Minor Searching, Normal Forms of Graph Relabelling: Two Applications Based on Enumerations by Graph Relabelling*

Anne Bottreau and Yves Métivier**

LaBRI, Université Bordeaux I, ENSERB
351 cours de la Libération 33405 Talence cedex FRANCE
{bottreau,metivier}@labri.u-bordeaux.fr, fax:(+33) 05 56 84 66 69

Abstract: *This paper deals with graph relabelling introduced in [LMS95]. Our first result concerns the open problem of searching a graph as a minor in a graph with a distinguished vertex, by means of graph relabellings. We give and prove a graph rewriting system which answers to this problem. Secondly we define and study normal forms of graph relabellings. We prove that any graph rewriting system can be simulated by a system in k-normal form (with an integer k depending on the original system). Proofs for both results are linked by the enumeration systems they used.*

Key-words: *Local computations, graph relabelling, enumerations, paths, minor, normal form of graph rewritings.*

Introduction

Graph rewriting systems have been introduced in [LMS95] as a suitable tool for expressing distributed algorithms on a network of communicating processors. In that model a network is considered as a labelled graph whose vertices stand for processors and edges stand for communication links. Vertex labels hold for the states of processors and edge labels for the states of communication links. A computation in a network then corresponds to a sequence of labels transformations leading to a final labelled graph. A computation step on a labelled graph consists in relabelling a connected subgraph, using a graph rewriting rule.

Given a vertex in the graph, the computation of its new state depends on its current state and on the states of its neighbours. In that way graph rewritings are an example of local computations.

Among models related to our model there are the local computations defined by Rosensthiel and al. [RFH72], Angluin [Ang80], and more recently by Yamashita and Kameda [YK96a,YK96b]. In [RFH72] a synchronous model is considered, where vertices represent identical deterministic finite automata. The basic computation step is to compute the next state of each processor according to its state and the states of its neighbours. In [Ang80] an asynchronous model is considered. A computation step means that two adjacent vertices exchange their labels and then compute new ones. In [YK96a,YK96b] an asynchronous model is studied where a basic computation step means that a processor either changes its state and then sends a message or receives a message. Our model is an asynchronous model too.

Limitations of our formalism have been discussed in [LMZ95] and [BM96]. Some graph properties have been proved to be unrecognizable by local computations.

On the other side, graph rewriting power has been studied in [LM93,LMS95]. It has more particularly concerned the definition of different classes of graph rewriting systems. Moreover authors dealed with graphs with a distinguished vertex (also called 1-graphs in [Cou90]), showing that graph rewriting were powerful on this kind of graph.

In [CM94], it has been proved that we can not decide whether or not a fixed graph is included as a minor in a given graph by means of local computations. This problem remained open for 1-graphs. In this paper we prove that searching a minor can be done by graph rewritings on 1-graphs

* This work has been supported by the EC TMR Network GETGRATS (General theory of Graph of Graph Transformation) through the University of Bordeaux.

** Member of the Institut universitaire de France.

(Theorem 1) : given a graph H, there is a graph rewriting system with priority which verifies if H is a minor of G where G is a 1-graph. We describe a system with a finite number of rules and labels depending on H. Rules number is given by a polynomial function of the edges number and the vertices number of H whereas the labels number is given by an exponential function in the number of vertices of H.

Given a positive integer k we define that a rewriting system is in k-normal form if each rule only rewrites a path of length bounded by $k-1$. In this paper we prove that graph rewriting systems with priority can be normalized in k-normal form, for a convenient integer k depending on the original system.

From any graph rewriting system $\mathcal{R}$ we use systems of enumeration so as to obtain a graph rewriting system with priority in k-normal form which has the same behaviour as $\mathcal{R}$ (Theorem 2).

The paper is organized as follows. The first section reviews the definitions related to graph rewriting. In the second part we present systems of enumeration (m-enumeration and enumeration of simple paths). The third part is devoted to the subgraph and minor searchings. Finally, in Section 4, we present the notion of k-normal form and we explain our method for the normalization of graph rewriting system.

1 Graph rewriting

All graphs considered in this paper are finite, undirected and simple (i.e. without multiple edges and self-loops). A graph G denoted $(V(G), E(G))$ is defined by a finite vertex-set and a finite edge-set. An edge with end-points v and v' is denoted $\{v, v'\}$. If v is a vertex of a graph G, the degree of v is denoted $deg_G(v)$ and the neighbourhood of v in G is denoted $N_G(v)$. The subscript G is omitted when there is no ambiguity.

1.1 Labelled graphs

Our work deals with labelled graph over an alphabet usually denoted L. A *labelled graph* over L is a couple (G, λ) where G is a connected graph, and λ is a mapping of $V(G) \cup E(G)$ in L. This function is called the *labelling function* of the graph.

Two labelled graphs are isomorphic if the underlying graphs are isomorphic and if the labellings are preserved.

An injection Θ of $V(G)$ in $V(G')$ is an *occurrence* of (G, λ) in (G', λ') if, for any vertices x and y of $V(G)$:

$$\begin{aligned} &\{x, y\} \in E(G) \implies \{\Theta(x), \Theta(y)\} \in E(G'), \\ &\lambda(x) = \lambda'(\Theta(x)), \\ &\lambda(\{x, y\}) = \lambda'(\{\Theta(x), \Theta(y)\}). \end{aligned}$$

The graph $(\Theta(G), \lambda')$ having $\Theta(V(G))$ as vertex-set and $\{\{\Theta(x), \Theta(y)\}/\{x, y\} \in E(G)\}$ as edge-set is a subgraph of (G', λ').

If the graph $(\Theta(G), \lambda')$ is an induced subgraph of (G', λ'), Θ is an *induced occurrence* of (G, λ) in (G', λ').

Let Θ be an occurrence of (G, λ) in (H, ν) and Θ' an occurrence of (G', λ') in (H, ν), Θ and Θ' are disjoint if the corresponding subgraph are disjoint, which is denoted $\Theta \cap \Theta' = \emptyset$.

