Calculs Locaux dans les Graphes (Local Computations in Graphs)

Inspirés des travaux de Y. Métivier et M. Mosbah
www.labri.fr/visidia

LaBRI - Université de Bordeaux 1

1 Graphs

The notations used here are essentially standard [Ros00]. We only consider finite, undirected, connected graphs without multiple edges and self-loops. If G is a graph, then $V(G)$ denotes the set of vertices and $E(G)$ denotes the set of edges. Two vertices u and v are said to be adjacent if $\{u,v\}$ belongs to $E(G)$. The distance between two vertices u, v is denoted $d(u,v)$. The set of neighbours of v in G, denoted $N_G(v)$, is the set of all vertices of G adjacent to v.

Let v be a vertex and k a non negative integer, we denote by $B_G(v,k)$, or briefly $B(v,k)$, the centered ball of radius k with center v i.e., the subgraph of G defined by the vertex set $V' = \{v' \in V(G) \mid d(v,v') \leq k\}$ and the edge set $E' = \{\{v_1,v_2\} \in E(G) \mid d(v,v_1) < k \text{ and } d(v,v_2) \leq k\}$; we denote by $B_G(v)$ the ball of radius 1 with center v.

A homomorphism between G and H is a mapping $\gamma\colon V(G) \to V(H)$ such that if $\{u,v\}$ is an edge of G then $\{\gamma(u),\gamma(v)\}$ is an edge of H. Since we deal only with graphs without self-loops, we have $\gamma(u) \neq \gamma(v)$ whenever $\{u,v\}$ is an edge of G. Note also that $\gamma(N_G(u)) \subseteq N_H(\gamma(u))$. For an edge $\{u,v\}$ of G we define $\gamma(\{u,v\}) = \{\gamma(u),\gamma(v)\}$; this extends γ to a mapping $V(G) \cup E(G) \to V(H) \cup E(H)$. We say that γ is an isomorphism if γ is bijective and γ^{-1} is a homomorphism, too. We write $G \simeq G'$ whenever G and G' are isomorphic. A class of graphs will be any set of graphs containing all graphs isomorphic to some of its elements. The class of all graphs will be denoted $\mathcal{G}$. Given a graph G, a graph H whose vertices and edges are all in G is a subgraph of G. An occurrence of G in G' is an isomorphism γ between G and a subgraph H of G'.

For any set S, $\mathrm{card}(S)$ (or $|S|$) denotes the cardinality of S. For any integer q, we denote by $[1,q]$ the set $\{1,2,\ldots,q\}$.

1.1 Labelled Graphs

Throughout the paper we will consider graphs where vertices and edges are labelled with labels from a recursive alphabet L . A graph labelled over L will be denoted by (G,λ), where G is a graph and $\lambda\colon V(G) \cup E(G) \to L$ is the labelling function. The graph G is called the underlying graph and the mapping λ is a labelling of G. For a labelled graph (G,λ), $lab((G,\lambda))$ is the set of labels that occur in (G,λ), i.e.,

$$lab((G,\lambda)) = \{\lambda(x) \mid x \in V(G) \cup E(G)\}.$$

The class of labelled graphs over some fixed alphabet L will be denoted by $\mathcal{G}_L$. Note that since L is recursive, also $\mathcal{G}_L$ is recursive.

Let (G,λ) and (G',λ') be two labelled graphs. Then (G,λ) is a subgraph of (G',λ'), denoted by $(G,\lambda) \subseteq (G',\lambda')$, if G is a subgraph of G' and λ is the restriction of the labelling λ' to $V(G) \cup E(G)$.

A mapping $\gamma\colon V(G) \to V(G')$ is a homomorphism from (G,λ) to (G',λ') if γ is a graph homomorphism from G to G' which preserves the labelling, i.e., such that $\lambda'(\gamma(x)) = \lambda(x)$ holds for every $x \in V(G) \cup E(G)$.

An *occurrence* of (G, λ) in (G', λ') is an isomorphism γ between (G, λ) and a subgraph (H, η) of (G', λ'). It shall be denoted $\gamma : (G, \lambda) \hookrightarrow (G', \lambda')$.

Labelled graphs will be designated by bold letters like $\mathbf{G}$, $\mathbf{H}, \ldots$. If $\mathbf{G}$ is a labelled graph, then G denotes the underlying graph.

2 Local Computations

They model networks of processors of arbitrary topology. The network is represented as a connected, undirected graph where vertices denote processors and edges denote direct communication links. Labels (or states) are attached to vertices and edges.

Graph relabelling systems and more generally local computations satisfy the following constraints, that arise naturally when describing distributed computations with decentralized control:

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

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

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

The relabelling is performed until no more transformation is possible, i.e., until a normal form is obtained.

