LaBRI Research Report No. Compiled on April 11, 2001
The final version of this paper has been published in *Computers and Artificial Intelligence* **16-2** (1997), 167–185.

GRAPH RELABELLING SYSTEMS: A GENERAL OVERVIEW [1]

Abstract. Graph relabelling systems have been introduced as a suitable model for expressing and studying distributed algorithms on a network of communicating processors. We recall the basic ideas underlying that model and we survey the main questions that have been considered and the main results that have been obtained in that framework.

Keywords. Distributed algorithm, Election, k-covering, Recognition, Local computations in graphs, Graph relabelling system.

1 Introduction

Graph relabelling systems have been introduced in [2] as a suitable tool for expressing distributed algorithms on a network of communicating processors. In that model a network is regarded as a labelled graph whose vertices stand for processors and edges stand for communication links. Vertex labels are used for describing the states of processors and edge labels for describing the states of communication links. A computation in a network then corresponds to a sequence of labels transformations leading to a final labelled graph considered as the result of the computation. Every elementary step in such a computation will be described by a *graph relabelling rule* expressing the modification of the corresponding labels.

Let us first illustrate this idea by considering a simple distributed algorithm which computes a spanning tree of a network. Assume that a unique given processor is in an "active" state (encoded by the label **A**), all other processors being in some "neutral" state (label **N**) and that all links are in some "passive" state (label **0**). The tree initially contains the unique active vertex. At any step of the computation, an active vertex may activate one of its neutral neighbours and mark the corresponding link which gets the new label **1**. This computation stops as soon as all the processors have been activated. The spanning tree is then obtained by considering all the links with label **1**. Fig. 1 describes a sample computation using this algorithm.

An elementary step in this computation may be depicted as a *relabelling step* by means of the following relabelling rule R which describes the corresponding label modifications:

$$R: \quad \overset{\mathbf{A}}{\bullet}\!\overset{\mathbf{0}}{\text{———}}\!\overset{\mathbf{N}}{\bullet} \quad \longrightarrow \quad \overset{\mathbf{A}}{\bullet}\!\overset{\mathbf{1}}{\text{———}}\!\overset{\mathbf{A}}{\bullet}$$

An application of this relabelling rule on a given graph (or network) consists in (i) finding in the graph a subgraph isomorphic to the left-hand-side of the rule (this subgraph is called the *occurrence* of the rule) and (ii) modifying its labels according to the right-hand-side of the rule.

The *relabelling sequence* depicted in Fig. 1 illustrates a *sequential* computation since the relabelling steps are sequentially applied. A *distributed* view of this computation can be obtained by considering that relabelling steps concerning *disjoint* parts of the graph may be applied in any order, or even concurrently (this is namely the case for the steps (2) and (3), or (4) and (5) in Fig. 1).

Among models related to those used here are computations defined by Fiksel *et al.* [6] and by Angluin [1]. The first one considers a synchronous model based on graphs equipped with identical

[1]Most of the work reported here has been supported by the Esprit Basic Research Working Group COMPUGRAPH, the Esprit Basic Research Action n^o 3166 (ASMICS) and the European Community Cooperation Action IC-1000 (ALTEC).

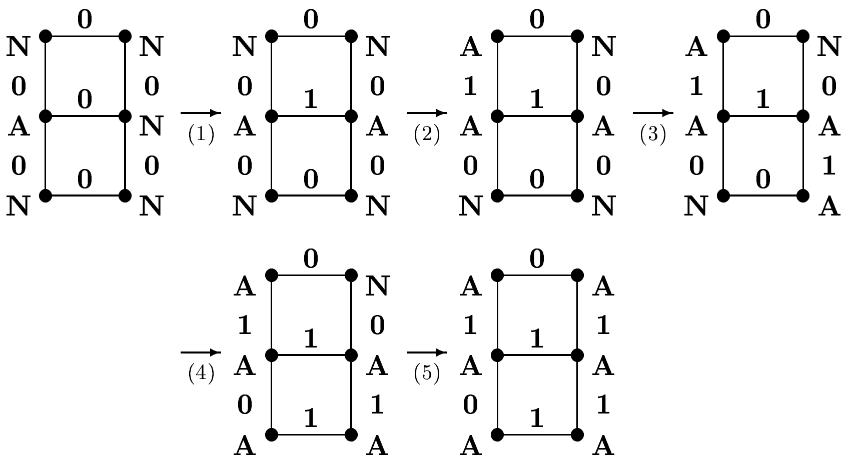


Figure 1: Computation of a spanning tree.

finite automata on all vertices. An elementary computation step then consists in computing the next state of each processor according to its own state and the states of all its neighbours. The latter one considers an asynchronous model. During an elementary computation step, two neighbouring vertices exchange their labels and then compute their new ones.

In this paper, we recall the basic ideas underlying the graph relabelling model and we survey the main questions that have been considered and the main results that have been obtained in that framework. The rest of this paper is organized as follows. In Section 2 we give the basic definitions and notation that will be used. In particular, we define two mechanisms, namely the notions of *priorities* and *forbidden contexts*, which allow to increase the expressive power of graph relabelling systems by adding some local control on the applicability of the relabelling rules.

One of the main motivations when introducing graph relabelling systems was to offer a suitable model for studying and for proving properties of distributed algorithms. In Section 3 we show how proof techniques issued from rewriting theory can be useful in this context.

Section 4 is devoted to the study of the well-known election problem. Starting from a graph whose all vertices have the same label, a vertex is said to be *elected* if, after some computation, it is the unique vertex in the graph having some distinguished label. This problem has only solutions for some special graph classes. We provide in particular a graph relabelling system which solves this problem for the class of so-called *prime graphs*.

From a more abstract point of view, graph relabelling systems allow to express *local computations* on graphs. One of the main questions is then to characterize those fonctions that can be locally computed in a graph. These notions are presented in Section 5.

In Section 6 we focus on the specific *recognition* problem. A graph relabelling system is said to recognize a given class of graphs if, starting from any uniformly labelled graph, it computes a final labelling which allows to decide whether the graph belongs to the class or not. We review some graph classes that can or cannot be recognized in such a way.

In Section 7 we introduce the notion of *k-covering* which generalizes the classical notion of graph covering. We show how this notion can be an useful tool for proving negative results concerning the capabilities of graph relabelling systems.

Finally, in Section 8 we deal with the so-called *termination detection* criteria which is a major parameter in distributed computing theory.

2 Basic definitions and notation

Unless otherwise stated, all the graphs considered in this paper are finite, undirected, without multiple edges, loopless and *connected.* For every graph G we denote by $V(G)$ its set of vertices and by $E(G)$ its set of edges. If G and G' are two graphs, we say that G' is a *subgraph* of G if $V(G') \subseteq V(G)$ and $E(G') \subseteq E(G)$. If X is a subset of $V(G)$, the subgraph of G *induced* by X has vertex set X and edge set the set of all edges whose both extremities belong to X. A *homomorphism* of a graph G to a graph H is a mapping φ from $V(G)$ to $V(H)$ such that $\varphi(x)\varphi(y)$ is an edge in H whenever xy is an edge in G. We say that φ is an *isomorphism* if φ is bijective and φ^{-1} is also a homomorphism. In the following, a set of graphs which is closed under isomorphism will be called a *class* of graphs.