1.2 Graph rewriting system

A rewriting rule r is a couple $\{(G_r, \lambda_r), (G_r, \lambda'_r)\}$ of two connected labelled graphs having the same underlying graph. Formally we define such a rule as a triplet :

Definition 1 *A* graph rewriting rule r *is a triplet* $(G_r, \lambda_r, \lambda'_r)$ *where* G_r *is a connected graph,* λ_r *the initial labelling function and* λ'_r *the final labelling function.*

A rewriting rule r is applicable to a labelled graph (G,λ) if there exists an occurrence (G_1,l_1) of (G_r,λ_r) in (G,λ). This will be denoted by $(G,\lambda) \underset{r}{\longrightarrow} (G,\lambda')$ with λ' equal to λ except on G_1 where it's equal to λ'_r.

Definition 2 *A* graph rewriting system $\mathcal{R}$ *(GRS for short) is a triplet* $\mathcal{R} = (L,I,P)$ *where* $L = L_v \cup L_e$ *is a set of labels,* $I = I_v \cup I_e$ *is the set of initial labels,* ($I_v \subseteq L_v$ *and* $I_e \subseteq L_e$*), and* P *the set of graph rewriting rules.*

If a rule r of a graph rewriting system $\mathcal{R}$ can be applied onto a labelled graph (G,λ), then we write $(G,\lambda) \underset{\mathcal{R}}{\longrightarrow} (G,\lambda')$ where λ' is equal to λ except on the rewritten part of the graph.

Consider a *GRS* $\mathcal{R} = (L,I,P)$, a labelled graph (G,λ_0) where λ_0 is a labelling function over I.

Definition 3 *A rewriting sequence of length* n*, coming from* (G,λ_0) *by means of* $\mathcal{R}$ *is defined as the sequence of labelled graphs* $(G,\lambda_i)_{0\leq i\leq n}$ *where* $\forall i, i<n$, $(G,\lambda_i) \underset{\mathcal{R}}{\longrightarrow} (G,\lambda_{i+1})$.

Our notion of rewriting sequence corresponds to a notion of *sequential* computation. We can define a distributed way of computing by saying that two consecutive relabelling steps concerning non-overlapping occurrencies may be applied in any order. Then they may be applied *concurrently*. Our notion of relabelling sequence may be regarded as a *serialization* [Maz87] of some distributed computation. This model is clearly asynchronous : several relabelling steps may be done at the same time but we do not demand all of them to be done.

Definition 4 *Given a rewriting sequence* $(G,\lambda_i)_{0\leq i\leq n}$ *and* x *in* $V(G)\cup E(G)$*, the history of* x *linked to the rewriting sequence* $(G,\lambda_i)_{0\leq i\leq n}$ *is the word* $h_n(x)$ *defined by:*

$$h_n(x) = \lambda_{i_0}(x)\lambda_{i_1}(x)\cdots\lambda_{i_j}(x).$$

with $i_0 = 0$,
$i_0 < i_1 < \cdots < i_j \leq n$
$\forall k \in \{1,\cdots,n\}$, $k \in \{i_0,\cdots,i_j\}$ *iff* x *belongs to the rewritten occurrence in the rewriting step* $(G,\lambda_{k-1}) \underset{\mathcal{R}}{\longrightarrow} (G,\lambda_k)$

Given a graph rewriting system $\mathcal{R}$, the reflexive and transitive closure of $\mathcal{R}$ is denoted $\overset{*}{\underset{\mathcal{R}}{\longrightarrow}}$.

Definition 5 *An* irreductible graph with respect to a *GRS* $\mathcal{R}$ *is a labelled graph to which no rule is applicable.*

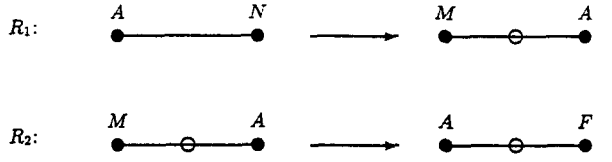
Given a labelled graph (G,λ) over I, we denote $Irred_{\mathcal{R}}((G,\lambda))$ the set of irreductible graphs coming from (G,λ):
$Irred_{\mathcal{R}}((G,\lambda)) = \{(G,\lambda') \,/\, (G,\lambda) \overset{*}{\underset{\mathcal{R}}{\longrightarrow}} (G,\lambda')$ and (G,λ') irreductible with respect to $\mathcal{R}\}$.

Definition 6 *A GRS* $\mathcal{R} = (L,I,P)$ *is called* noetherian *if there doesn't exist any infinite rewriting sequence coming from a graph labelled over* I.

A graph rewriting system where the set of rules is given with a partial order is called a *graph rewriting system with priority* (*PGRS* for short).
The partial order defined on the set of rules is denoted $<$, the applicability of the rules of such a system is defined in the following way.

Let $\mathcal{R}$ be a *PGRS*, r a rule of this system, and (G,λ) a labelled graph. The rule r is applicable to an occurrence Θ of (G_r,λ_r) in (G,λ) if there doesn't exist in (G,λ) any occurrence Θ' of $(G_{r'},\lambda_{r'})$ with $r' > r$ which overlaps Θ.

Example 1 Let us consider the following *PGRS* with two rules.



With, $R_1 > R_2$.

The order defined on the set of rules has the following meaning : the rule R_2 is applicable to an occurrence Θ if and only if there is no occurrence for R_1 overlapping Θ.

This system labels vertices and edges in order to form a spanning tree.

