $PpGRS$ computes a spanning forest (i.e. a subgraph which is a forest including all the vertices) of the initial graph such that each tree is "rooted" at a vertex with initial label A. The intuitive idea is the following : from each initially A-labelled vertex starts a computation which consists in "attaching" free vertices (i.e. with a N-label) by marking the corresponding edges (with a $\times$-label).

R_1


priority 1

R_2


priority 0

Every tree in the spanning forest is computed in a depth-first way : every active (A-labelled) vertex chooses one of its free neighbours which becomes active (rule R_1). When an active vertex has no more free neighbours, it activates its father, i.e. the M-labelled vertex to which it is linked by a marked edge (rule R_2). If it does not have such a neighbour, that means that it was one of the initially A-labelled vertices. At the end of the computation, any initially A-labelled vertex has label A, every initially N-labelled vertex has label F. Note that we are interested in initial graphs with at least one A-labelled vertex (otherwise, the initial graph is irreducible). If there is exactly one initially A-labelled vertex x, this $PpGRS$ constructs a spanning tree rooted at x.
Figure 2 shows a sample derivation of this $PpGRS$.

Remark 8 The computation of any tree in the spanning forest is done "sequentially" : at any moment, there is only one active vertex in each tree. We can increase the "parallelism" of this computation by simply replacing the M-label of the right-hand side of rule R_1 by a A-label and deleting rule R_2. The computation is no longer "depth-first-search-like" (neither "breadth-first-search-like") and the vertices of an irreducible graph are then all A-labelled.

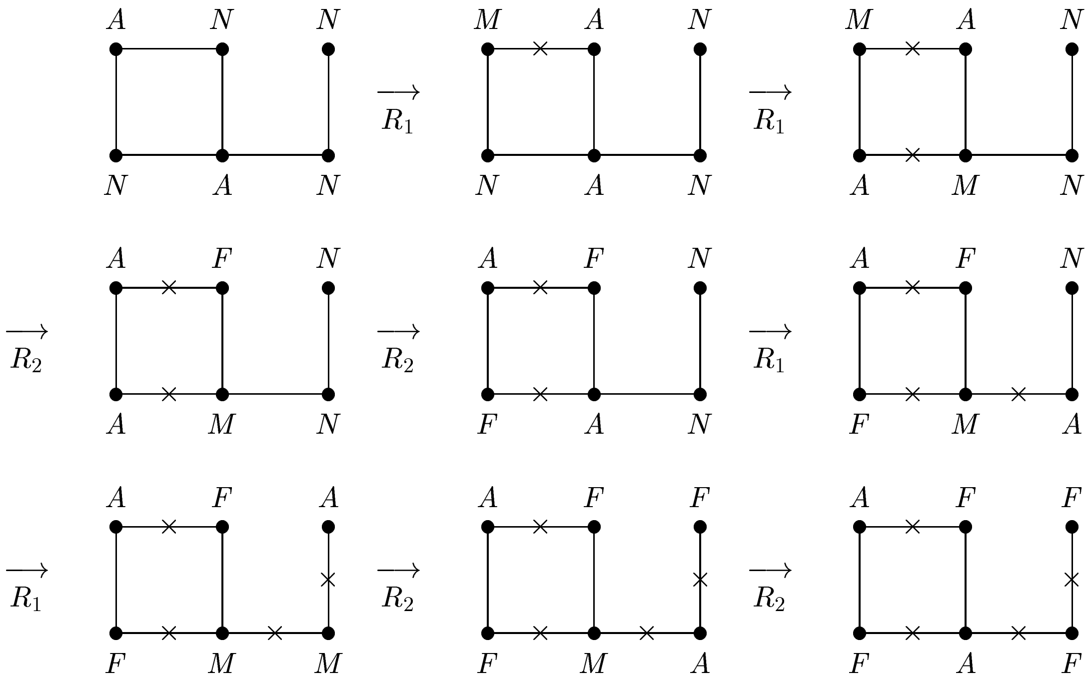
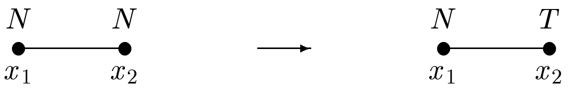


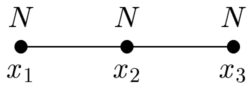
Figure 2: Computing a spanning forest.

Example 9 The following *FCpGRS* is such that $L = (\{N, T\}, \{\varepsilon\})$, $I = (\{N\}, \{\varepsilon\})$ and P has two rules given below. We have the property that an irreducible graph (G, λ) has only T-labels if and only if G is a tree. The first relabelling rule is:

R_1

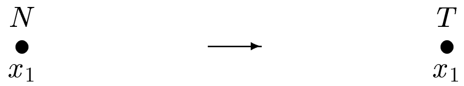


and its forbidden context is:

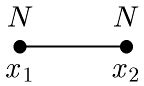


The second rule is:

R_2



and its forbidden context is:

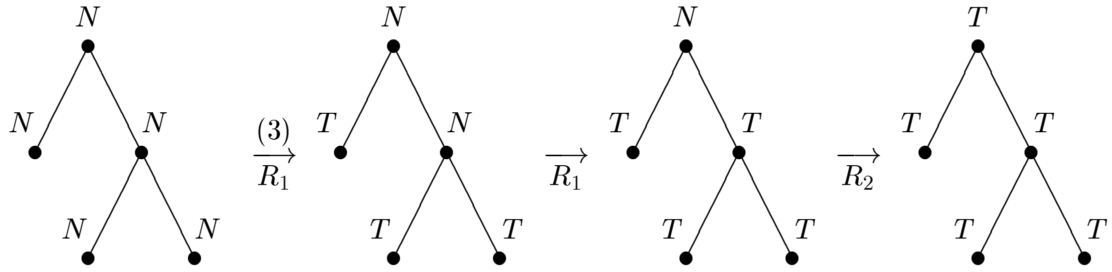


The forbidden context of R_1 forces x_2 to have exactly one neighbour with label N, and the forbidden context of R_2 forces x_1 to have no neighbour with label N.

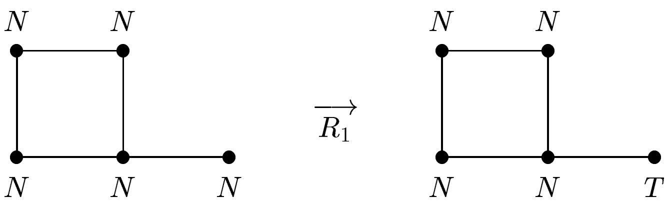
Figure 3 shows two sample derivations of this *FCpGRS*, each one leading to an irreducible graph.

3 The k-election problem

It is not possible to elect exactly one vertex in a graph with a *PpGRS* [9]. For this reason, we are interested in "local" elections called *k-elections* where k is a given integer. The *k-election problem* on a graph can be intuitively introduced as follows. Each vertex of the graph stands for a town, each edge for a road segment joining two different towns. Initially, each town has a *neutral* status. We want



(a) Derivation of a tree.



(b) Derivation of a cyclic graph.

Figure 3: The recognition of trees.

to organize the graph by delimiting *countries*, each country having one *capital*. In each country, the distance between any town and the capital must at most be k. Moreover, the distance between any two capitals in the graph must be at least $k+1$. Each capital (resp. each town) has also to "know" the towns (resp. the capital) of its country — in the sense that there is a marked path between a capital and each of its towns.