Let $\mathcal{L}$ be a set whose elements are called *labels.* A *$\mathcal{L}$-labelled graph* is a pair (G, λ) where G is a graph and λ a mapping from $V(G) \cup E(G)$ to $\mathcal{L}$. If (G, λ) and (G', λ') are two labelled graphs, we say that (G', λ') is a (labelled) subgraph of (G, λ) if G' is a subgraph of G and λ' is the restriction of λ to $V(G') \cup E(G')$. We will denote by $\mathcal{G}_{\mathcal{L}}$ the set of all $\mathcal{L}$-labelled graphs. An isomorphism between two labelled graphs (G, λ) and (H, μ) is an isomorphism φ between G and H which preserves the labels, that is $\lambda(x) = \mu(\varphi(x))$ for every x in $V(G)$ and $\lambda(xy) = \mu(\varphi(x)\varphi(y))$ for every xy in $E(G)$. An *occurrence* of (G, λ) in (H, μ) is an isomorphism φ between G and a subgraph (H', μ') of (H, μ). We will then write $\varphi(G, \lambda) = (H', \mu')$.

A *(graph) relabelling rule* is a triple $R = (G_R, \lambda_R, \lambda'_R)$ such that (G_R, λ_R) and (G_R, λ'_R) are two labelled graphs. The labelled graph (G_R, λ_R) (resp. (G_R, λ'_R)) is called the *left-hand side* (resp. *right-hand side*) of R.

A *graph relabelling system* (GRS for short) is a triple $\mathcal{R} = (\mathcal{L}, \mathcal{I}, P)$ where $\mathcal{L}$ is a set of labels, $\mathcal{I}$ a subset of $\mathcal{L}$ called the set of *initial labels* and P a finite set of relabelling rules. A *$\mathcal{R}$-relabelling step* is a 5-tuple $(G, \lambda, R, \varphi, \lambda')$ such that R is a relabelling rule in P and φ is both an occurrence of (G_R, λ_R) in (G, λ) and an occurrence of (G_R, λ'_R) in (G, λ'). Intuitively speaking, the labelling λ' of G is obtained from λ by modifying all the labels of the elements of $\varphi(G_R, \lambda_R)$ according to the labelling λ'_R. Such a relabelling step will be denoted by $(G, \lambda) \longrightarrow_{R,\varphi} (G, \lambda')$. A *$\mathcal{R}$-relabelling sequence* is a tuple $(G, \lambda_0, R_0, \varphi_0, \lambda_1, R_1, \varphi_1, \lambda_2, \ldots, \lambda_{n-1}, R_{n-1}, \varphi_{n-1}, \lambda_n)$ such that for every i, $0 \leq i < n$, $(G, \lambda_i, R_i, \varphi_i, \lambda_{i+1})$ is a $\mathcal{R}$-relabelling step. The existence of such a relabelling sequence will be denoted by $(G, \lambda_0) \longrightarrow^*_{\mathcal{R}} (G, \lambda_n)$.

A labelled graph (G, λ) is said to be *$\mathcal{R}$-irreducible* if there exists no occurrence of (G_R, λ_R) in (G, λ) for every relabelling rule R in P. For every labelled graph (G, λ) in $\mathcal{G}_{\mathcal{I}}$ we denote by $Irred_{\mathcal{R}}(G, \lambda)$ the set of all $\mathcal{R}$-irreducible labelled graphs (G, λ') such that $(G, \lambda) \longrightarrow^*_{\mathcal{R}} (G, \lambda')$. Intuitively speaking, the set $Irred_{\mathcal{R}}(G, \lambda)$ contains all the final labellings that can be obtained from a $\mathcal{I}$-labelled graph (G, λ) by applying relabelling rules in P and may be viewed as the set of all the possible results of the computation encoded by the system $\mathcal{R}$.

Example 1 The algorithm introduced in Section 1 may be encoded by the graph relabelling system $\mathcal{R}_1 = (\mathcal{L}_1, \mathcal{I}_1, P_1)$ defined by $\mathcal{L}_1 = \{\mathbf{N}, \mathbf{A}, \mathbf{0}, \mathbf{1}\}$, $\mathcal{I}_1 = \{\mathbf{N}, \mathbf{A}, \mathbf{0}\}$, and $P_1 = \{R\}$ where R is the following rule:

R:

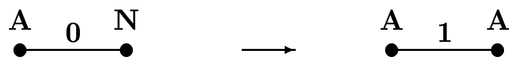


Fig. 1 describes a sample $\mathcal{R}_1$-relabelling sequence.

The notion of relabelling sequence defined above obviously corresponds to a notion of *sequential* computation. We can define a more distributed way of computing by saying that two relabelling steps concerning "disjoint" occurrences may be applied in any order, or even concurrently. It is easy to check that if $(G, \lambda_i, R_i, \varphi_i, \lambda_{i+1})$ and $(G, \lambda_{i+1}, R_{i+1}, \varphi_{i+1}, \lambda_{i+2})$ are two labelling steps such that $\varphi_i(G)$ and $\varphi_{i+1}(G)$ do not intersect then $(G, \lambda_i, R_{i+1}, \varphi_{i+1}, \lambda')$ and $(G, \lambda', R_i, \varphi_i, \lambda_{i+2})$ are two relabelling steps leading to the same resulting labelled graph (G, λ_{i+2}). More generally, any two relabelling sequences such that the latter one may be obtained from the former one by a succession of such "commutations" lead to the same resulting graph. Hence, our notion of relabelling sequence may be

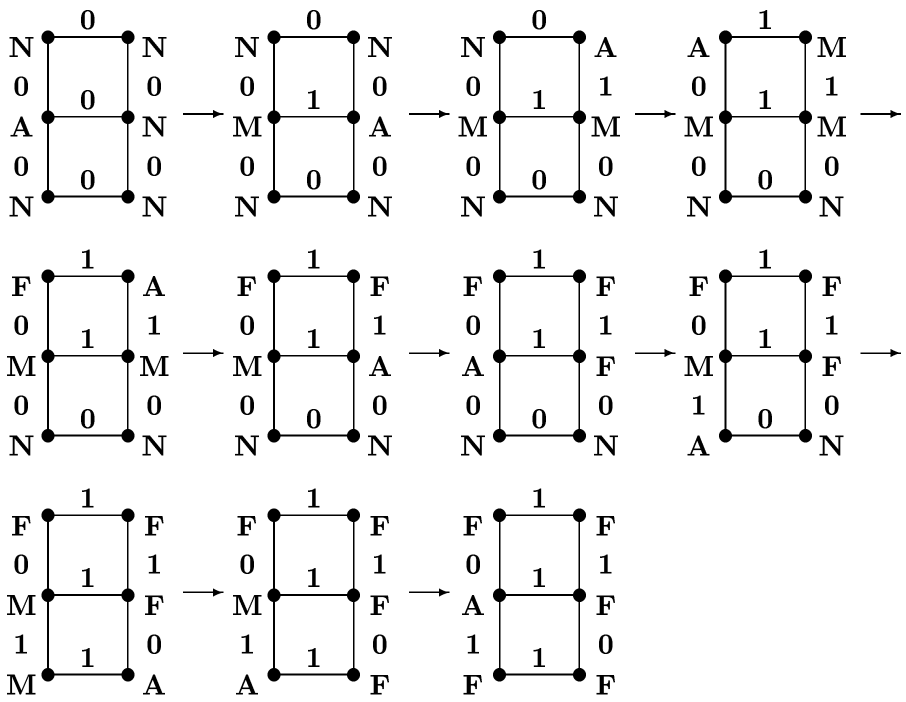


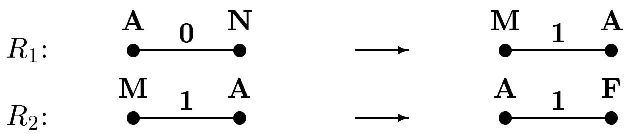
Figure 2: A sample $\mathcal{R}_2$-relabelling sequence.