A graph rewriting rule with forbidden contexts is a pair $(r, \mathcal{H}_r)$ where r is a rewriting rule $(G_r, \lambda_r, \lambda'_r)$ and $\mathcal{H}_r$ is a finite family of pairs $\{((G_i, \lambda_i), \Theta_i)\}_{i \in I_r}$ with (G_i, λ_i) a labelled graph (called forbidden context) and Θ_i an occurrence of (G_r, λ_r) in (G_i, λ_i). The forbidden contexts of such a rule are used as follows :

Let $(r, \mathcal{H}_r)$ be a graph rewriting rule with fobidden contexts, let Θ be an occurrence of (G_r, λ_r) in a graph (G, λ). The rule $(r, \mathcal{H}_r)$ is applicable to Θ if there doesn't exist, for no i, an occurrence Φ_i of (G_i, λ_i) such that $\Phi_i \Theta_i = \Theta$.

Such rules define *graph rewriting system with forbidden contexts* (*FCGRS* for short).

2 Some enumeration's problems solved by graph rewriting systems

Several graph rewriting systems exist for the computation of a spanning tree on a labelled graph with a distinguished vertex. Such a computation is done thanks to labelling. A set of edges is labelled so that it forms a spanning tree of the graph in which the root is the distinguished vertex. Given such a labelled graph, there exists a graph rewriting system with priority which allows depth-first traversals of the tree. Such a *PGRS* has been introduced in [LMS95].

In this section we recall a well-known *PGRS* for the enumeration of m-tuples of vertices and we introduce a new *PGRS* for the enumeration of simple paths. These graph rewriting systems use a *PGRS* for the computation of a spanning tree which we call $\mathcal{R}_{span}$ and a *PGRS* for the traversal of a tree which we call $\mathcal{R}_{trav}$.

2.1 m-enumeration

In [LMS95], it was proved that enumerating all the m-tuples of vertices of a labelled graph can be done by means of a graph rewriting system. Without going into further details, we recall how this system runs.

We consider labelled graphs with a distinguished vertex. Firstly, $\mathcal{R}_{span}$ is used on such a labelled graph in order to obtain a spanning tree (by labelling). This enumeration uses m traversals of the spanning tree in order to obtain a m-tuple $(x_1, x_2, \cdots, x_m)$. Then, given a m-tuple $(x_1, x_2, \cdots, x_m)$, a new traversal is started so as to obtain a new m-tuple $(x_1, x_2, \cdots, y)$ with $y \neq x_m$. This process is repeated until we can't find any vertex y for this last position. Then we start new traversals by changing the two last vertices of the m-tuple, and so on until there is no vertex to be the first vertex of a m-tuple.

Thus this graph rewriting system is based on the system $\mathcal{R}_{trav}$. The labels of the enumeration system are made up of three components :

- a label issued from the traversal system.
- a label of the set $\{Search, Return, Reset, Stop\}$ with the following meaning :
  - **Search** : a vertex is searched.
  - **Return** : a vertex has been found.
  - **Reset** : the current m-tuple is modified.
  - **Stop** : the enumeration is done.
- a m-tuple of labels such that the label in position i gives an information about the position of the vertex in the current m-tuple. There are three different values :
  - **0** : the vertex is not the i^{th} vertex of the current m-tuple.
  - **1** : the vertex is the i^{th} vertex of the current m-tuple.
  - $\overline{\mathbf{1}}$: the vertex was the i^{th} vertex of all the m-tuple having the same first $i-1^{th}$ components.

The system has a finite number of rules ($\#rules_{enum} = O(m)$) and a finite number of labels ($\#labels_{enum} = O(m * 2^m)$).

2.2 Enumeration of simple paths

In a connected labelled graph, we consider the simple paths coming from a source vertex to a target vertex. Our aim is to enumerate all these simple paths by means of graph rewritings. To this end, we encode a graph rewriting system which labels these paths one by one. Each path is encoded by labels on its vertices and edges. We consider that the source vertex is labelled *Search* and the target vertex is labelled *Ending*.

Description

We work on a connected labelled graph G. We denote by I the *Search*-labelled vertex of G. We denote by J the *Ending*-labelled vertex. At the beginning, no edge is labelled.

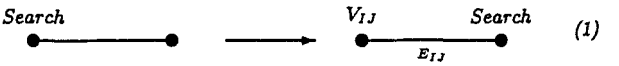
We start on I. We mark a simple path from I to J, by labelling the edges and the vertices used in the path (the labels are E_{IJ} and V_{IJ}). When we have a path, backtracking is used in order to change the last edge and to look for a new path. So we keep the same prefix of the path, we just change the last edge. We go on until we have tried all the possibilities from the vertex I.

Let us now describe a graph rewriting system encoding this algorithm.

Let $Y \in \{Ending, \epsilon\}$ where ϵ design the empty word that is to say "no label".

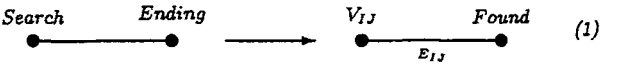
Graph rewriting system with priority $\mathcal{R}_{enum,\mathcal{P}}(I,J)$ ▼

The first rule allows the traversal to go on. We label the vertex and the edge which we put in the path we are building.

R_{E1}: 

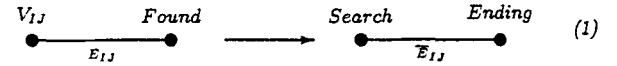
(1)

If we reach the Ending-labelled vertex, then we have found a simple path coming from I :

R_{E2}: 

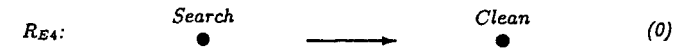
(1)

As we have a simple path, we use the backtracking in order to search another path. We label this edge with $\overline{E}_{IJ}$ so that we won't use it in a new path with the same prefix.