The *PpGRS* which solves this problem will fulfill two additional requirements : the towns in all the countries will be classified according to their distance to the capital and any town will belong to one of the countries whose capital is the nearest. This will be done by constructing a "spanning forest" of the initial graph (i.e. a partial subgraph of the initial graph which is a forest including every vertex). Each tree (standing for a country) of this forest will be rooted at a capital.

To solve this problem, we consider the *PpGRS* $\mathcal{R}_{k-elec} = (L, I, P)$. The set of labels $L = (\{N, C, T, T_1, ..., T_k\}, \{\times, \varepsilon\})$ where N stands for *Neutral*, C for *Capital*, T for a town belonging to a country (but not yet classified), T_i $(1 \leq i \leq k)$ for a classified town, $\times$ for marked edges (i.e. edges belonging to the spanning forest) and ε (the empty symbol) for unmarked edges. The set of initial labels is given as $I = (\{N\}, \{\varepsilon\})$.

The set of rules P is given in Figure 4. Labels X_i stand for any vertex label. Except when it is explicitly specified (rules $R_4(i)$), any edge can be marked or unmarked, and this is preserved in the right-hand side of any rule.

The priorities are given as :

$$R_1 < R_4(k) < R_4(k-1) < \cdots < R_4(1) < \left\{R_3(i)\right\}_{k<i\leq 2k} < \left\{R_2(i)\right\}_{1\leq i\leq k}$$

Rule R_1 says that any neutral town can spontaneously become a capital, except if there already exists a capital in its k-neighbourhood (rules $R_2(i)$ prevent rule R_1 to be applied).

Rules $R_2(i)$ are intended to mark any town in the k-neighbourhood of a capital as a town belonging to a country (label T). The classification of these towns will be done later, by using rules $R_4(i)$.

Rules $R_3(i)$ have been essentially introduced for technical purposes. Thanks to them, a capital will only begin to classify its towns when the capitals of its neighbouring countries are elected. This

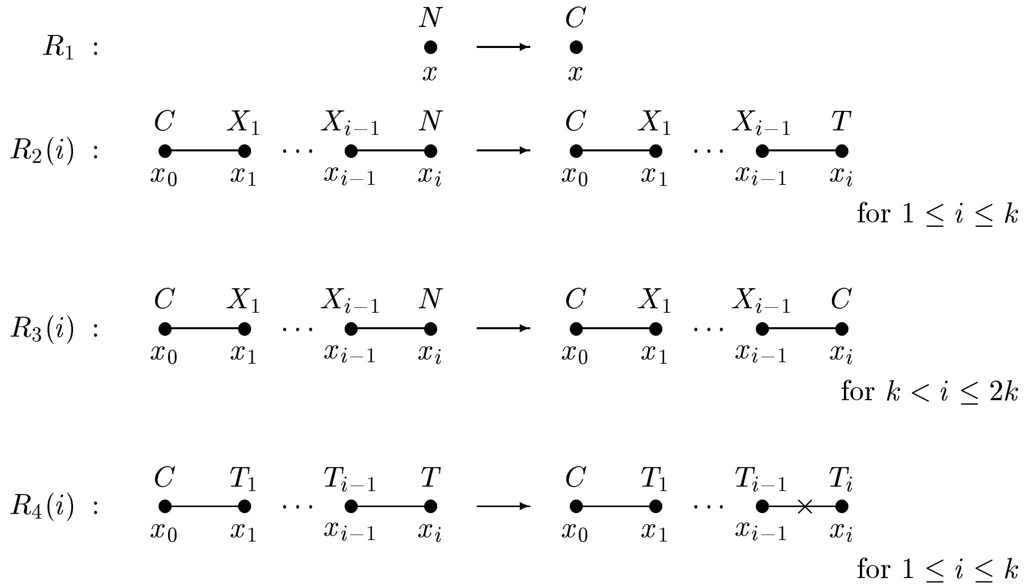


Figure 4: The *PpGRS* $\mathcal{R}_{k-elec}$.

(a) A bad intermediate configuration

(b) A good terminal configuration

Figure 5: The classification problem.

will ensure a good classification of the towns in a country. Figure 5 shows what kind of problem would arise if we do not use such rules : let $k = 2$ and suppose vertex x_0 becomes a capital. But for rules $R_3(i)$, it can mark vertices x_1 and x_2 as classified towns (see figure 5(a)). Then the system terminates by electing, say x_3, as a capital and relabelling x_4 by T; x_4 remains an unclassified town. Figure 5(b) shows a good terminal configuration, obtained by using rules $R_3(i)$.

Rules $R_4(i)$ implement the classification of the towns. Their priority ensures that each town will be labelled according to its distance to the nearest capital (a T_i-label means that this town is at distance i from the nearest capital).

We now give some results on the behaviour of the *PpGRS* $\mathcal{R}_{k-elec}$.

Proposition 10 *The PpGRS $\mathcal{R}_{k-elec}$ is noetherian. Moreover, if (G, λ) is a graph with unmarked edges and n vertices all N-labelled, the number of relabelling steps in any derivation sequence issued from (G, λ) is bounded by $2n$.*

Proof. The termination criteria is given by the strict decreasing of the ordered pair of labels $(|N|, |T|)$: whenever the number of N's does not decrease the number of T's does.

Any vertex is successively labelled by the sequence (N, C) or (N, T, T_i). As every rule modifies the label of at least one vertex, we can say that the length of any derivation sequence is bounded by the quantity $2n$. $\square$

Proposition 11 *Let (G, λ) be a graph with every vertex N-labelled and with unmarked edges. Let (G, λ') be a graph such that :*

$$(G, \lambda) \underset{\mathcal{R}_{k-elec}}{\overset{*}{\longrightarrow}} (G, \lambda')$$

Then the graph (G, λ') satisfies :

$(P1)$ *If x and y are two C-labelled vertices of (G, λ') then the distance from x to y is at least $k+1$.*

$(P2)$ *If x is a T-labelled vertex of (G, λ'), then there exists a C-labelled vertex y at distance at most k from vertex x.*

$(P3)$ *If x is a T_i-labelled $(i > 1)$ vertex in (G, λ'), then there exists no N-labelled vertex at distance less than $k+1$ from x.*

$(P4)$ *The subgraph induced by the marked edges is a forest with all vertices labelled C, T_1, ..., T_{k-1} or T_k and such that :*

- *Every tree has exactly one C-labelled vertex,*
- *Every T_i-labelled vertex x belongs to that subgraph and is connected to the C-labelled vertex of its tree by a marked path whose length is i.*

$(P5)$ *If x is a T_i-labelled vertex of (G, λ'), then we have $d(x, \{\lambda'^{-1}(C)\}) = i$.*

Proof. All these properties are obviously true for the initial graph. Let us suppose that (G, λ_i) satisfies these properties and let (G, λ_{i+1}) be such that

$$(G, \lambda_i) \underset{\mathcal{R}_{k-elec}}{\longrightarrow} (G, \lambda_{i+1})$$

We have to show that (G, λ_{i+1}) also satisfies these properties. Most of them are very easy to check. Hence, we only give proofs of $(P1)$ and $(P3)$ as examples.

$(P1)$ Only rules R_1 and $R_3(i)$ can lead to a C-labelled vertex. This property is guaranteed by the greater priority of rules $R_2(i)$.