regarded as a *serialization* [14] 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. In the sequel we will essentially deal with sequential relabelling sequences but the reader should keep in mind that such sequences may be done in a distributed way.

In order to reach a satisfactory expressive power, we introduce some *local control mechanisms*. These mechanisms allow us to restrict in some sense the applicability of relabelling rules.

A graph relabelling system *with priorities* (PGRS for short) is a 4-tuple $\mathcal{R} = (\mathcal{L}, \mathcal{I}, P, >)$ such that $(\mathcal{L}, \mathcal{I}, P)$ is a graph relabelling system and $>$ is a partial order defined on the set P called the *priority relation*. A *$\mathcal{R}$-relabelling step* is then defined as a 5-tuple $(G, \lambda, R, \varphi, \lambda')$ such that R is a relabelling rule in P, φ is both an occurrence of (G_R, λ_R) in (G, λ) and an occurrence of (G_R, λ'_R) in (G, λ') and there exists no occurrence φ' of a relabelling rule R' in P with $R' > R$ such that $\varphi(G_R)$ and $\varphi(G_{R'})$ intersect in G (that is $V(\varphi(G_R)) \cap V(\varphi(G_{R'})) = \emptyset$). The notion of relabelling sequence is defined as previously.

Example 2 Let $\mathcal{R}_2 = (\mathcal{L}_2, \mathcal{I}_2, P_2, >_2)$ be the PGRS defined by $\mathcal{L}_2 = \{\mathbf{N}, \mathbf{A}, \mathbf{M}, \mathbf{F}, \mathbf{0}, \mathbf{1}\}$, $\mathcal{I}_2 = \{\mathbf{N}, \mathbf{A}, \mathbf{0}\}$, $P_2 = \{R_1, R_2\}$ where R_1 and R_2 are the following relabelling rules:



with the priority relation: $R_1 >_2 R_2$.

Suppose that (G, λ) is a labelled graph containing exactly one **A**-labelled vertex. As before, this system computes a spanning tree of G but in a strictly sequential way, using the well-known depth-first search algorithm: the (unique) active vertex, with label **A**, may activate one of its **N**-labelled neighbours and become marked (label **M**). When an active vertex has no **N**-labelled neighbour, it reactivates its "father" (which corresponds to the unique **M**-labelled vertex to which it is linked by a **1**-labelled edge), and becomes **F**-labelled. Fig. 2 shows a sample $\mathcal{R}_2$-relabelling sequence.

Let (G, λ) be a labelled graph. A *context* of (G, λ) is a triple (H, μ, ψ) such that (H, μ) is a labelled graph and ψ an occurrence of (G, λ) in (H, μ). A *relabelling rule with forbidden contexts*

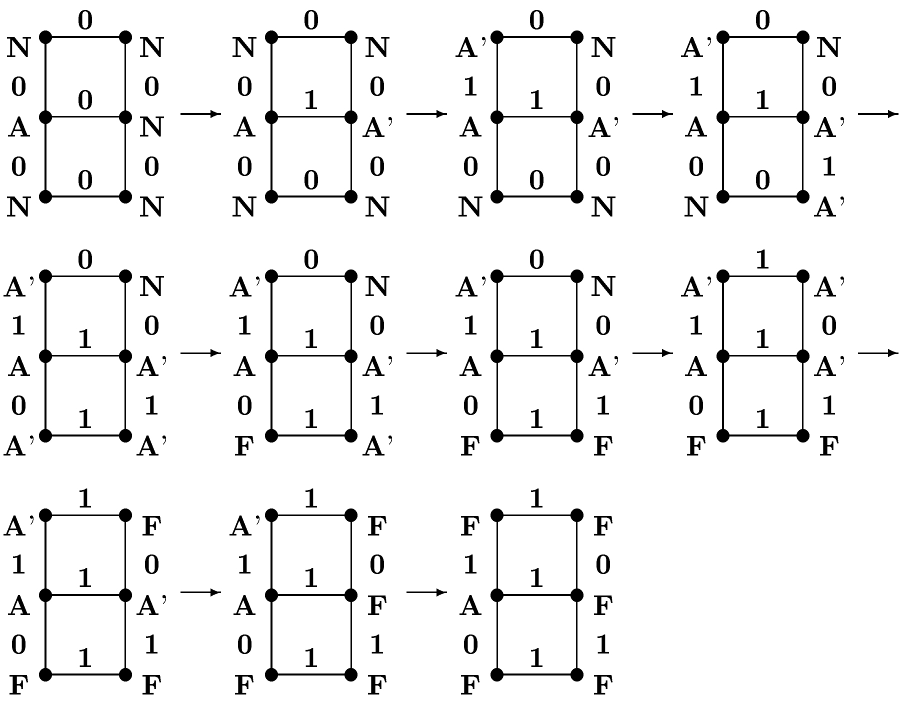
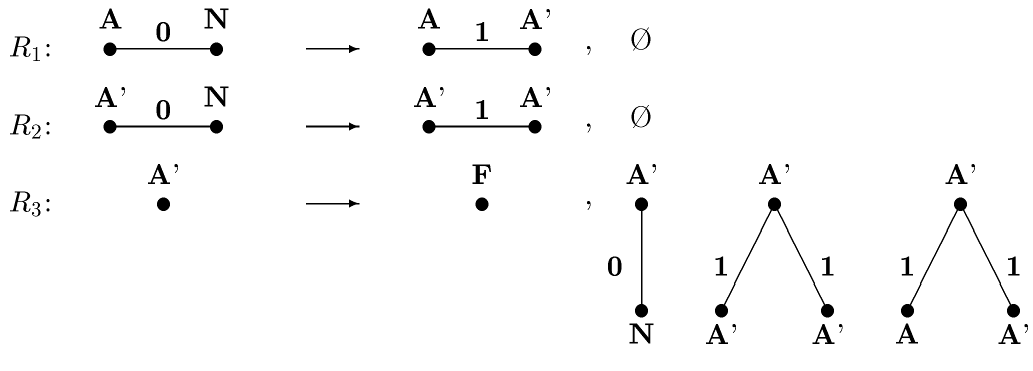


Figure 3: A sample $\mathcal{R}_3$-relabelling sequence.