R_{E3}: 

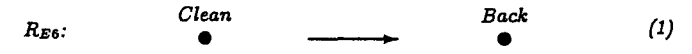
(1)

If there are no unlabelled edges incident to the Search-labelled vertex, then there are no paths with this prefix anymore. We have to change this prefix :



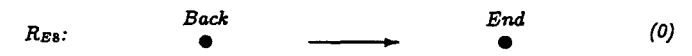
We erase the labels $\overline{E}_{IJ}$ from the edges incident to the Clean-labelled vertex :

When the cleaning mode is done, we start a backtrack :



We go back from the vertex labelled Back (the edge labelled E_{IJ} and incident to the Back-labelled vertex changes its label). Then we start a new search of path :

When we can no longer backtrack, the enumeration is done.



▲

Invariants and properties

Let G be a connected graph. The initial labelling function of G λ_0 is defined by :

$$\lambda_0(I) = Search$$
$$\lambda_0(J) = Ending$$
$$\forall x \in V(G) \cup E(G) \setminus \{I_0\} \cup \{J\},\ \lambda_0(x) = \epsilon.$$

Let L be the set of labels :

$$L = \{Search, Found, Ending, Clean, Back, End\}.$$

We say that L is the set of active labels.

Let A be the set of labels of the whole system :

$$A = \{\epsilon, V_{IJ}, E_{IJ}, \overline{E}_{IJ}\} \cup L.$$

From now on we consider a connected graph G with an initial labelling function λ_0 (as we defined it before). We consider a rewriting sequence $(G_i)_{i\geq 0}$[1] obtained by the application of $\mathcal{R}_{enum,\mathcal{P}}$ on (G, λ_0).

In order to prove the ending and the validity of our system, we give some properties of $\mathcal{R}_{enum,\mathcal{P}}$. The easy proofs of the four following invariants will be omitted.

[1] G_i stands for (G, λ_i).

Invariant 1 *$\forall i \geq 0$, there exists only one vertex x in G_i such that $\lambda_i(x) \in L \setminus \{Ending\}$.*

We denote this vertex x_L.

Each unlabelled edge can not receive E_{IJ} as a label if its end-points are labelled V_{IJ} (R_{E1}).

Invariant 2 *$\forall i \geq 0$, the set of edges labelled E_{IJ} in G_i forms a simple path from I to x_L.*

We denote this simple path $C_i(x_L)$.

Invariant 3 *$\forall i \geq 0$, any vertex x labelled V_{IJ} by λ_i is on the path $C_i(x_L)$.*

Invariant 4 *$\forall i \geq 0$, let a be an edge of G such that $\lambda_i(a) = \overline{E}_{IJ}$. The edge a is incident to only one vertex of $C_i(x_L)$.*

We denote this vertex by $\overline{x}_a$ and we denote by $\overline{C}_i(a)$ the prefix of $C_i(x_L)$ from I to $\overline{x}_a$.

Let P be a simple path, e be an edge incident to an end-point of P, $(P.e)$ denotes the path obtained by extending P by the edge e.

Invariant 5 *Let $i \geq 0$, let a be an edge of G with $\lambda_i(a) = \overline{E}_{IJ}$. $\forall k \geq i$, one of the following propositions is true :*

(i) *$\lambda_k(a) = \overline{E}_{IJ}$;*
(ii) *$\lambda_k(a) \neq \overline{E}_{IJ}$, and there is a vertex x of $\overline{C}_i(a)$ such that $\lambda_k(x) \in \{Clean, Back, End\}$;*
(iii) *$\lambda_k(a) \neq \overline{E}_{IJ}$, $\overline{C}_i(a)$ is no longer a prefix of $C_k(x_L)$ and there is an edge b of G such that $\lambda_k(b) = \overline{E}_{IJ}$ and such that $(\overline{C}_k(b).b)$ is a prefix of $\overline{C}_i(a)$.*

Proof Proof is rather technical and not detailed there. We use an induction on k, starting with $k = i$. □

Consider a vertex x, labelled *Search* after i steps of rewriting, then the vertices, which are labelled V_{IJ}, are not concerned by the rewritings until x is not labelled *Back*. The history of x, $h_i(x)$, concerning the sequence of rewriting of length i, is the prefix of all the histories of x concerning any sequence of length j, for $j > i$. We denote $h_j(x) = h_i(x)m_{i,j}(x)$, and we state that :

Property 1 *Let $x \in V(G)$ and $i \geq 0$ such that $h_i(x)$ is ending by Search. For any vertex x' of $V(G)$ which has a history $h_i(x')$ ending by V_{IJ}, and for all j, $j > i$, such that $m_{i,j}(x)$ doesn't contain Back, the vertex x' keeps the same history : $h_j(x') = h_i(x')$.*

We denote by $S(G_i, x)$ the subgraph of G_i induced by the vertices labelled ϵ or *Ending* which are connected to x by simple paths made of unlabelled edges. This connected subgraph contains x.

Lemma 1 *For any vertex x of G, for any positive integer i such that $\lambda_i(x) = Search$,there is j, $j > i$, such that the three following propositions are true :*

i) *$\lambda_j(x) = Clean$ and $m_{i,j}(x)$ doesn't contain Back.*
ii) *The subgraphs $S(G_i, x)$ and $S(G_j, x)$ are isomorphic.*
iii) *The rewriting sequence from G_i to G_j allowed to enumerate all the simple paths of $S(G_j, x)$ starting at x and finishing on the vertex labelled Ending if this vertex is in the subgraph.*

Proof By induction on the number of edges of G. □

Proposition 1 *The graph rewriting system $\mathcal{R}_{enum,\mathcal{P}}$ is noetherian for any connected graph G given with an initial labelling function λ_0 as it has been previously defined.*

Proof Consider a connected graph G, with an initial labelling function λ_0 such that :

$$\exists x \in V(G), \lambda_0(x) = Search.$$

Lemma 1 is applicable to G with x and the initial labelling : $\exists j > 0$ such that $\lambda_j(x) = Clean$, $m_{0,j}(x)$ doesn't contain $Back$, and such that the subgraphs $S(G_0, x)$ and $S(G_j, x)$ are equal. On G_j, we can apply the rule R_{E6}, R_{E7} and then R_{E8}, and we have after these two steps of rewriting : $\lambda_{j+2}(x) = End$, and $\forall y \in (V(G) \setminus \{x\}) \cup E(G), \lambda_{j+2}(y) = \epsilon$. Eventually, no more rules are applicable to G_{j+2}. □

Proposition 2 *On any connected graph G given with an initial labelling function λ_0 such that one vertex is Search-labelled and another one is Ending-labelled, the system $\mathcal{R}_{enum,\mathcal{P}}$ enumerates all the simple paths having these two singular vertices as end-points.*

Proof The proof directly comes from the Lemma 1 applied to the graph G with the labelling function λ_0. □

Our system $\mathcal{R}_{enum,\mathcal{P}}$ has a constant number of rules and a constant number of labels.