$(P3)$ Every T_i-labelled vertex, say x, is issued from a rule $R_4(i)$. If there exists a N-labelled vertex y at distance less than $k+1$ from x, rules $R_2(i)$ or $R_3(i)$ should have been applied first, thanks to the priorities.

□

We now state the main result of this section, which says that the *PpGRS* $\mathcal{R}_{k-elec}$ solves the k-election problem.

Theorem 12 *Let (G, λ) be a graph with unmarked edges and whose vertices are all N-labelled. Then, any graph (G, λ') in $Irred_{\mathcal{R}_{k-elec}}((G, \lambda))$ is such that the subgraph induced by the marked edges is a spanning forest of (G, λ') with all vertices labelled C, T_1, ..., T_{k-1} or T_k and satisfying :*

- *Every tree of that forest has exactly one C-labelled vertex,*
- *The distance between two C-labelled vertices is at least $k+1$,*
- *Every T_i-labelled vertex x satisfies $d(x, \lambda'^{-1}(C)) = i$.*

Proof. If (G, λ') is irreducible, it contains neither N-labelled vertices (rules R_1, $R_2(i)$ or $R_3(i)$ should be applicable) nor T-labelled vertices (rule $R_4(i)$ should be applicable). Hence, (G, λ') only contains C-, T_1-, ..., T_{k-1}- or T_k-labelled vertices. Property $(P1)$ ensures that the distance between two C-labelled vertices is at least $k+1$. By property $(P5)$, the subgraph induced by the marked edges is a spanning forest of (G, λ'), each tree having exactly one C-labelled vertex. Moreover, by property $(P6)$ each T_i-labelled vertex satisfies the stated property. □

4 The m-enumeration problem

Let G be a graph and m, r be two integers. We can construct a *PpGRS* which enables us to "enumerate" all the m-tuples of distinct vertices in a ball $B(x, r)$ (given together with a rooted spanning tree $T(x)$) for any "given" vertex x in G. A *given* vertex x means that x has a special label which does not appear elsewhere in G and "a *PpGRS* $\mathcal{R}$ *enumerates* all the m-tuples of $B(x, r)$" means that during a computation of $\mathcal{R}$ on G, for every m-tuple $(y_1, \dots, y_m)$ of $B(x, r)$, there is exactly one intermediate relabelling λ such that $h(\lambda_v(y_1)) = 1, \dots, h(\lambda_v(y_m)) = m$, and $h(\lambda_v(x)) = *$ for all $x \notin \{y_1, \dots, y_m\}$ where h is a fixed mapping from L_v to $\{1, \dots, m, *\}$. Note that we can obtain a spanning tree of $B(x, r)$, rooted at x, by using an obvious variation of the *PpGRS* of example 1.4 (in which all nodes of the tree are forced to have a distance at most r to x).

Let us intuitively describe the behaviour of such a *PpGRS* : vertex x will be the controller of the computation. It first looks for a vertex, say v_1, which can be chosen as the first component of a new m-tuple. When such a vertex is found, it looks for a second one and so on. When for i given vertices $v_1, \dots, v_i$ the system has enumerated all the m-tuples having these vertices as first (ordered) components, vertex v_i is marked as having been the i^{th} component of all such m-tuples and a new i^{th} component is searched. All these steps will be handled by depth-first traversals : vertex x initiates a traversal which looks for a given vertex ; when the control returns to x, it initiates a new traversal for the next search. This computation terminates when all the vertices in the graph are marked as having been the first component of all possible m-tuples.

Lemma 13 *There is a (non noetherian) PpGRS $\mathcal{R}_{trav}$ which, given a rooted tree $T(x)$, makes alternating tree traversals of $T(x)$.*

Proof. The *PpGRS* $\mathcal{R}_{trav}$ that we now describe works on $T(x)$ where the vertex x is the root of $T(x)$. Initially the root has label R_0 and all other vertices label N_0. In the first traversal there is always exactly one A_0-labelled vertex (the "active" vertex, initially the root), all vertices on the path from the root to the active vertex have label W, and the other vertices have label N_0 (if they have not yet been visited) or label N_1 (if they have been visited). At the end of the first traversal the root has label R_1 and all other vertices label N_1. The second traversal can then be made (simply interchange the roles of 0's and 1's) and the system indefinitely alternates such traversals.

In the following rules, $i \in \{0, 1\}$ and R stands for *Root*, A for *Active*, N for *Neutral* and W for *Waiting*:

R_1	$\overset{R_i}{\underset{x_1}{\bullet}}$	$\longrightarrow$	$\overset{A_i}{\underset{x_1}{\bullet}}$	priority -1
R_2	$\overset{A_i}{\underset{x_1}{\bullet}}$ — $\overset{N_i}{\underset{x_2}{\bullet}}$	$\longrightarrow$	$\overset{W}{\underset{x_1}{\bullet}}$ — $\overset{A_i}{\underset{x_2}{\bullet}}$	priority 2
R_3	$\overset{W}{\underset{x_1}{\bullet}}$ — $\overset{A_i}{\underset{x_2}{\bullet}}$	$\longrightarrow$	$\overset{A_i}{\underset{x_1}{\bullet}}$ — $\overset{N_{1-i}}{\underset{x_2}{\bullet}}$	priority 1
R_4	$\overset{A_i}{\underset{x_1}{\bullet}}$	$\longrightarrow$	$\overset{R_{1-i}}{\underset{x_1}{\bullet}}$	priority -1

□

Proposition 14 *Let m and r be two integers $(m, r > 0)$. There exists a PpGRS $\mathcal{R}_{enum}$ which, given a graph (G, λ) and any vertex $x \in \boldsymbol{v}(G)$, enumerates all m-tuples of the ball $B(x, r)$ in (G, λ).*

Proof. As the vertex x is given, we may assume that a rooted spanning tree $T(x)$ of $B(x, r)$ is constructed, which means that some edges in (G, λ) are marked as belonging to $T(x)$. The *PpGRS* $\mathcal{R}_{enum}$ that we now describe works on $T(x)$ (all the edges in the following rules have to be considered as marked). Each computation of $\mathcal{R}_{enum}$ corresponds to a sequence of alternating tree traversals of $T(x)$ based on the *PpGRS* $\mathcal{R}_{trav}$.

Every vertex $v \in B(x, r)$ is labelled by a couple or a triple of components. The first component is a label of $\mathcal{R}_{trav}$: R_i, A_i, N_i or W. The second component is a m-tuple of labels $(l_1, \ldots, l_m)$ describing the state of v with respect to the enumeration of m-tuples. These labels are such that :

- $l_j = 1$ means that v is the j^{th} component in the current m-tuple,
- $l_j = \overline{1}$ means that v has been the j^{th} component of all the m-tuples whose $j-1$ first components are the (unique) vertices such that $l_1 = 1, \ldots, l_{j-1} = 1$,
- $l_j = 0$ means that v is not in one of the previous cases.

Finally the active vertex (with A_i or R_i as first component) has an additional "action label" : $Search_j$ for Searching the j^{th} component of a m-tuple, $Return_j$ when the j^{th} component has been found (then the active vertex has to return back to the root), $Reset_j$ when the j^{th} component has been used as j^{th} component of all the m-tuples having the same j first components and has to be marked as such (i.e., has to change its l_j-label from 1 to $\overline{1}$), and *Stop* when the computation has terminated.