is a 4-tuple $R = (G_R, \lambda_R, \lambda'_R, F_R)$ such that $(G_R, \lambda_R, \lambda'_R)$ is a relabelling rule and F_R is a finite set of contexts of (G_R, λ_R). A *graph relabelling system with forbidden contexts* (FCGRS for short) is a triple $\mathcal{R} = (\mathcal{L}, \mathcal{I}, P)$ defined as a GRS except that the set P is a set of relabelling rules with forbidden contexts. A $\mathcal{R}$-*relabelling step* is a 5-tuple $(G, \lambda, R, \varphi, \lambda')$ such that R is a relabelling rule with forbidden contexts in P, φ is both an occurrence of (G_R, λ_R) in (G, λ) and an occurrence of (G_R, λ'_R) in (G, λ'), and for every context (H_i, μ_i, ψ_i) of (G_R, λ_R), there is no occurrence φ_i of (H_i, μ_i) in (G, λ) such that $\varphi_i(\psi_i(G_R, \lambda_R)) = \varphi(G_R, \lambda_R)$. In other words, a relabelling rule with forbidden contexts may be applied on some occurrence if and only if this occurrence is not "included" in an occurrence of some of its forbidden contexts.

Example 3 Let $\mathcal{R}_3 = (\mathcal{L}_3, \mathcal{I}_3, P_3)$ be the FCGRS defined by $\mathcal{L}_3 = \{\mathbf{N}, \mathbf{A}, \mathbf{M}, \mathbf{F}, \mathbf{0}, \mathbf{1}\}$, $\mathcal{I}_3 = \{\mathbf{N}, \mathbf{A}, \mathbf{0}\}$, $P_3 = \{R_1, R_2, R_3\}$ where R_1, R_2 and R_3 are the following relabelling rules with forbidden contexts:



The unique vertex of the left-hand side of the rule R_3 is associated with the top vertex of its forbidden contexts. Roughly speaking, the rule R_3 means that a $\mathbf{A}'$-labelled vertex may become $\mathbf{F}$-labelled if it has no $\mathbf{N}$-labelled neighbour (in that case rule R_2 should be applied) and at most one $\mathbf{A}$- or $\mathbf{A}'$-labelled neighbour (it means that the $\mathbf{A}'$-labelled vertex is a leaf of the computed spanning tree).

This system provides a distributed implementation of the sequential algorithm encoded in Example 2 (we may have several active vertices, with label $\mathbf{A}$ or $\mathbf{A}'$, at the same time). Fig. 3 shows a sample $\mathcal{R}_3$-relabelling sequence.

Due to the control mechanism on the applicability of relabelling rules in PGRSs and FCGRSs, only relabelling steps concerning "far enough" occurrences may be applied concurrently [11]. Roughly speaking, in order to check whether a relabelling rule may be applied on a given occurrence or not it is necessary to consider some "control area" surrounding this occurrence. Two relabelling steps are then "independant" if their corresponding control areas do not intersect. The reader should note here that the diameter of this control area is bounded by some constant only depending on the graph relabelling system.

The comparison between the expressive power of PGRSs and FCGRSs, together with some other types of GRSs, has been done in [11]. In particular, it has been proved that PGRSs and FCGRSs are equivalent: for every PGRS (resp. FCGRS) there exists a FCGRS (resp. PGRS) achieving the same computation. In the rest of the paper we will thus indifferently provide examples under the PGRS or FCGRS form.

3 Proof techniques

Graph relabelling systems provide a formal model for expressing distributed algorithms. The aim of this Section is to show that this model is suitable for studying and proving properties of distributed algorithms.

A graph relabelling system $\mathcal{R}$ is *noetherian* if there is no infinite $\mathcal{R}$-relabelling sequence starting from a graph with initial labels in $\mathcal{I}$. Thus, if a distributed algorithm is encoded by a noetherian graph relabelling system then this algorithm always terminates. In order to prove that a given system is noetherian we generally use the following technique. Let $(S, <)$ be a partially ordered set with no infinite decreasing chain (that is every decreasing chain $x_1 > x_2 > \ldots > x_n > \ldots$ in S is finite). We say that $<$ is a *noetherian order compatible with* $\mathcal{R}$ if there exists a mapping f from $\mathcal{G}_{\mathcal{L}}$ to S such that for every $\mathcal{R}$-relabelling step $(G, \lambda, R, \varphi, \lambda')$ we have $f(G, \lambda) > f(G, \lambda')$. It is not difficult to see that if such an order exists then the system $\mathcal{R}$ is noetherian: since there is no infinite decreasing chain in S, there cannot exist any infinite $\mathcal{R}$-relabelling sequence.

In order to prove the correctness of a graph relabelling system, that is the correctness of an algorithm encoded by such a system, it is useful to exhibit (i) some *invariant properties* associated with the system (by invariant property, we mean here some property of the graph labelling that is satisfied by the initial labelling and that is preserved by the application of every relabelling rule) and (ii) some properties of irreducible graphs. These properties generally allow to derive the correctness of the system.

Let us illustrate these techniques by considering the simple graph relabelling system $\mathcal{R}_1$ given in Example 1.

Termination: Let f be the mapping from $\mathcal{G}_{\mathcal{L}_1}$ to the set of natural integers $\mathbb{N}$ which associates with each $\mathcal{L}_1$-labelled graph the number of its **N**-labelled vertices. Observing that this number strictly decreases when we apply the relabelling rule R_1 we get that $(\mathbb{N}, >)$ is a noetherian order compatible with the system $\mathcal{R}_1$. Thus $\mathcal{R}_1$ is a noetherian system.

Correctness: Let (G, λ) be a $\mathcal{L}_1$-labelled graph and P_1, P_2 be the following properties:

P_1 : Every **1**-labelled edge is incident with two **A**-labelled vertices,

P_2 : The subgraph of G made of the **1**-labelled edges and the **A**-labelled vertices has no cycle.

Every $\mathcal{I}_1$-labelled graph satisfies P_1 and P_2 since it has no **1**-labelled edge. Moreover, these two properties are clearly preserved when we apply the rule R_1. Thus, P_1 and P_2 are invariant with respect to $\mathcal{R}_1$.

Let now (G, λ) be any $\mathcal{I}_1$-labelled graph having at least one **A**-labelled vertex and (G, λ') be a labelled graph in $Irred_{\mathcal{R}_1}(G, \lambda)$. Considering the relabelling rule R_1, (G, λ') cannot have any **N**-labelled vertex. From property P_2, we get that the subgraph of (G, λ') induced by the **1**-labelled edges has no cycle. If (G, λ) has exactly one **A**-labelled vertex we thus obtain a spanning tree of G. If (G, λ)

has more than one **A**-labelled vertex we obtain a spanning forest having as many components as the number of these initially **A**-labelled vertices.

The reader interested in more substantial examples is referred to [9]. In particular, the graph relabelling systems introduced in Examples 2 and 3 are considered there.

The complexity of a distributed algorithm encoded by a graph relabelling system can also be studied by using classical techniques from rewriting theory. The space complexity is well-captured by the number of labels that are used, and the (sequential) time complexity by the length of a relabelling sequence. The degree of parallelism may also be measured by considering the ratio between the length of a parallel relabelling sequence and the length of a sequential relabelling sequence. Of course, this ratio strongly depends on the specific topology of the graph under consideration.