2.1 Definitions

Local computations as considered here can be described in the following general framework. Let $\mathcal{G}$ be the class of L-labelled graphs and let $\mathcal{R} \subseteq \mathcal{G} \times \mathcal{G}$ be a binary relation on $\mathcal{G}$. Then $\mathcal{R}$ will denote a graph rewriting relation. We assume that $\mathcal{R}$ is closed by isomorphism, i.e. whenever $G \ \mathcal{R} \ G'$, if $G_1 \simeq G$ then $G_1 \ \mathcal{R} \ G'_1$ for some graph $G'_1 \simeq G'$.

Definition 1. *Let $\mathcal{R} \subseteq \mathcal{G} \times \mathcal{G}$ be a graph rewriting relation and let $k > 0$ be an integer.*

1. *$\mathcal{R}$ is a* relabelling relation *if whenever two labelled graphs are in relation then the underlying graphs are equal (we say equal, not only isomorphic), i.e.:*

$$(G, \lambda) \ \mathcal{R} \ (H, \lambda') \quad \Longrightarrow \quad G = H.$$

2. *$\mathcal{R}$ is called $k-$*local *if only labels of a ball of radius k may be changed by $\mathcal{R}$, i.e. $(G, \lambda) \ \mathcal{R} \ (G, \lambda')$ implies that there exists a vertex $v \in V(G)$ such that*

$$\lambda(x) = \lambda'(x) \textit{ for every } x \notin V(B_G(v, k)) \cup E(B_G(v, k)).$$

The relation $\mathcal{R}$ is called local, *if it is $k-$local for some $k > 0$.*

3. *An $\mathcal{R}$ $-$*normal form *of $G \in \mathcal{G}$ is a graph G' such that $G \ \mathcal{R}^* \ G'$, but $G' \ \mathcal{R} \ G''$ holds for no $G'' \in \mathcal{G}$. We say that $\mathcal{R}$ is* noetherian *if there is no infinite relabelling chain $G_1 \ \mathcal{R} \ G_2 \ \mathcal{R} \cdots$.*

The next definition states that a local relabelling relation $\mathcal{R}$ is *$k-$locally generated* if its restriction on centered balls of radius k determines its computation on any graph.

Definition 2. *Let $\mathcal{R}$ be a relabelling relation and $k > 0$ be an integer. Then $\mathcal{R}$ is called $k-$*locally generated *if the following holds: For any labelled graphs (G, λ), (G, λ'), (H, η), (H, η') and any vertices $v \in V(G)$, $w \in V(H)$ such that the balls $B_G(v, k)$ and $B_H(w, k)$ are isomorphic via $\varphi\colon V(B_G(v, k)) \longrightarrow V(B_H(w, k))$ and $\varphi(v) = w$, the following three conditions*

1. $\lambda(x) = \eta(\varphi(x))$ *and* $\lambda'(x) = \eta'(\varphi(x))$ *for all* $x \in V(B_G(v, k)) \cup E(B_G(v, k))$
2. $\lambda(x) = \lambda'(x)$, *for all* $x \notin V(B_G(v, k)) \cup E(B_G(v, k))$
3. $\eta(x) = \eta'(x)$, *for all* $x \notin V(B_H(w, k)) \cup E(B_H(w, k))$

imply that $(G, \lambda)\ \mathcal{R}\ (G, \lambda')$ holds if and only if $(H, \eta)\ \mathcal{R}\ (H, \eta')$.

$\mathcal{R}$ *is called* locally generated *if it is $k-$locally generated for some $k > 0$.*

2.2 Local Computations on Balls of Radius 1

Local computations as considered here can be described in the following general framework. Let $\mathcal{G}_L$ be the class of L-labelled graphs and let $\mathcal{R} \subseteq \mathcal{G}_L \times \mathcal{G}_L$ be a binary relation on $\mathcal{G}_L$. Then $\mathcal{R}$ is called a *graph rewriting relation.* We assume that $\mathcal{R}$ is closed under isomorphism, i.e., if $\mathbf{G}\ \mathcal{R}\ \mathbf{G}'$ and $\mathbf{H} \simeq \mathbf{G}$ then $\mathbf{H}\ \mathcal{R}\ \mathbf{H}'$ for some labelled graph $\mathbf{H}' \simeq \mathbf{G}'$. In the remainder of the paper $\mathcal{R}^*$ stands for the reflexive-transitive closure of $\mathcal{R}$. The labelled graph $\mathbf{G}$ is $\mathcal{R}$*-irreducible* (or just irreducible if $\mathcal{R}$ is fixed) if there is no $\mathbf{G}'$ such that $\mathbf{G}\ \mathcal{R}\ \mathbf{G}'$. For $\mathbf{G} \in \mathcal{G}_L$, $\mathrm{Irred}_{\mathcal{R}}(\mathbf{G})$ denotes the set of $\mathcal{R}$-irreducible graphs obtained from $\mathbf{G}$ using $\mathcal{R}$, i.e., $\mathrm{Irred}_{\mathcal{R}}(\mathbf{G}) = \{\mathbf{H} \mid \mathbf{G}\ \mathcal{R}^*\ \mathbf{H} \text{ and } \mathbf{H} \text{ is } \mathcal{R}\text{-irreducible}\}$.