Initially, for every vertex, we have $l_j = 0$, $\forall\, j \in \{1, \ldots, m\}$. Moreover, the root has a $(A_0, Search_1)$ label and any other vertex has a N_0 label. The irreducible graphs are such that for every vertex, $l_1 = \overline{1}$ and $l_j = 0, 1 < j \leq m$.

The *PpGRS* $\mathcal{R}_{enum}$ contains rules R_2 to R_4, extended with all possible values of the second and third components of the labels (but does not contain the rule R_4 when the active label has a *Stop* component). The role of rule R_4 is taken over by rules $R_6(j)$, $R_7(j)$ and $R_{10}(j)$, below.

In the following rules of $\mathcal{R}_{enum}$, 0_j (resp. 1_j, $\overline{1}_j$) means that $l_j = 0$ (resp. $l_j = 1$, $l_j = \overline{1}$), and only the components involved in the relabelling are specified.

$R_5(j)$ is used when the j^{th} component of a m-tuple is found.

$$R_5(j) \qquad \underset{x_1}{\overset{(A_i, 0_j, Search_j)}{\bullet}} \quad \longrightarrow \quad \underset{x_1}{\overset{(A_i, 1_j, Return_j)}{\bullet}} \qquad \text{priority } 4$$

for an active vertex such that $\forall\, x \in \{1, \ldots, j-1, j+1, \ldots, m\}, l_x \neq 1$:

When such a j^{th} component has been found, the $(A_i, Return_j)$ label finishes its traversal, returns to the root by means of rules R_2 and R_3 of $\mathcal{R}_{trav}$ with the corresponding labels of $\mathcal{R}_{enum}$ and becomes $(R_{1-i}, Return_j)$ by rule R_4. Then, when $j < m$, the root initiates a new traversal for searching a $(j+1)^{\text{th}}$ component. This is done by the following rules :

$$R_6(j; j < m) \qquad \underset{x_1}{\overset{(R_i, Return_j)}{\bullet}} \quad \longrightarrow \quad \underset{x_1}{\overset{(A_i, Search_{j+1})}{\bullet}} \qquad \text{priority } 0$$

When a m^{th} component has been found and the $(A_i, Return_m)$ label has returned to the root, this m^{th} component must be marked (it can no more be the m^{th} component of a m-tuple having the same $m-1$ first vertices). This process is initiated by rule $R_6(m)$, and done by rules $R_8(m)$ and rules R_2 to R_4 of $\mathcal{R}_{trav}$ with the corresponding labels of $\mathcal{R}_{enum}$. Next, a new m^{th} component has to be searched, which is initiated by rules $R_{10}(m)$.

$$R_6(m) \qquad \underset{x_1}{\overset{(R_i, Return_m)}{\bullet}} \quad \longrightarrow \quad \underset{x_1}{\overset{(A_i, Reset_m)}{\bullet}} \qquad \text{priority } 0$$

When a j^{th} component ($j > 1$) has not been found after a complete traversal (this means that all remaining vertices have already been the j^{th} component or that there is not enough vertices in the ball), the vertex with label 1_{j-1} must be marked. This process is initiated by rule $R_7(j)$, and done by rules $R_8(j-1)$ and R_2 to R_4. In such a case, all the vertices with labels $\bar{1}_j$ have to be reset to 0_j (the $(j-1)^{\text{th}}$ component will change). This will be done by rules $R_9(j)$. Next, a new $(j-1)^{\text{th}}$ component has to be searched, which is initiated by rules $R_{10}(j-1)$.

$$R_7(j; j>1) \qquad \underset{x_1}{\overset{(R_i, Search_j)}{\bullet}} \longrightarrow \underset{x_1}{\overset{(A_i, Reset_{j-1})}{\bullet}} \qquad \text{priority } 4$$

$$R_8(j) \qquad \underset{x_1}{\overset{(A_i, 1_j, Reset_j)}{\bullet}} \longrightarrow \underset{x_1}{\overset{(A_i, \bar{1}_j, Reset_j)}{\bullet}} \qquad \text{priority } 4$$

$$R_9(j; j>1) \qquad \underset{x_1}{\overset{(A_i, \bar{1}_j, Reset_{j-1})}{\bullet}} \longrightarrow \underset{x_1}{\overset{(A_i, 0_j, Reset_{j-1})}{\bullet}} \qquad \text{priority } 4$$

$$R_{10}(j) \qquad \underset{x_1}{\overset{(R_i, Reset_j)}{\bullet}} \longrightarrow \underset{x_1}{\overset{(A_i, Search_j)}{\bullet}} \qquad \text{priority } 4$$

When a first component has not been found during a complete depth-first tree traversal, the *PpGRS* stops (all the vertices have been the first component of any possible m-tuple).

$$R_7(1) \qquad \underset{x_1}{\overset{(R_i, Search_1)}{\bullet}} \longrightarrow \underset{x_1}{\overset{(A_i, Stop)}{\bullet}} \qquad \text{priority } 0$$

As every tree traversal stops, the strict increasing of the $3m$-tuple $(|\bar{1}_1|, \ldots, |\bar{1}_m|, |1_1|, \ldots, |1_m|, |Reset_1|, \ldots, |Reset_m|)$ evaluated whenever the root has a label R_i is a termination criterium for the complete *PpGRS* $\mathcal{R}_{enum}$. Hence $\mathcal{R}_{enum}$ is noetherian.

Furthermore, we have the following invariant properties, ensuring that the *PGRS* $\mathcal{R}_{enum}$ enumerates all the m-tuples of vertices of $B(x, r)$:

(P_1) if $Search_j$ is the current Action label, then for $1 \leq x < j$ exactly one vertex has a component $l_x = 1$ and for $j \leq y \leq m$ every vertex has a component $l_y \neq 1$

(P_2) $l_j = \bar{1}$ implies that the vertex v has been the j^{th} component of all the m-tuples such that :

- (C_1) $\forall\, x < j$, the x^{th} component is the unique vertex such that $l_x = 1$
- (C_2) the j^{th} component is the vertex v.

At each moment that $Return_m$ has just become the action label (by rule $R_5(m)$), a new m-tuple $(y_1, \ldots, y_m)$ is enumerated : for every $1 \leq j \leq m$ exactly one vertex y_j has a component $l_j = 1$. The mapping h from L_v to $\{1, \ldots, m, *\}$ mentioned in the beginning of this section can now be defined in an obvious way (e.g., $h(A_i, 1_m, Return_m) = m$). □

5 Local Simulation of a *FCpGRS* by a *PpGRS*

Let $\mathcal{R}$ be a *FCpGRS*. In this section, we describe a *PpGRS* $\mathcal{R}_{locsim}$ working on a country of a given capital c (in the sense defined in Section 2) within a given graph (G, λ). We assume that a rooted spanning tree $T(c)$ of the country has been constructed. Then $\mathcal{R}_{locsim}$ will realize a nondeterministic application on $T(c)$ of one applicable fc-rule of $\mathcal{R}$ when such a rule exists.