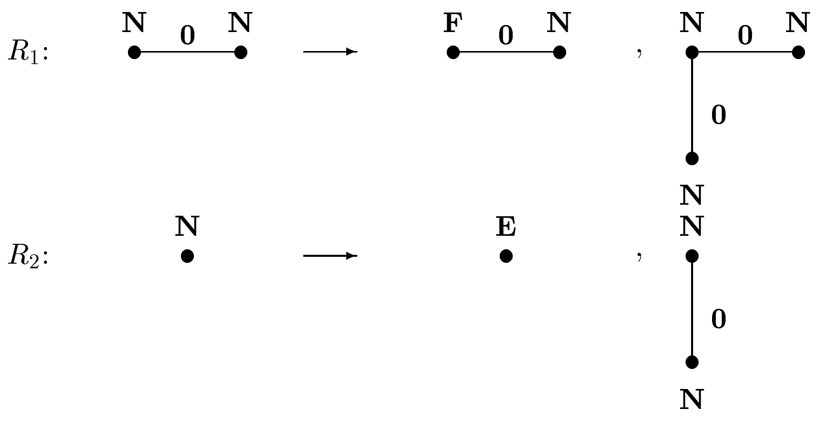
4 The election problem

The election problem is one of the paradigms of the theory of distributed computing [22]. Considering a network of processors we say that a given processor p has been *elected* when the network is in some global state such that the processor p knows that it is the elected processor and all other processors know that they are not. Using our terminology, it means that we get a labelling of the graph in which a unique vertex has some distinguished label.

This problem may be considered under various assumptions [22]: the network may be directed or not, the network may be anonymous (all vertices have the same initial label) or not (every two distinct vertices have distinct initial labels), all vertices, or some of them, may have some specific knowledge on the network or not (such as the diameter of the network, the total number of vertices or simply an upper bound of these parameters), etc.

We first illustrate this problem with a sample FCGRS electing a vertex in a tree.

Example 4 Let $\mathcal{R}_4 = (\mathcal{L}_4, \mathcal{I}_4, P_4)$ be the FCGRS defined by $\mathcal{L}_4 = \{\mathbf{N}, \mathbf{F}, \mathbf{E}, \mathbf{0}\}$, $\mathcal{I}_4 = \{\mathbf{N}, \mathbf{0}\}$ and $P_4 = \{R_1, R_2\}$ where R_1, R_2 are the following relabelling rules with forbidden contexts:



Let us call a *pendant* vertex any **N**-labelled vertex having exactly one **N**-labelled neighbour. The rule R_1 then consists in "cutting" a pendant vertex in the tree, this cut vertex becoming **F**-labelled. Thus, if (G, λ) is a labelled tree whose all vertices are **N**-labelled and all edges are **0**-labelled then this cuting procedure leads to a unique **N**-labelled vertex which becomes elected thanks to the rule R_2.

It is not difficult to observe that every vertex in the tree may be elected by this algorithm. A precise analysis of this algorithm is proposed in [18]. In particular, it is proved that there exist one or two vertices having the highest probability of being elected, namely the *medians* of the graph (recall here that a vertex is called a median if the sum of the distances of this vertex to all other vertices in the graph is minimum).

The following algorithm has been proposed by Mazurkiewicz [15] and is designed for oriented rings (that is networks whose corresponding graph is a directed cycle) having a prime number of vertices.

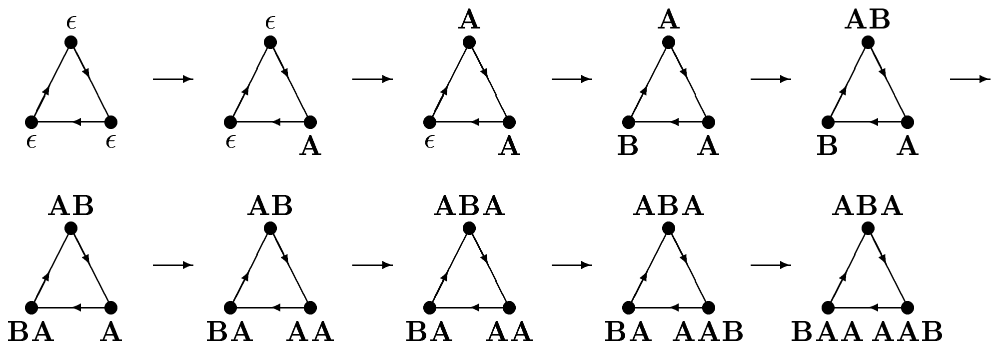
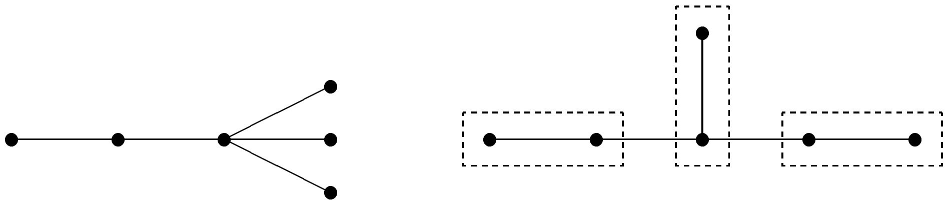


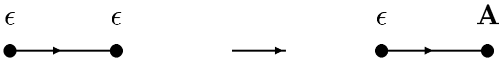
Figure 4: Election in a prime oriented ring



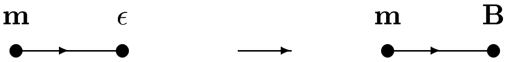
A prime tree *2-decomposition of a tree*

Figure 5: Prime and not prime graphs.

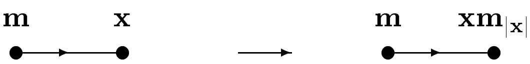
Example 5 Let $\mathcal{L}$ be the set of words on the alphabet $\{A, B\}$ with length at most n, $n \geq 3$. Let ϵ denotes the empty word, $|m|$ denotes the length of a word m and m_i denote the i^{th} letter of the word m. Consider the following rules:

R_1: 

For every non-empty word m:

$R_2(m)$: 

For every words m and x with $0 < |x| < n$ and $|x| \leq |m|$:

$R_3(m, x)$: 

Mazurkiewicz proved that if (G, λ) is a directed cycle on n vertices, n being a prime number, whose all vertices are initially ϵ-labelled then this algorithm always terminates and leads to a final labelling such that (i) all vertices are labelled by distinct words of length n, (ii) all these labels are conjugate of some word w (recall that two words u and v are conjugate if $u = u_1u_2$ and $v = u_2u_1$). The elected vertex is then the vertex having the minimum label with respect to the lexicographic ordering. Therefore, every vertex may know whether it has been elected or not by considering the set of all the conjugates of its own final label. Fig. 4 shows a sample execution of this algorithm on an oriented triangle. The vertex with final label **AAB** is the elected vertex.

Observe that this algorithm requires that every vertex knows the total number of vertices in the cycle, this number being used in the definition of the relabelling rules.