3 Subgraph and minor searching

In the previous section we introduced two systems encoding two different kinds of enumeration. Our purpose is now to present a first application of these two systems :

The m-enumeration is used so as to verify if a connected labelled graph contains a connected labelled graph H as a subgraph.

The enumeration of simple paths is used in order to verify if a connected labelled graph contains a connected labelled graph H as a minor.

3.1 Subgraph searching

We consider a connected labelled graph H with m vertices. We know that we are able to enumerate all the m-tuples of vertices of any graph G with an appropriate labelling function, thanks to a graph rewriting system with priority. Given a m-tuple of vertices of G, it's rather easy to associate each vertex to a vertex of H. Thus, we just have to check if this mapping is a good one.

Our graph rewriting system works into two parts of computation :

First part It consists in enumerating all the m-tuples of vertices of G. So, we use the *PGRS* defined in [LMS95] $\mathcal{R}_{enum}$. When a m-tuple is found (we use a label $Found_m$ when we find the last vertex of the m-tuple), the second part has to start. If we can't find H as subgraph thanks to this m-tuple, then we have to change it i.e. to resume the m-enumeration. If the end of the m-enumeration is reached, then H isn't a subgraph of G.

Second part It consists in checking that the mapping of the vertex-set $V(H)$ into the m-tuple of G is an isomorphism between H and a subgraph of G having the m-tuple as vertex-set. Let us describe how we solve this problem by means of a graph rewriting system $\mathcal{R}_{const}$.

First, we use a graph traversal to label the jth vertex of the m-tuple with the degree of the jth vertex of H. Then, using another graph traversal, we just have to check if for any edge $\{i, j\}$ in H there is in G an edge linking the jth and ith vertices of the m-tuple. Then we use another graph traversal in order to verify if every edges have been found (partial subgraph) and if there isn't any other edge between vertices of the m-tuple (induced subgraph). Thus at the end of such a traversal, either the last vertex of the m-tuple is labelled $Fail$ or the root of the spanning tree is labelled Win. In the first case, the m-enumeration has to resume. In the second case, the rewriting has to be stopped.

These parts are realized by means of graph rewriting systems with priority. Our general system, called $\mathcal{R}_{subgraph}$, is the result of the composition of $\mathcal{R}_{enum}$ [LMS95] (with a weak modification), and $\mathcal{R}_{const}$ introduced and proved in [Bot97].

For the sake of brevity we shan't give this system in details. For such a composition we use couples of labels. The first component concerns the m-enumeration. The second component concerns the subgraph's checking. We consider that such a system works on a labelled graph with a distinguished vertex (with a labelling function issued from $\mathcal{R}_{span}$).

In order to prove the termination and the validity of $\mathcal{R}_{subgraph}$, we use the fact that each part is noetherian and valid. Moreover the rules used in this system are very simple (the left-hand-side are isomorphic to a single vertex or a single edge). Therefore we state that :

Proposition 3 *Given a connected labelled graph H, the graph rewriting system with priority $\mathcal{R}_{subgraph}$ allows to check on any connected labelled graph with a distinguished vertex if H is one of its subgraph (partial or induced).*

Our graph rewriting system $\mathcal{R}_{subgraph}$ has a finite number of rules depending of the number of rules of $\mathcal{R}_{enum}$ and linearly depending on m^2 where m is the vertices number of H : $\#rules_{subgraph} = O(m^2)$. The number of labels depends (linearly) on $\#labels_{enum}$ and m : $\#labels_{subgraph} = \#labels_{enum} = O(m * 2^m)$.

3.2 Minor searching

Thanks to the notion of model defined in [RS95], we are able to prove the following equivalence :

Lemma 2 *Given two connected graphs H and G, the following statements are equivalent :*

- *H is a minor of G ;*
- *There exists a model Φ from H onto G defined by :*
  - *for any edge e of H, $\Phi(e)$ is an edge of G;*
  - *for any vertex u of H, $\Phi(u)$ is a connected partial subgraph of G (non empty).*

  The model Φ has the following properties :
  1) *for any u and v of $V(H)$, the intersection of $\Phi(u)$ by $\Phi(v)$ is empty;*
  2) *for any $e \in E(H)$, for any $u \in V(H)$, the edge $\Phi(e)$ doesn't belong to the partial subgraph $\Phi(u)$;*
  3) *Let $e = \{u, v\}$ be an edge of H, then $\Phi(e)$ has an end-point in $V(\Phi(u))$ and the other in $V(\Phi(v))$.*
- *There exists an injection γ from $V(H)$ to $V(G)$ such that for any edge $\{u, v\}$ of H, there is a simple path in G between $\gamma(u)$ and $\gamma(v)$, denoted $P(\gamma(u), \gamma(v))$. Moreover these paths are said to be valid i.e. they verify the following properties :*
  1) *For any edges $\{a, b\}$ and $\{c, d\}$ of H, with disjoint end-points, the paths $P(\gamma(a), \gamma(b))$ and $P(\gamma(c), \gamma(d))$ are vertex-disjoint.*
  2) *For any edge $\{a, b\} \in E(H)$, the path $P(\gamma(a), \gamma(b))$ has at least one edge that is disjoint from any other path $P(\gamma(c), \gamma(d))$ for $\{c, d\} \in E(H)$. Such kind of edge is called own edge.*

We present a graph rewriting system based on the fact that a minor of graph can be defined thanks to particular simple paths. Such simple paths (as defined in our lemma 2) will now be called *valid simple paths*.