Proposition 15 *Let $\mathcal{R}$ be a FCpGRS and let k be the greatest diameter of a graph in the definition of $\mathcal{R}$. There exists a PpGRS $\mathcal{R}_{locsim}$ which, for any given rooted tree $T(c)$ in a given graph (G, λ), can test whether a fc-rule r of $\mathcal{R}$ is applicable in (G, λ) to an occurrence θ of G_r such that $\theta(G_r)$ is a partial subgraph of the ball $B(v, k)$ for some vertex v of $T(c)$. Furthermore, a nondeterministic application of such an applicable fc-rule (when it exists) is done.*

Proof. By using a depth-first tree traversal, $\mathcal{R}_{locsim}$ activates every vertex $v \in T(c)$. Then, from any vertex v, $\mathcal{R}_{locsim}$ marks in (G, λ) a rooted spanning tree $T(v)$ of the ball $B(v, k)$ in (G, λ). This ball may include vertices of other countries but only of countries which are *near* the country of v (two countries C_1, C_2 are *near* if their respective capitals are at distance at most $3k$: thus a ball $B(v, k)$ with center in C_1 can intersect C_2). Hence, it will be the main point of the global simulation in Section 5 to ensure that two near countries are not simultaneously active.

For each fc-rule $(r, \mathcal{H}_r)$ in the *FCpGRS* $\mathcal{R}$, with $r = (G_r, \lambda_r, \lambda'_r)$, the *PpGRS* $\mathcal{R}_{locsim}$ uses a *PpGRS* $\mathcal{R}_r$ that tests wether r is applicable to an occurrence of G_r that lies inside $B(v, k)$. Let m be the number of vertices of G_r. To find all occurrences of G_r in $B(v, k)$, $\mathcal{R}_r$ will enumerate all m-tuples of vertices of $B(v, k)$. Thus, $\mathcal{R}_r$ is obtained from the *PpGRS* $\mathcal{R}_{enum}$ by adding the checking and the eventual application of r whenever we have a label $(A_i, 1_m, Found_m)$. Labels $(A_i, 1_m, Found_m)$ appear instead of the labels $(A_i, 1_m, Return_m)$ of rules $R_5(m)$, and mean that we have found a new m-tuple and that we must try to apply the fc-rule r on the m-tuple. The *PpGRS* $\mathcal{R}_{locsim}$ (that we will not describe in detail) activates the *PpGRS*'s $\mathcal{R}_r$ one after another (in some fixed order) to ensure that all fc-rules of $\mathcal{R}$ are tried.

Let us now describe the *PpGRS*'s $\mathcal{R}_r$. We first choose a numbering $v_1, \ldots, v_m$ of the vertices of G_r. Next, for each vertex $v_j \in \boldsymbol{v}(G_r)$, we concatenate to $\lambda_v(v_j)$ a label chosen from $\mathcal{R}_{enum}$: $(W_i, 1_j)$ or $(N_i, 1_j)$ if $j < m$ and a label $(A_i, 1_m, Found_m)$ if $j = m$. Hence, for each of these choices, we obtain a new labelling of G_r. Let γ be such a labelling of G_r and let (G_r, γ, γ') be the new corresponding relabelling rule. For each pair $((G_l, \lambda_l), \theta_l) \in \mathcal{H}_r$, we consider all possible graphs (G_l, γ_l) where for every $x \in \boldsymbol{v}(G_l) \cup \boldsymbol{e}(G_l)$, $\gamma_l(x)$ is the concatenation of the label $\lambda_l(x)$ and a label from $\mathcal{R}_{enum}$. The labelling γ_l must be such that for every $x \in \boldsymbol{v}(G_r) \cup \boldsymbol{e}(G_r)$, $\gamma_l(\theta_l(x)) = \gamma(x)$. Furthermore a label *Done* or *Notdone* is possibly added to the label A_i. *Done* means that r has been applied; *Notdone* means that r is applicable, but it has not been applied.

Now we can add the following rules to the *PpGRS* $\mathcal{R}_{enum}$ in order to obtain the *PpGRS* $\mathcal{R}_r$ that simulates the application of the fc-rule r.

$$R_{51}(r) \qquad \underset{x_1}{\overset{(A_i, 0_m, Search_m)}{\bullet}} \quad \longrightarrow \quad \underset{x_1}{\overset{(A_i, 1_m, Found_m)}{\bullet}} \qquad \text{priority } 5$$

for an active vertex such that $\forall\, x \in \{1, \ldots, m-1\}, l_x \neq 1$.

$$R_{52}(r, l) \qquad (G_l, \gamma_l) \quad \longrightarrow \quad (G_l, \gamma'_l) \qquad \text{priority } 7$$

where $\gamma_l = \gamma'_l$ except for the component $(A_i, 1_m, Found_m)$ which becomes $(A_i, 1_m, Return_m)$. For every possible γ and γ_l (depending on how we add labels of $\mathcal{R}_{enum}$) such rules are added.

$$R_{53}(r) \qquad (G_r, \gamma) \quad \longrightarrow \quad (G_r, \gamma') \qquad \text{priority } 6$$

where $\gamma = \gamma'$ but the "λ part" of the label is replaced by λ' and the vertex numbered m has now a label $(A_i, Done)$ instead of A_i or $(A_i, Notdone)$.

$$R_{54}(r) \qquad (G_r, \gamma) \quad \longrightarrow \quad (G_r, \gamma'') \qquad \text{priority } 6$$

where $\gamma = \gamma''$ but the component $(A_i, 1_m, Found_m)$ of the vertex numbered m has become $(A_i, Notdone, 1_m, Return_m)$.

$$R_{55}(r) \qquad \underset{x_1}{\overset{(A_i, 1_m, Found_m)}{\bullet}} \quad \longrightarrow \quad \underset{x_1}{\overset{(A_i, 1_m, Return_m)}{\bullet}} \qquad \text{priority } 5$$

Rules $R_{51}(r)$ replace rules $R_5(m)$ and mean that a new current m-tuple is found.

Rules $R_{52}(\mathrm{r}, l)$ mean that the current m-tuple is an occurrence θ of G_r, such that $\theta(G_r)$ is in the context (G_l, λ_l). In this case, the fc-rule r cannot be applied to the current m-tuple and a new

m-tuple must be looked for.

Rules $R_{53}(r)$ mean that the fc-rule r is applicable to the current m-tuple (thanks to the respective priorities of $R_{52}(r,l)$ and $R_{53}(r)$) and that the relabelling has been done. Then the label $(A_i, Done)$ stops the enumeration of m-tuples and gives a label $Done$ to the root v of the tree $T(v)$ (and $\mathcal{R}_{locsim}$ will then give label $Done$ to the capital c, in a way that is not specified here).

Rules $R_{54}(r)$ mean that the fc-rule r is applicable to the current m-tuple (thanks to the respective priorities of $R_{52}(r,l)$ and $R_{54}(r)$) but the relabelling has not been done. Rules $R_{53}(r)$ and Rules $R_{54}(r)$ have the same priority, thus they have the same probability to be applied. This possibility of non-application ensures a nondeterministic application of the fc-rules of $\mathcal{R}$ in T(c).

Rule $R_{55}(r)$ means that the current m-tuple is not an occurrence of G_r. Hence, a new m-tuple must be searched.