The election problem has been considered in the undirected case in [19]. For instance, it has been proved that the election problem can be solved for the so-called *prime graphs*, provided that every vertex knows the total number of vertices in the graph. Let G be an undirected graph and r be a positive integer. A r-*decomposition* of G is a spanning forest of G whose all connected components (trees) contain exactly r vertices. A graph having n vertices is then said to be prime if it only admits 1- and n-decompositions. Fig. 5 illustrates this notion of primality. The class of prime graphs obviously contains all the graphs having a prime number of vertices.

An election algorithm for the class of prime graphs can then be intuitively described as follows: we associate with each vertex x of the graph a weight denoted by $w(x)$. Initially, the weight of every vertex is 1. The algorithm maintains a spanning forest of the graph whose every tree has a distinguished vertex called the *leader* of the tree. The weight of this leader equals the size of the tree. Initially, every vertex constitutes a tree of the spanning forest and it is the leader of this tree. We say that two trees T_1 and T_2 of the spanning forest are *adjacent* if there exists an edge x_1x_2 such that x_1 is a vertex in T_1 and x_2 is a vertex in T_2. The algorithm then proceeds as follows:

1. If two leaders with weight 1 are adjacent then they are combined into a unique tree; one of them becomes the new leader (with weight 2), the weight of the other one is set to 0.

2. A leader L with a weight $w(L) \geq 2$ tries to find an adjacent tree whose leader L' is such that $w(L) > w(L')$. If it finds one, then the two trees are combined into a unique tree with leader L. The weight of L becomes the size of the new tree and the weight of L' is set to 0.

If the graph is prime, it is not difficult to check that this algorithm stops when the spanning forest contains a unique tree. The leader of this tree is then the elected vertex. The complete description of this algorithm can be found in [19].

5 Local computations in graphs

One of the main characteristics of distributed algorithms is the *locality* of the computation [8, 22]. Every computation step occurring on some processor only depends on the local context of this processor. This locality concept is captured via the notion of *local graph relabelling relations* [13].

Let G be a graph, x a vertex in $V(G)$ and k some positive integer. We denote by $B_G(x,k)$ the *ball* of radius k centered at x, that is the subgraph of G induced by all vertices that are at distance at most k from x (recall that the distance between two vertices is the length of a shortest path linking these two vertices). A *graph relabelling relation* (over $\mathcal{L}$) is a binary relation $\mathcal{R}$ defined on the set of $\mathcal{L}$-labelled graphs such that every pair in $\mathcal{R}$ is of the form $((G,\lambda),(G,\lambda'))$. Thus, two labelled graphs in relation only differ on their labelling function. We will write $(G,\lambda)\mathcal{R}(G,\lambda')$ whenever the pair $((G,\lambda),(G,\lambda'))$ is in $\mathcal{R}$. A $\mathcal{L}$-labelled graph (G,λ) is said to be *$\mathcal{R}$-irreducible* if there exists no (G,λ') such that $(G,\lambda)\mathcal{R}(G,\lambda')$. We will denote by $\mathcal{R}^*$ the reflexive and transitive closure of $\mathcal{R}$ and, for every $\mathcal{L}$-labelled graph (G,λ), by $Irred_{\mathcal{R}}(G,\lambda)$ the set of $\mathcal{R}$-irreducible graphs (G,λ') such that $(G,\lambda)\mathcal{R}^*(G,\lambda')$.

We say that a graph relabelling relation $\mathcal{R}$ is *k-local* for some positive integer k if for every pair $((G,\lambda),(G,\lambda'))$ in $\mathcal{R}$, there exists some vertex x in $V(G)$ such that λ and λ' coincide on $V(G) \setminus V(B_G(x,k)) \cup E(G) \setminus E(B_G(x,k))$. Intuitively speaking, it means that λ and λ' only differ on a centered ball of radius at most k. A graph relabelling relation is *local* if it is k-local for some k. A graph relabelling relation $\mathcal{R}$ is *k-locally generated* if it can be computed for any graph as soon as it is known on the set of graphs with diameter at most $2k$. More formally, if (G,λ), (G',λ'), (H,μ), (H',μ') are four labelled graphs, $B_G(x,k)$ and $B_H(y,k)$ two isomorphic balls in G and H respectively such that (i) λ and λ' coincide on $V(G) \setminus V(B_G(x,k)) \cup E(G) \setminus E(B_G(x,k))$, (ii) μ and μ' coincide on $V(H) \setminus V(B_H(y,k)) \cup E(H) \setminus E(B_H(y,k))$ and (iii) λ and μ coincide respectively on $B_G(x,k)$ and $B_H(y,k)$ then $(G,\lambda)\mathcal{R}(G',\lambda')$ if and only if $(H,\mu)\mathcal{R}(H',\mu')$. A graph relabelling relation is *locally generated* if it is k-locally generated for some k.

Graph relabelling systems (GRSs, PGRSs, FCGRSs) are thus special cases of locally generated graph relabelling relations. One of the main questions in that framework is "what can be computed by means of locally generated graph relabelling relations ?". This question is obviously strongly related to the general problem of characterizing those functions that can be computed by distributed algorithms in an asynchronous way (see e.g. [20]). The next Section is devoted to that question and discuss the problem of characterizing those classes of graphs that can be recognized by means of such local computations.

6 The recognition problem

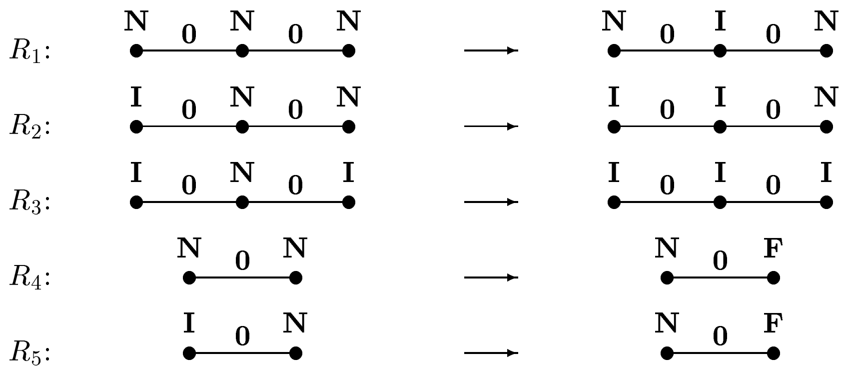
The problem adressed in this section can be informally described as follows: let $\mathcal{F}$ be some class of (unlabelled) graphs. We will say that this class can be *locally recognized* if there exists some graph relabelling system or, more generally, some locally generated graph relabelling relation, which, starting from any uniformly labelled graph (G, λ_0) (that is all vertices and edges have the same label), leads to some final labelling that allows to decide whether G belongs to the class $\mathcal{F}$ or not.