Explanations The connected labelled graph H is known. We assume that we perfectly know its vertex-set $\mathcal{V}$ and its edge-set $\mathcal{E}$. Let m be the number of vertices of H. We assume that $\mathcal{V} = \{1, 2, 3, \cdots, m\}$. The edges are denoted $\{i, j\}$ with $i < j$. Thus an order is defined on $\mathcal{E}$: $\{i, j\} < \{l, k\}$ iff $(i, j) <_2 (l, k)$ (i.e $i < l$ or $i = l$ and $j < k$). We denote by $succ(i, j)$ the successor of $\{i, j\}$ and $pred(i, j)$ the predecessor of $\{i, j\}$ according to $<$. We consider that $succ(i, j) = \{i, j\}$ if it is the greatest edge in $\mathcal{E}$ (denoted $max(i, j)$), $pred(i, j) = \{i, j\}$ if it is the smallest one (denoted $min(i, j)$).

The whole system consists of a part of m-enumeration and a part of research of valid paths linking vertices of the m-tuple. We explain the algorithm we used for the second part.

The computation starts on a graph G with a m-tuple $(x_1, x_2, \cdots, x_m)$. For any edge $\{i, j\}$ of H, we mark in G a valid simple path between the vertices x_i and x_j (starting with the smallest edge). The construction of valid paths is made with the enumeration of simple paths (with a checking of validity) and also backtracking. At the end of this computation, we have two possibilities. If we have found all the valid simple paths, then H is a minor of G. If we haven't succeeded with the current m-tuple, then it means that we have to change the m-tuple i.e. to resume the m-enumeration. If the m-enumeration is done, then H isn't minor of G.

Valid paths We are able to mark simple paths thanks to the system $\mathcal{R}_{enum,\mathcal{P}}$. In order to mark a simple path concerning the jth and ith vertices of the m-tuple, we use this previous system with parameter (I, J). We have to check that :

- For any couple of vertices (L, K) disjoint from (I, J), any vertex labelled V_{LK} mustn't be labelled V_{IJ} by $\mathcal{R}_{enum,\mathcal{P}}(I, J)$. It must be the same for the edges.
- Given a path from I to J, there is at least one edge uniquely labelled with E_{IJ}.

The first condition is easy to realize, we just have to change the two first rules to prevent the labelling. The second one is done by means of a traversal of the simple path in order to check that this path contains at least one own edge, and that all the other valid paths are still valid.

The new graph rewriting system obtained is denoted $\mathcal{R}_{EV}(i, j)$ for the edge $\{i, j\}$. Such a system is made up of traversals based on a spanning tree.

Sum up The graph rewriting system $\mathcal{R}_{minor}$ consists of the following systems with the following priorities :

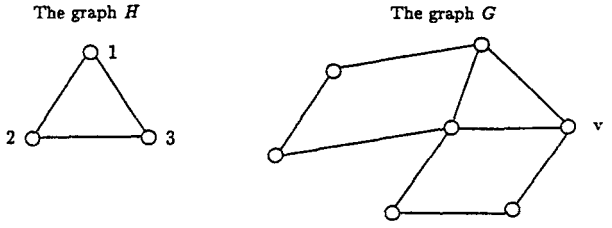
$$\mathcal{R}_{enum} > \mathcal{R}_{init} > \mathcal{R}_{EV}(i,j)_{min} > \cdots > \mathcal{R}_{EV}(i,j)_{max}.$$

With,

- $\mathcal{R}_{enum}$, enumeration of m-tuples in G ;
- $\mathcal{R}_{init}$, beginning of the second part ;
- $\mathcal{R}_{EV}(i, j)$, system of enumeration of valid simple path between the vertices i and j in the current m-tuple. These system are made by the system of enumeration of simple paths, a part for the checking of validity, and optionally a part for acknowledgment sending (for $\{i, j\}$ different from the minimal edge) and cleaning (if $\{i, j\}$ is the minimal edge).

We show on the following example how we use acknowledgment in order to compute valid simple paths according to the order $<$.

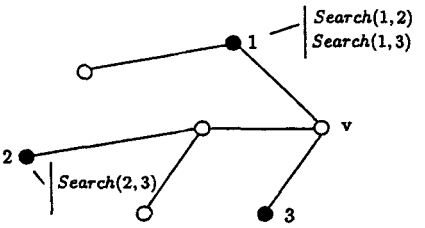
Example 2 Consider the following graphs H and G. The graph H has three vertices and three edges : $\{1,2\} < \{1,3\} < \{2,3\}$. The graph G has a distinguished vertex called v, which is the root of a spanning tree (denoted $T(G)$) computed by a graph rewriting system.



Given a 3-tuple of vertices labelled on G, we start the construction of valid simple paths for the three edges of H. Firstly, a traversal of the spanning tree is used to label the vertex 1 and 2 by the list of $Search$ labels.

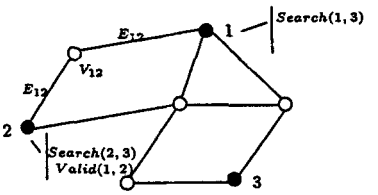
This part is done by the rules of the system $\mathcal{R}_{init}$.

The spanning tree $T(G)$ rooted in v.

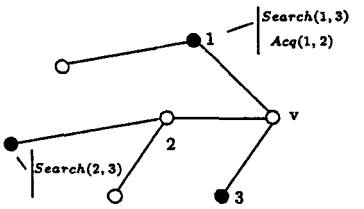


The smallest edge of H related to $<$ is the edge $\{1,2\}$. Computations start now by the labelling of a valid simple path for this edge, thanks to a system $\mathcal{R}_{EV}(1,2)$. As this is the smallest edge, we haven't to wait for an acknowledgment.