When all fc-rules of $\mathcal{R}$ are sequentially processed, if the capital c has a label $Notdone$, a new searching of applicable fc-rules is started, and the first applicable fc-rule is applied (we do not describe the $PpGRS$). Thus a nondeterministic application of an applicable fc-rule of $\mathcal{R}$ in $T(c)$, if it exists, is made by the complete $PpGRS$ $\mathcal{R}_{locsim}$ so constructed. Furthermore, when $\mathcal{R}_{locsim}$ stops, if a relabelling has been made, the capital c has the label $Done$ and if there is no applicable rule in $T(c)$, the capital c has the label $Nothing$ (the $Nothing$-labelling is not described). □

6 Simulating the activity of a *FCpGRS* by using a *PpGRS*

To achieve the simulation of a given $FCpGRS$ $\mathcal{R}$ by a $PpGRS$, we now give a $PpGRS$ $\mathcal{R}_{actsim}$ which supervises the *activity* of capitals. A capital x is said to be *active* if the $PpGRS$ $\mathcal{R}_{locsim}$ from the previous section is looking for applying a fc-rule in the country of the capital x.

Let k be the greatest diameter of a graph in the definition of $\mathcal{R}$. Given a graph (G,λ), we consider the graph $Cap(G)$ whose vertices are the capitals obtained via a k-election in (G,λ), and whose edges are linking two capitals when these two capitals are near (i.e. whose distance is at most $3k$). We are going to give a $PpGRS$ $\mathcal{R}_{cap}$ which simulates, in $Cap(G)$, the *activity* on (G,λ) of every execution of the given $FCpGRS$ $\mathcal{R}$. More precisely, let us assume that we have a graph (G,λ) where a k-election has been made. An execution of $\mathcal{R}$ on G is defined by the sequence of relabelled occurrences $\theta_1,\ldots,\theta_n$ in G. Each occurrence θ_i intersects one or more countries with capitals $C_{i,1},\ldots,C_{i,j_i}$ respectively. The goal of the $PpGRS$ $\mathcal{R}_{cap}$ is to ensure that first, one of the capitals $C_{1,1},\ldots,C_{1,j_1}$ is active (step$_1$), next one of the capitals $C_{2,1},\ldots,C_{2,j_2}$ is active (step$_2$),..., next one of the capitals $C_{n,1},\ldots,C_{n,j_n}$ (step$_n$) is active, and finally every capital must be inactive (i.e. N-labelled in the $PpGRS$ of Figure 6). Thus, at each step$_i$ $(1 \le i \le n)$, the $PpGRS$ $\mathcal{R}_{locsim}$ may simulate the relabelling of $\mathcal{R}$ on θ_i $(1 \le i \le n)$. To obtain a $PpGRS$ no longer working on $Cap(G)$, but on the whole graph (G,λ), it is sufficient to consider that the edges in Figure 6 are in fact paths of length at most $3k$.

Proposition 16 *Let $\mathcal{R}$ be a $FCpGRS$. There exists a $PpGRS$ $\mathcal{R}_{actsim}$ which, for every given graph (G,λ), can simulate on $Cap(G)$ the activity of any execution of the given $FCpGRS$ $\mathcal{R}$.*

Proof. We use the following labels for every capital $x \in Cap(G)$:

- W means that the country of x is Waiting to be active,
- A means that the country of x is Active : the $PpGRS$ $\mathcal{R}_{locsim}$ is looking for applying a fc-rule of $\mathcal{R}$ in the country of x (after which the label of x becomes D or N),
- D (= $Done$) means that a fc-rule has been applied to an occurrence containing a vertex in the country of x,
- N (= $Nothing$) means that no fc-rule can be applied to an occurrence containing a vertex of the country of x.

All capitals are initially W-labelled. We must ensure that:

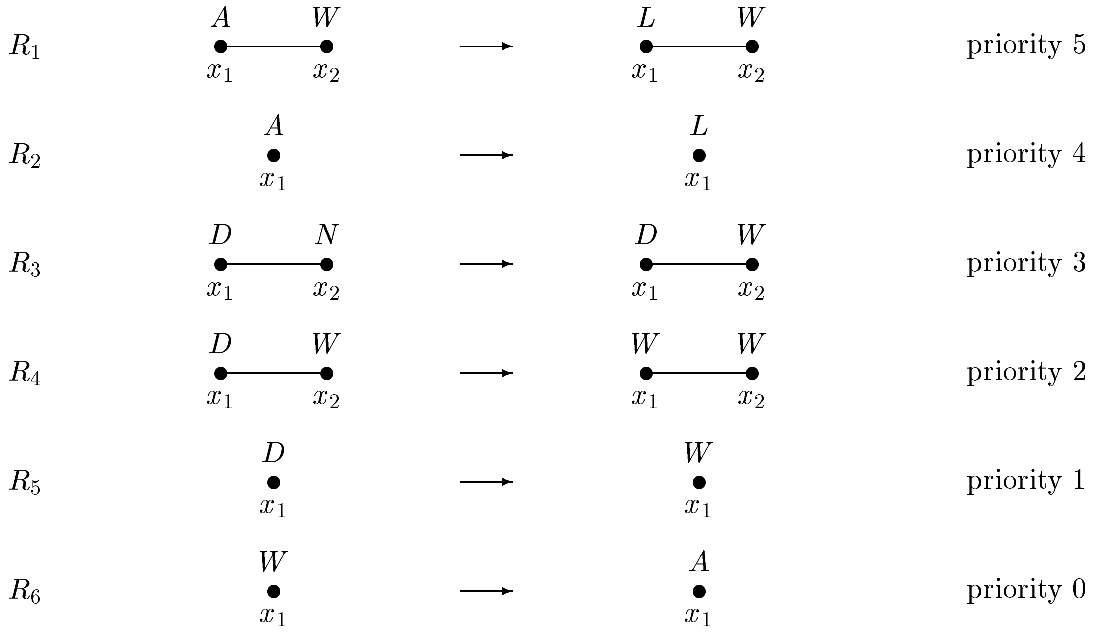


Figure 6: The *PpGRS* $\mathcal{R}_{actsim}$ of global simulation

(P_1) two neighbouring capitals in $Cap(G)$ are not simultaneously Active, to ensure that two local simulations do not work on a common subgraph of G (see Section 4),

(P_2) whenever a capital x has applied a fc-rule to an occurrence which intersects its country, all neighbouring capitals of x in $Cap(G)$ and the capital x itself will become Waiting,

(P_3) every Waiting capital will become Active.

The *PpGRS* $\mathcal{R}_{actsim}$ is given by the rules in Figure 6 where L stands for a label D or N. Rule R_1 indicates that the local simulation of the *FCpGRS* in the country of the A-labelled capital has terminated. The relative priorities of R_1 and R_6 ensure that two neighbouring capitals are not simultaneously active ((P_1) is satisfied). Rule R_2 does the same thing when the capital has no W-labelled neighbouring capitals. Rule R_3 is used when a capital has done an application and allows us to "wake up" a sleeping capital. Then rule R_4 is applied when all the neighbouring capitals have been waked up and the D-labelled capital becomes itself W-labelled ((P_2) is satisfied). Rule R_5 concerns graphs having only one capital. Rule R_6 activates a waiting capital ((P_3) is satisfied).

Furthermore we have the following properties:

(P_4) $\forall \{x,y\} \in \boldsymbol{e}(Cap(G)),\ \lambda_v(x) = A \Longrightarrow \lambda_v(y) \in \{N, W\}$ (indeed, according to $R_6 < R_1$ we have $\lambda_v(y) \neq A$, and according to $R_6 < R_4$ we have $\lambda_v(y) \neq D$),

(P_5) $\forall \{x,y\} \in \boldsymbol{e}(Cap(G)),\ \lambda_v(x) = D \Longrightarrow \lambda_v(y) \in \{N, W\}$ (according to (P_4)),

(P_6) whenever a vertex x becomes *Done*, all its neighbours are W-labelled or will become again W-labelled (according to (P_5) and rule R_3),

(P_7) a D-labelled vertex will be W-labelled only when all its neighbours are W-labelled (according to R_4, $R_5 < R_3$),

(P_8) $\forall \{x,y\} \in \boldsymbol{e}(Cap(G))$, if $\lambda_v(x) = D$ and $\lambda_v(y) = W$, y will stay W-labelled as long as x is D-labelled (according to (P_5)).