More formally, we define a *graph recognizer* as a pair $(\mathcal{R}, \mathcal{K})$ where $\mathcal{R}$ is a graph relabelling relation and $\mathcal{K}$ a class of labelled graphs. The set of labelled graphs *recognized* by $(\mathcal{R}, \mathcal{K})$ is then defined as the set of labelled graphs (G, λ) such that $Irred_{\mathcal{R}}(G, \lambda) \cap \mathcal{K} \neq \emptyset$. Such a recognizer is said to be *deterministic* if (i) $\mathcal{R}$ is noetherian and (ii) for every labelled graph (G, λ), either $Irred_{\mathcal{R}}(G, \lambda) \subseteq \mathcal{K}$ or $Irred_{\mathcal{R}}(G, \lambda) \cap \mathcal{K} = \emptyset$.

We are essentially interested in graph recognizers where the relation $\mathcal{R}$ is locally generated (with the particular case of graph relabelling systems) and the set $\mathcal{K}$ is defined in some "simple way". In [13] this set $\mathcal{K}$ is defined by means of a so-called *final condition*, that is a logical formula inductively defined as follows: (i) for every label $\ell \in \mathcal{L}$, ℓ is a formula and (ii) if φ and ψ are formulas then so do $\neg\varphi$, $\varphi \vee \psi$ and $\varphi \wedge \psi$. Now, for $\ell \in \mathcal{L}$, a labelled graph satisfies the formula ℓ if it contains at least one ℓ-labelled component, and by induction, it satisfies $\varphi \vee \psi$ if it satisfies φ or ψ and so on in the usual way. Thus, such final conditions allow to verify the presence or the absence of some specific labels but not to count the number of such labels. We will denote by $\mathcal{K}(\varphi)$ the set of labelled graphs which satisfy the formula φ.

We first illustrate this recognition mechanism with a tree recognizer given in [9].

Example 6 Let $\mathcal{R}_5 = (\mathcal{L}_5, \mathcal{I}_5, P_5, >_5)$ be the PGRS defined by $\mathcal{L}_5 = \{\mathbf{N}, \mathbf{I}, \mathbf{F}, \mathbf{0}\}$, $\mathcal{I}_5 = \{\mathbf{N}, \mathbf{0}\}$, $P_5 = \{R_1, R_2, R_3, R_4, R_5\}$ with the rules:



and the priority relation: $\{R_1, R_2, R_3\} >_5 \{R_4, R_5\}$.

Let now φ be the final condition $\varphi = \neg\mathbf{I}$. It can be proved that if (G, λ) is a labelled graph whose all vertices are $\mathbf{N}$-labelled and all edges are $\mathbf{0}$-labelled then every labelled graph (G, λ') in $Irred_{\mathcal{R}_5}(G, \lambda)$ has no $\mathbf{I}$-labelled vertex, and thus satisfies φ, if and only if G has no cycle. Hence, the pair $(\mathcal{R}_5, \mathcal{K}(\varphi))$ is a deterministic recognizer for the class of trees.

In [10, 13] the recognizable classes of graphs are compared to the classes of graphs definable by logic formulas (see [4] for the notion of definability by logic formulas). In particular, it is proved that (deterministically or not) recognizable classes of graphs are not comparable with classes of graphs definable by logic formulas expressed in first-order logic (FOL), monadic second-order logic (MSOL) or second-order logic (SOL). The case of the so-called *1-graphs*, that is graphs having a distinguished

Table 1: Recognizable and not-recognizable graph classes

Graph properties	Graphs	1-Graphs
FOL		
exactly one ℓ-labelled vertex	No	Yes
k-regular	Yes	Yes
MSOL		
bipartite	No	Yes
k-colorable ($k > 2$)	No	?
hamiltonian	No	Yes
acyclic	Yes	Yes
SOL		
even number of vertices	No	Yes

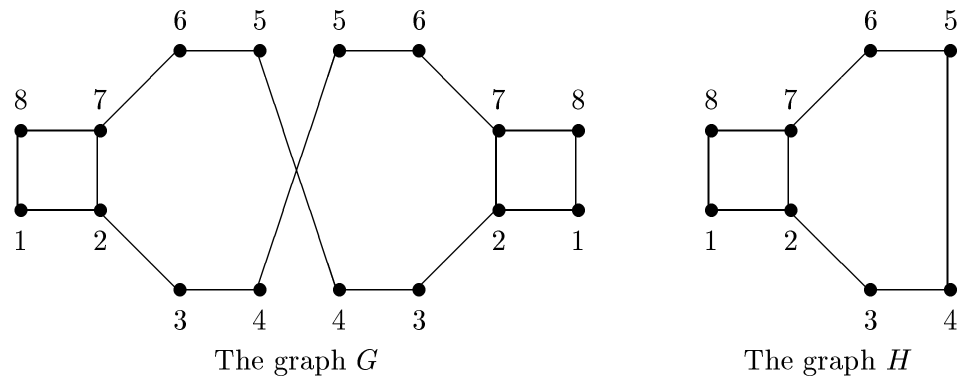


Figure 6: The graph G is a 2-covering of the graph H.

vertex is also considered. Table 1 gives some sample graph classes or 1-graph classes that can or cannot be deterministically recognized.

The class of graphs having an even number of vertices can be undeterministically recognized but cannot be deterministically recognized. The class of graphs having an odd number of vertices cannot be recognized, even in an undeterministic way. Thus, the set of deterministically recognizable classes of graphs is not closed under taking complement and is strictly included in the set of undeterministically recognizable classes of graphs. However, the set of deterministically recognizable classes of graphs is closed under union and intersection [13].

The majority problem has been considered in [12]. It is proved that the class of graphs having strictly more **A**-labelled vertices than **B**-labelled vertices is deterministically recognizable. However, for every positive integer m, the class of graphs such that the difference between the number of **A**-labelled vertices and the number of **B**-labelled vertices is at most m cannot be recognized, even in an undeterministic way.

The main question here is to find some characterization of the classes of graphs that can be recognized by locally generated graph relabelling relations. Up to now, this question is still an open problem.

7 k-coverings of graphs

Inspired by techniques used by Angluin [1] and Fisher *et al.* [7], we define the notion of k-covering, introduced in [13], which generalizes the classical notion of covering from graph theory. This notion is useful for proving negative results concerning locally generated graph relabelling relations.

Let k be a positive integer. We say that a labelled graph (G, λ) is a *k-covering* of a labelled graph (H, μ) via a mapping γ from $V(G)$ to $V(H)$ if γ is a surjective homomorphism such that for every vertex x of $V(G)$, the restriction of γ to $B_G(x, k)$ induces an isomorphism between $B_G(x, k)$ and $B_H(\gamma(x), k)$ which preserves vertex and edge labels. Fig. 6 shows two sample (unlabelled for simplicity) graphs G and H such that G 2-covers H. The numbering of the vertices defines the corresponding 2-covering γ.

The notion of k-covering is related to k-locally generated graph relabelling relations by the following result:

Theorem 7 [13] If a labelled graph (G, λ) is a k-covering of a labelled graph (H, μ) then every k-locally generated graph relabelling relation $\mathcal{R}$ that recognizes (H, μ) also recognizes (G, λ). If $\mathcal{R}$ recognizes deterministically then (G, λ) is recognized if and only if (H, μ) is recognized.