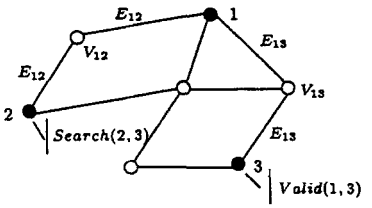
The following picture shows a computation leading to a valid path : the vertex 2 receives a label of success $Valid(1,2)$.



As a valid path has been found, an acknowledgment is sent to the vertex 1, smallest end-point of the next edge. A traversal of the spanning tree is used.

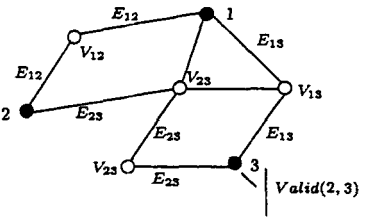


The vertex 1 has got labels $Search(1,3)$ and $Acq(1,2)$: rules of $\mathcal{R}_{EV}(1,3)$ are thus applicable and the enumeration of valid simple paths for this couple of vertices can start.



In this example, a valid simple path has been found for the couple $(1,3)$. Thus, an acknowledgment is sent to the vertex 2 (smallest end-point of the next edge). This is done by a traversal. The rules of $\mathcal{R}_{EV}(2,3)$ become applicable to the graph because of this acknowledgment. If a valid simple path is found for this couple, then the computation stops (i.e. no more rules are applicable) : H is a minor of G. In the case where no valid simple path exists, the enumeration of valid simple path is resumed for the previous couple $(1,3)$, and so on.

The last picture gives us successful computations showing that H is a minor of G.



Details about this graph rewriting system can be found in [Bot97]. We recall that h denotes the number of edges of H. The number of rules of $\mathcal{R}_{minor}$ is a linear function of $\#rules_{enum}$, m and h^2 : $\#rules_{minor} = O(h^2 + m)$. The number of labels is a linear function of $\#labels_{enum}$, h and h^2 : $\#labels_{minor} = O(m * 2^m + h^2)$.

The system $\mathcal{R}_{minor}$ satisfies the following theorem :

Theorem 1 *Given a connected labelled graph H, there exists a graph rewriting system with priority which allows to check onto any connected labelled graph G with a distinguished vertex if H is a minor of G.*

Thus, given a family of graphs defined with a finite set of forbidden minors, there exists a graph rewriting system with priority which verifies if a given graph with a distinguished vertex belongs to the family. The forbidden minors must be known. We just have to compose a set of systems $\mathcal{R}_{minor}$ corresponding to the forbidden minors.

Corollary 1 *Let $\mathcal{F}$ be a family of connected graphs, defined by a finite set of forbidden minors. We can check by means of a graph rewriting system if a connected graph G with a distinguished vertex belongs to $\mathcal{F}$.*

Therefore we are able to give a graph rewriting system with priority which verifies if a labelled graph with a distinguished vertex is planar or not.

4 Normal forms for graph rewriting systems

In this part we introduce different kinds of normal forms for graph rewritings and more particularly the k-normal form of graph rewriting. Then we prove that for any graph rewriting system there exists a *PGRS* in k-normal form equivalent to the original system : any *GRS* can be normalized according to the k-normal form. Our method consists in building the *PGRS* in k-normal form using systems of enumeration.

4.1 Definitions

We are interested in the structure of the subgraphs which are rewritten by the rules of our systems.

As a first normal form we consider the case where the left-hand sides of the rules are isomorphic to a vertex or an edge :

Definition 7 *A graph rewriting system has a* 2-normal form *if each rule rewrites one vertex or one edge and the two incident vertices.*

Most of our graph rewriting systems are in 2-normal form. The computation of a spanning tree, the traversal of a tree, and of course the subgraph searching can be done thanks to graph rewriting systems in 2-normal form.

We can also consider that the left-hand sides are equal to simple paths of bounded length.

Definition 8 *A graph rewriting system has a* k-normal form *if each rule rewrites a simple path of length bounded by* $k-1$.

4.2 Simulation of a *FCGRS* by a *PGRS* in k-normal form

We want to prove that any *GRS* without normal form can be simulated by a *GRS* in 2-normal or k-normal form. To this end, we use the method introduced in [LMS95] to simulate any *FCGRS* by a *PGRS*. In a first part we recall this method, and then we provide our application.

Method for the simulation of a *FCGRS* by a *PGRS*

This method is made up of three steps.

I The first part concerns the partition of the initial graph into subgraphs of k-bounded diameter where k is the maximal diameter of the graphs in the rules of the *FCGRS*. This part is called the k-election. The k-election problem (introduced in [LMS95]) can be explained as follows. Each vertex of the graph stands for a *town*. We want to organize the graph by delimiting *countries*, each country having one *capital*. In each country the distance between town and the capital must at most be k. Moreover, the distance between two different capitals in the graph must be at least $k+1$. This part is done by a *PGRS* in $(2k+1)$-normal form.

II The second part consists in supervising the activity of the capitals. If a capital is active, it means that we can simulate on its country the application of a rule of the system. This part is done by a *PGRS* in $k+1$-normal form.

III The third part consists in simulating the application of the rules on a country having an active capital. This part is called the **local simulation**. We have to adapt this local simulation to our problem.

Application to the k-normal form

We are able to realize a local simulation by a *PGRS* in 2-normal form.
We consider we are working on a country with an active capital.