Thanks to these properties, the *PpGRS* $\mathcal{R}_{actsim}$ simulates on $Cap(G)$ the activity of any execution of the *FCpGRS* $\mathcal{R}$ on (G, λ) (if no rule of $\mathcal{R}$ is applicable, then all capitals will become N-labelled, and no rule $R_1, \ldots, R_6$ is then applicable). Conversely any execution of the previous *PpGRS* is a simulation of an execution of the *FCpGRS* $\mathcal{R}$ in (G, λ). □

7 Equivalence between *PpGRS*'s and *FCpGRS*'s

We are now going to show that *PpGRS's* and *FCpGRS's* are in fact equivalent. We first give, for any *FCpGRS* $\mathcal{R}$, a *PpGRS* $\mathcal{R}$' which can simulate the behaviour of $\mathcal{R}$ on any graph G labelled with initial labels. The intuitive idea is the following one : let k (resp. m) be the greatest diameter (resp. number of vertices) of the graphs in the rules of $\mathcal{R}$. We first construct a covering of the graph G by means of countries and capitals (two capitals being at a distance at least $k+1$ from each other). Then, any capital can test whether a fc-rule can or cannot be applied on an occurrence overlapping its country, by enumerating the m-tuples in its neighbourhood, and apply one of these rules when possible. The whole activity of the capitals is managed by the *PpGRS* $\mathcal{R}_{actsim}$ seen in the previous section.

Proposition 17 *The class of FCpGRS's is less powerful than the class of PpGRS's.*

Proof. Let $\mathcal{R}$ be a given *FCpGRS*. Putting together the *PpGRS*'s constructed in Sections 2, 4 and 5 with appropriate priorities we obtain a *PpGRS* $\mathcal{R}'$ equivalent to $\mathcal{R}$. First, we observe that the W-labels in $\mathcal{R}_{actsim}$ are here C-labels from *PpGRS* $\mathcal{R}_{k-elec}$. Rule R_1 of $\mathcal{R}_{actsim}$ is replaced by the rules of $\mathcal{R}_{locsim}$ where paths of length at most 3k are added from the Active vertex to a C-labelled vertex. The so-constructed rules (with the priorities of $\mathcal{R}_{locsim}$) lead to a *PpGRS* called $\mathcal{R}'_{locsim}$. Finally, rule R_2 of $\mathcal{R}_{actsim}$ is replaced by the rules of $\mathcal{R}_{locsim}$.

The priorities of the so-obtained *PpGRS* $\mathcal{R}$' are given as:

$$\mathcal{R}_{k-elec} > \mathcal{R}'_{locsim} > \mathcal{R}_{locsim} > R_{3,actsim} > R_{4,actsim} > R_{5,actsim} > R_{6,actsim}$$

where in each used *PpGRS*, the respective priorities of the rules are respected. Giving the greatest priorities to the rules of $\mathcal{R}_{k-elec}$ we ensure that a capital is Active (i.e. A-labelled) only if all near capitals (i.e. at a distance at most 3k) have been elected (i.e. have a C-label). In such a system, simulations of relabelling may be processed before the k-election has globally terminated, but the greater priorities of the rules of $\mathcal{R}_{k-elec}$ ensure that the near countries are constructed. Thus the so-processed relabelling steps are allowed. □

Conversely, for any *PpGRS* $\mathcal{R}$, one can easily construct a *FCpGRS* $\mathcal{R}'$ which simulates the behaviour of $\mathcal{R}$. For any rule r in $\mathcal{R}$, one can characterize the contexts which have to prevent the application of r : it suffices to take into account all the rules r' in $\mathcal{R}$ which have a higher priority than r and which can overlap an occurrence of G_r. Hence, we obtain the following result :

Proposition 18 *The class of PpGRS's is less powerful than the class of FCpGRS's.*

Proof. Let $\mathcal{R}$ be a *PpGRS*. We are going to construct a *FCpGRS* $\mathcal{R}_1$ which, for any graph (G, λ), leads to the same irreducible graphs (by allowing the same relabelling steps than $\mathcal{R}$). For each rule $r \in \mathcal{R}$, we consider the set $\mathcal{S}$ of rules $r' \in \mathcal{R}$ which have a higher priority than r. For every $r' \in \mathcal{S}$, we consider the set $G_{r'} \uplus G_r$ of pairs $((G, \lambda), \theta)$ defined as follows. A pair $((G, \lambda), \theta)$ belongs to $G_{r'} \uplus G_r$ iff θ is an occurrence of G_r in (G, λ) and there exists an occurrence θ' of $G_{r'}$ such that:

$$\boldsymbol{v}(G) = \boldsymbol{v}(\theta(G_r)) \cup \boldsymbol{v}(\theta'(G_{r'})) \text{ and } \boldsymbol{v}(\theta(G_r)) \cap \boldsymbol{v}(\theta'(G_{r'})) \neq \emptyset$$

Then $(r, G_{r'} \uplus G_r)$ is the fc-rule associated in $\mathcal{R}_1$ with r. Thus a rule $r \in \mathcal{R}$ is applicable to an occurrence θ iff the fc-rule $(r, G_{r'} \uplus G_r) \in \mathcal{R}_1$ is applicable to θ. Hence, the *PpGRS* $\mathcal{R}$ and the so-constructed *FCpGRS* $\mathcal{R}_1$ allow exactly the same relabelling steps. □

Finally, we obtain the following result :

Theorem 19 *The PpGRS's and the FCpGRS's are equivalent.*

8 Other comparisons between the classes of relabelling systems

The aim of this section is to achieve all the comparisons between the different kinds of relabelling systems we have introduced. We first classify the two basic systems based on the relabelling of partial or induced subgraphs. Then, we will consider the two mechanisms of local control that we have defined : the priorities on the set of rules and the use of forbidden contexts. We will see that whenever we use such a local control, the differences between the powers of partial or induced relabellings disappear.

Proposition 20 *The class of pGRS's is strictly less powerful than the class of iGRS's.*

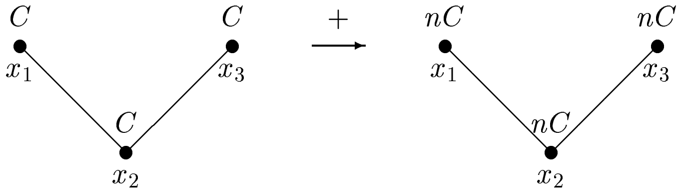
Proof. Let $\mathcal{R}$ be a given *pGRS*. The constructed *iGRS* will have the same sets of labels as $\mathcal{R}$. Each rule $(G_r, \lambda, \lambda') \in \mathcal{R}$ is simulated by the set of rules (G'_r, γ, γ') on induced subgraphs such that $\boldsymbol{v}(G'_r) = \boldsymbol{v}(G_r)$, (G_r, λ) is a subgraph of (G'_r, γ) and $\gamma'_e(\{x, y\}) = \gamma_e(\{x, y\})$ for every edge $\{x, y\} \in \boldsymbol{e}(G'_r) \setminus \boldsymbol{e}(G_r)$. Thus $\mathcal{R}$ can be simulated by a *iGRS*. □

The following example shows that the stated inclusion is strict.