This result follows from the easy observation that if $(H, \mu)\mathcal{R}(H, \mu')$ then there exists a labelling function λ' such that $(G, \lambda)\mathcal{R}^*(G, \lambda')$ and (G, λ') k-covers (H, μ'). If μ' modifies the centered ball $B_H(x, k)$, then λ' is obtained from λ by reproducing these modifications on the corresponding inverse image $\gamma^{-1}(B_H(x, k))$ (which is a finite set of balls isomorphic to $B_H(x, k)$), where γ stands for the k-covering of (H, μ) by (G, λ).

Since there exist planar graphs with non-planar k-coverings for every k, we get that the class of planar graphs cannot be recognized, even in an undeterministic way [13].

Using this notion of k-covering, it is proved in [5] that every non-trivial minor-closed class of graphs containing at least one graph with at least two cycles cannot be recognized by a k-locally generated graph relabelling relation.

A standard method for producing coverings of a graph G is to consider the kronecker product of G by the complete graph K_2 (recall that the kronecker product of G and H is the graph with vertex set $V(G) \times V(H)$ and with edge set those pairs $\{\{x, y\}, \{z, t\}\}$ with $\{x, z\} \in E(G)$ and $\{y, t\} \in E(H)$). This construction has been studied in [3]. By considering properties of this construction it has been proved in particular that the classes of graphs having a cut-vertex or a cut-edge, of graphs with trivial automorphism group, of non-bipartite or non-planar graphs are not recognizable by locally generated graph relabelling relations, even in an undeterministic way.

In [21] Reidemeister gave an elegant method for constructing all the coverings of a graph. Up to now, no such construction method is known in the case of k-coverings.

8 The termination detection problem

An important property in distributed computing theory is the capability, for a given vertex, to detect the termination of the algorithm [22]. To be really effective, this detection should be done in some simple way, namely by examining the labels of the "closed neighbourhood" of a vertex.

More formally, we will say that a graph relabelling system $\mathcal{R}$ has the *k-local termination detection property* (k-LTDP for short) if (i) there exists a (not necessarily finite) set $\mathcal{B}$ of triples (B_i, λ_i, x_i) such that (B_i, λ_i) is a labelled graph and x_i a vertex in $V(B_i)$ such that x_i is at distance at most k from any other vertex in $V(B_i)$, and (ii) for every labelled graph (G, λ), there exists a vertex x in $V(G)$, a positive integer $k' \leq k$ and an isomorphism between $B_G(x, k')$ and some (B_i, λ_i, x_i) which maps x to x_i if and only if the graph (G, λ) is $\mathcal{R}$-irreducible.

Let us illustrate this notion on the three examples given in Section 2. The graph relabelling system $\mathcal{R}_1$ (see Example 1) is not k-LTDP, for every k. Every **N**-labelled vertex knows that the computation is not terminated, but a **A**-labelled vertex cannot detect the termination since the computation may be still "active" in a part of the graph which is unboundedly far from this vertex. On the contrary, considering the PGRS $\mathcal{R}_2$ (see Example 2) and the FCGRS $\mathcal{R}_3$ (see Example 3), it is easy to check that in both cases, if a **A**-labelled vertex is such that all its neighbours are **F**-labelled, then the graph is necessarily irreducible. Thus, the systems $\mathcal{R}_2$ and $\mathcal{R}_3$ are both 1-LTDP.

In [17] it is proved that if $\mathcal{C}$ is a class of labelled graphs, (G, λ) is a labelled graph and (H, μ) is a connected non-trivial k-covering of (G, λ) such that both (G, λ) and (H, μ) belong to the class $\mathcal{C}$, then we cannot locally detect whether a graph in $\mathcal{C}$ is irreducible with respect to any k-locally generated graph relabelling relation. From other results given in [17] we can deduce that to elect a vertex in the class of prime graphs (see Section 4) it is necessary to know the size of the graph.

References

[1] Angluin, D.: Local and global properties in networks of processors. Proc. 12th ACM Symposium on the Theory of Computing (STOC), 82–93, 1980.

[2] Billaud, M.—Lafon, P.—Métivier, Y.—Sopena, E.: Graph rewriting systems with priorities. Proc. WG'89. Lecture Notes in Comput. Sci. 411:94–106, 1989.

[3] Bottreau, A.—Métivier, Y.: The Kronecker product and local computations in graphs. Proc. CAAP'96. Lecture Notes in Comput. Sci. 1059:2–16, 1996.

[4] Courcelle, B.: The monadic second-order logic of graphs I: Recognizable sets of finite graphs. Inform. and Comput. 85:12–75, 1990.

[5] Courcelle, B.—Métivier, Y.: Coverings and minors : application to local computations in graphs. Europ. J. Combin. 15:127–138, 1994.

[6] Fiksel, J.R.—Holliger, A.—Rosenstiehl, P.: Intelligent graphs. In: Graph Theory and Computing, R.C. Read (Editor), Academic Press, New York, 219–265, 1972.

[7] Fisher, M.J.—Lynch, N.A.—Merrit, M.: Easy impossibility proofs for distributed consensus problem. Distrib. Comput. 1:26–39, 1986.

[8] Linial, N.: Locality in distributed graph algorithms. Siam J. Comput. 21(1):193–201, 1992.

[9] Litovsky, I.—Métivier, Y.: Computing trees with graph rewriting systems with priorities. In : Tree automata and languages, M. Nivat and A. Podelski (Editors), Elsevier, 115–139, 1992.

[10] Litovsky, I.—Métivier, Y.: Computing with graph rewriting systems with priorities. Theoret. Comput. Sci. 115:191–224, 1993.

[11] Litovsky, I.—Métivier, Y.—Sopena, E.: Different local controls for graph relabelling systems. Math. Syst. Theory 28:41–65, 1995.

[12] Litovsky, I.—Métivier, Y.—Sopena, E.: Checking global graph properties by means of local computations : the majority problem. Electronic Notes in Comput. Sci. 1, 1995.

[13] Litovsky, I.—Métivier, Y.—Zielonka, W.: On the recognition of families of graphs with local computations. Inform. and Computation 118(1):110–119, 1995.

[14] Mazurkiewicz, A.: Trace Theory. Lecture Notes in Comput. Sci. 255:279–324, 1987.

[15] Mazurkiewicz, A.: Solvability of the asynchronous ranking problem. Inform. Process. Letters 28:221–224, 1988.

[16] Mazurkiewicz, A.: Election in planar graphs. Research Report 90-105, Univ. Bordeaux I, 1990.

[17] Métivier, Y.—Muscholl, A.—Wacrenier, P.A.: About the local detection of the global termination of a distributed algorithm. In preparation.

[18] Métivier, Y.—Saheb, N.: Probabilistic analysis of an election algorithm in a tree. Proc. CAAP'94. Lecture Notes in Comput. Sci. 787:234–245, 1994.

[19] Métivier, Y.—Saheb, N.—Wacrenier, P.A.: A distributed algorithm for computing a spanning tree for the family of prime graphs. In preparation.

[20] Naor, M.—Stockmeyer, L.: What can be computed locally?. Proc. 25th ACM Symposium on the Theory of Computing (STOC), 184–193, 1993.

[21] Reidemeister, K.: Einfürung in die Kombinatorische Topologie. Vieweg, Brunswick, 1932.

[22] Tel, G.: Introduction to distributed algorithms. Cambridge University Press, 1994.