1. Using a tree traversal, towns are activated one by one ($\mathcal{R}_{trav}$).
2. Given an active town, we construct a spanning tree of the ball of center the active town and of radius k ($\mathcal{R}_{span}(k)$ with orientation from the root to the leaves).
3. For each rule r with forbidden context, we make a system $\mathcal{R}_r$ so as to test the applicability of r on the ball of radius k. We now explain this part of the simulation :
   (a) We look for a subgraph isomorphic to (G_r, λ_r) in the ball of center this town and of radius k. We can do that by means of $\mathcal{R}_{subgraph}$. Then in G, some vertices have label $(1_i, x)$ and some edges have label (p, x) where x is a symbol holding for the label issued from λ_r. These vertices and edges form a subgraph isomorphic to (G_r, λ_r). The values of i are in $\{1, \cdots, |V(G_r)|\}$.
   (b) Then, given an occurrence of (G_r, λ_r), we search all the forbidden contexts using one *PGRS* $\mathcal{R}_{subgraph}$ by context.
   (c) If we find such a forbidden context, then we resume the searching of another occurrence.

(d) If there aren't any forbidden contexts, then we have to apply the rule r by changing the labels of the edges and then of the vertices. In this way we will realize a rewriting in 2-normal form. Let us now introduce the system $\mathcal{R}_{norme}$ in 2-normal form. We consider we are working on a connected graph having a labelled spanning tree (one vertex is labelled *Edge*, the others N_0). Some vertices and edges have labels coming from λ_r (as explained before). A first traversal is done in order to change the label for the edges (p, x), a second traversal deals with the vertices. The symbol x' means the label issued from λ'_r.

System $\mathcal{R}_{norme}$

We walk on a branch of the tree (by using edges of the tree).

R_1: 

(3)

If we meet a vertex labelled 1_l, then we change the labels of all the edges incident to this vertex. Edges could be edges of the spanning tree, we don't specify it in our rule.

$k > l$, $R_2(l,k)$: 

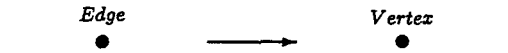
(2)

When we reach a leaf or when there is nothing else to do, then we come back in the tree.

R_3: 

(1)

When we are on the root of the tree, then we start a new traversal in order to rewrite the vertices.

R_4: 

(0)

We advance on a branch of the tree.

R_5: 

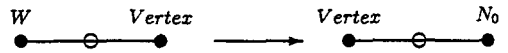
(3)

When we reach a vertex which is an image of a vertex of $V(G_r)$, then we change its label.

R_6: 

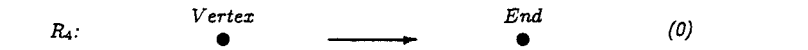
(2)

The traversal goes on by going back to the root.

R_7: 

(1)

When we reach the root, then the computation is done.



This graph rewriting system comes from the traversal of a tree. A system for tree traversals has been proved to be noetherian and valid in [LMS95]. Thus our system is noetherian and valid because we are sure to reach all the vertices and the edges we have to rewrite.

For our simulation we use graph rewriting systems in $k+1$-normal form and systems in 2-normal form. The k-election problem and the computation of a spanning tree of a ball of radius k are realized by graph rewriting systems in $k+1$-normal form (in respect of our notation).

Proposition 4 *Any graph rewriting system with forbidden context can be simulated by a graph rewriting system with priority which is in $k+1$-normal form.*

Moreover any graph rewriting system with priority can be moved into a graph rewriting system with forbidden context as it is explained in [LMS95]. Thus,

Theorem 2 *Any graph rewriting system (with priority or forbidden context) can be normalized into a graph rewriting system with priority in k-normal form with a convenient integer k.*

References

[Ang80] D. Angluin. Local and global properties in networks of processors. In 12^{th} *STOC*, pages 82–93, 1980.

[BM96] A. Bottreau and Y. Métivier. Kronecker product and local computation in graphs. In *CAAP'96*, volume 1059 of *Lect. Notes in Comp. Sci.*, pages 2–16, 1996.

[Bot97] A. Bottreau. *Réécritures de graphe et calculs distribués.* PhD thesis, Université Bordeaux I, LaBRI, juin 1997.

[CM94] B. Courcelle and Y. Métivier. Coverings and minors : Application to local computations in graphs. *Europ. J. Combinatorics*, 15:127–138, 1994.

[Cou90] B. Courcelle. The monadic second order logic of graphs i. recognizable sets of finite graphs. *Inform. and Comput.*, 85:12–75, 1990.

[LM92] I. Litovsky and Y. Métivier. Computing trees with graph rewriting systems with priorities. *Tree Automata and Languages*, pages 115–139, 1992.

[LM93] I. Litovsky and Y. Métivier. Computing with graph rewriting systems with priorities. *Theoretical Computer Science*, 115:191–224, 1993.

[LMS95] I. Litovsky, Y. Métivier, and E. Sopena. Different local controls for graph relabelling systems. *Mathematical Systems Theory*, 28:41–65, 1995.

[LMZ95] I. Litovsky, Y. Métivier, and W. Zielonka. On the recognition of families of graphs with local computations. *Information and computation*, 115(1):110–119, 1995.

[Maz87] A. Mazurkiewicz. *Petri nets, applications and relationship to other models of concurrency*, volume 255, chapter Trace Theory, pages 279–324. W. Brauer et al., 1987.

[RFH72] P. Rosensthiel, J.R. Fiksel, and A. Holliger. Intelligent graphs : networks of finite automata capable of solving graph problems. In *Graph Theory and Computing*, pages 219–265. Academic Press, 1972.

[RS95] N. Robertson and P.D. Seymour. Graph minors xiii. the disjoint paths problem. *Journal of combinatorial theory, Series B*, 63:65–110, 1995.

[YK96a] M. Yamashita and T. Kameda. Computing on anonymous networks: Part i - characterizing the solvable cases. *IEEE Transactions on parallel and distributed systems*, 7(1):69–89, 1996.

[YK96b] M. Yamashita and T. Kameda. Computing on anonymous networks: Part ii - decision and membership problems. *IEEE Transactions on parallel and distributed systems*, 7(1):90–96, 1996.