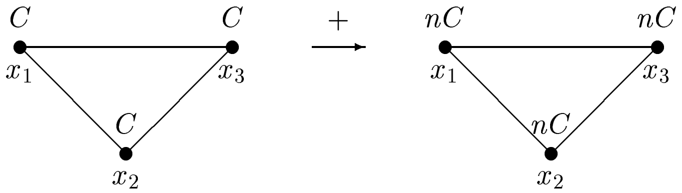
Example 21 Consider the *iGRS* previously defined in Example 1.2. The following properties hold for every graph (G, λ') in some $Irred((G, \lambda))$ where (G, λ) has only C-labels.

- an irreducible graph (G, λ') has only C-labels or only nC-labels,
- an irreducible graph (G, λ') has only C-labels if and only if G is a complete graph.

On the other hand, a *pGRS* working on initially C-labelled graphs cannot satisfy the previous properties. To see that, let us suppose that a *pGRS* $\mathcal{R}$ satisfies these properties. Then, $\mathcal{R}$ would allow the following relabelling:



A C-labelled triangle should be irreducible, but there is no way to prevent the *pGRS* $\mathcal{R}$ from applying to it the same relabelling rules as before, which gives the contradiction :



□

Proposition 22 *The class of iGRS's is strictly less powerful than the class of FCpGRS's.*

Proof. Let $\mathcal{R}$ be a given *iGRS*. Each rule $(G_r, \lambda, \lambda') \in \mathcal{R}$ can be simulated by a fc-rule given by (G_r, λ, λ') where the forbidden contexts are the graphs obtained by adding any new edge linking two vertices of $\boldsymbol{v}(G_r)$. On the other hand, the set of trees is "recognized" by a *FCpGRS* (see Example 9), but is not "recognizable" by a *iGRS*. To see that, let us assume that a *iGRS* $\mathcal{R}$' "recognizes" the set of trees, that is, $\mathcal{R}$' works on initially N-labelled graphs and is such that an irreducible graph

(G, λ) has only T-labels iff G is a tree. Hence, $\mathcal{R}$' must recognize a string (i.e. a "line-graph"). Now, consider a long enough string (whose length is greater than the greatest diameter of a rule in $\mathcal{R}$') ; if we add an edge linking the two end-points of this string, $\mathcal{R}$' will be able to apply the same relabelling rules as it did for the string, and then recognizes a ring. This leads to a contradiction, thus the stated inclusion is strict. □

As in Theorem 19, by adapting the constructed *PpGRS* to the notion of induced occurrences instead of partial ones, one can prove that:

Proposition 23 *The PiGRS's and the FCiGRS's are equivalent.*

We still have to relate the classes of *YpGRS*'s and *ZiGRS*'s, for $Y, Z \in \{P, FC\}$. To do that, we first need the following result :

Lemma 24 *There exists a PpGRS which, given a rooted spanning tree of a graph G and given a graph G_i having m vertices, makes an enumeration of all the m-tuples of vertices of G and during this enumeration recognizes the m-tuples which correspond to an induced occurrence of G_i in G.*

Sketch of proof. One takes again a *PpGRS* which enumerates all m-tuples of vertices of G (see Section 3). Then we add a rule r_i with G_i as left-hand side and a label *recognized* in the right-hand side. For any possible G_j, obtained from G_i by adding a new edge, we add a rule r_j with G_j as left-hand side and a label *unrecognized* in the right-hand side, and assign to r_j a greater priority than r_i. □

Proposition 25 *The class of FCiGRS's is less powerful than the class of PpGRS's.*

Proof. Let $\mathcal{R}$ be a *FCiGRS*. The simulation principle is the same that we have used in the proof of Proposition 17. Let $(r, \mathcal{H}_r)$ be a rule of $\mathcal{R}$, with $r = (G_r, \lambda_r, \lambda'_r)$, $\mathcal{H}_r = \{((G_i, \lambda_i), \theta_i)\}_{i \in I_r}$ and d be the greatest diameter of the context graphs G_i's. We only need to change the application test of the fc-rule r. We informally describe this new test, which replaces rules R_{52}, R_{53} and R_{54} in R_{locsim}. Let m be the number of vertices of the relabelled graph G_r of r. According to the previous lemma, during an enumeration of m-tuples, one can recognize whether the current m-tuple is an induced occurrence of (G_r, λ_r) or not. When it is the case one can test (lemma 7.5), for any $i \in I_r$, whether a n_i-tuple is an induced occurrence of G_i (n_i is the number of vertices of G_i). The fc-rule r is then applicable to the current m-tuple if and only if no n_i-tuple is an induced occurrence of G_i. □

Thus, any *PiGRS*, equivalent to a *FCiGRS*, can also be simulated by a *PpGRS*. On the other hand, by adding priorities to the sets of rules obtained by simulating, like in Proposition 20, each rule of a *PpGRS*, one proves that any *PpGRS* can be simulated by a *PiGRS*. Thus, we have the following result :

Proposition 26 *The PiGRS's and the PpGRS's are equivalent.*

Finally, we obtain the global classification — given in the introduction — of the graph relabelling systems introduced in this paper.

References

[1] M.W. Alford, J.P. Ansart, G. Hommel, L. Lamport, B. Liskov, G.P. Mullery and F.B. Schneider, *Distributed Systems*, Lecture Notes in Computer Science **190** (1985).

[2] D. Angluin, *Local and global properties in networks of processors*, Proceedings of the 12^{th} STOC (1980), 82-93.

[3] C. Berge, *Graphs and Hypergraphs*, North Holland, Amsterdam (1977).

[4] M. Billaud, P. Lafon, Y. Métivier and E. Sopena, *Graph Rewriting Systems with Priorities*, Lecture Notes in Computer Science **411** (1989), 94-106.

[5] B. Courcelle, *Recognizable sets of unrooted trees* in *Tree Automata and Languages*, M. Nivat and A. Podelski (editors), Elsevier Science Publishers (1992), 141-157.

[6] J. Dassow and G. Paun, *Regulated Rewriting in Formal Language Theory*, EATCS Series **18** (1989).

[7] A. Gibbons, *Algorithmic graph theory*, Cambridge University Press (1985).

[8] F.T. Leighton, *Finite common coverings of graphs*, J. Combin. Theory Ser. B **33** (1982), 231-238.

[9] I. Litovsky and Y. Métivier, *Computing with Graph Rewriting Systems with Priorities*, Fourth International Worshop on Graph Grammars and their Applications to Computer Science, Bremen. Lecture Notes in Computer Science **532** (1991), 549-563.

[10] I. Litovsky and Y. Métivier, *Computing trees with graph rewriting systems with priorities*, in *Tree Automata and Languages*, M. Nivat and A. Podelski (editors), Elsevier Science Publishers (1992), 115-139.

[11] I. Litovsky, Y. Métivier and W. Zielonka *The power and limitations of local computations on graphs and networks*, Proceedings of Graph-Theoretic Concepts in Computer Science (WG'92), To appear in Lecture Notes in Computer Science.

[12] A. Mazurkiewicz *Elections in planar graphs*, Internal report, University Bordeaux I 90-105 (1990).