Definition 3. *Let $\mathcal{R} \subseteq \mathcal{G}_L \times \mathcal{G}_L$ be a graph rewriting relation.*

1. $\mathcal{R}$ *is a* relabelling relation *if whenever two labelled graphs are in relation then the underlying graphs are equal (we say equal, not just isomorphic), i.e.:*

$$\mathbf{G}\ \mathcal{R}\ \mathbf{H} \textit{ implies that } G = H.$$

2. $\mathcal{R}$ *is* local *if it can only modify balls of radius* 1, *i.e.,* $(G, \lambda)\ \mathcal{R}\ (G, \lambda')$ *implies that there exists a vertex* $v \in V(G)$ *such that*

$$\lambda(x) = \lambda'(x) \textit{ for every } x \notin V(B_G(v)) \cup E(B_G(v)).$$

The labelled ball $(B_G(v), \lambda)$ is a support *of the relabelling relation.*

The next definition states that a local relabelling relation $\mathcal{R}$ is *locally generated* if the applicability of any relabelling depends only on the balls of radius 1.

Definition 4. *Let $\mathcal{R}$ be a relabelling relation. Then $\mathcal{R}$ is* locally generated *if it is local and the following is satisfied: For all labelled graphs (G, λ), (G, λ'), (H, η), (H, η') and all vertices $v \in V(G)$, $w \in V(H)$ such that the balls $B_G(v)$ and $B_H(w)$ are isomorphic via $\varphi\colon V(B_G(v)) \longrightarrow V(B_H(w))$ and $\varphi(v) = w$, the following three conditions*

1. $\lambda(x) = \eta(\varphi(x))$ *and* $\lambda'(x) = \eta'(\varphi(x))$ *for all* $x \in V(B_G(v)) \cup E(B_G(v))$
2. $\lambda(x) = \lambda'(x)$, *for all* $x \notin V(B_G(v)) \cup E(B_G(v))$
3. $\eta(x) = \eta'(x)$, *for all* $x \notin V(B_H(w)) \cup E(B_H(w))$

imply that $(G, \lambda)\ \mathcal{R}\ (G, \lambda')$ if and only if $(H, \eta)\ \mathcal{R}\ (H, \eta')$.

We only consider recursive relabelling relations such that the set of irreducible graphs is recursive. The purpose of all assumptions about recursiveness done throughout the paper is to have "reasonable" objects w.r.t. the computational power. By definition, local computations on graphs are computations on graphs corresponding to locally generated relabelling relations.

A sequence $(\mathbf{G}_i)_{0\leq i\leq n}$ is called an $\mathcal{R}$-*relabelling sequence* (or *relabelling sequence,* when $\mathcal{R}$ is clear from the context) if $\mathbf{G}_i \ \mathcal{R} \ \mathbf{G}_{i+1}$ for every $0 \leq i < n$ (with n being the length of the sequence). A relabelling sequence of length 1 is a *relabelling step.* The relation $\mathcal{R}$ is called *noetherian* on a graph $\mathbf{G}$ if there is no infinite relabelling sequence $\mathbf{G}_0 \ \mathcal{R} \ \mathbf{G}_1 \ \mathcal{R} \ldots$, with $\mathbf{G}_0 = \mathbf{G}$. The relation $\mathcal{R}$ is noetherian on a set of graphs if it is noetherian on each graph of the set. Finally, the relation $\mathcal{R}$ is called noetherian if it is noetherian on each graph.

3 Graph Relabelling Systems on Star-graphs

We present now graph relabelling systems as used for modelling distributed algorithms, by describing the exact form of the relabelling steps. Each step will modify a *star-graph*, i.e., a graph with a distinguished center vertex connected to all other vertices (and having no other edge besides these edges). As any ball of radius one is isomorphic to a star-graph, the support (or precondition) of any relabelling rule will be supposed to be a labelled star-graph.

Graph Relabelling Rules. A *graph relabelling rule* is a triple $r = (B_r, \lambda_r, \lambda'_r)$, where B_r is a star-graph and λ_r, λ'_r are two labellings of B_r. We refer to (B_r, λ) as the *precondition* of the rule r, whereas (B_r, λ') is referred to as the *relabelling* through r.

Let $r = (B_r, \lambda_r, \lambda'_r)$ be a relabelling rule, H an (unlabelled) graph and η, η' two labellings of H. We say that (H, η') is obtained from (H, η) by applying the rule r to the occurrence φ of B_r in H (and we write $(H,\eta)\underset{r,\varphi}{\Longrightarrow}(H,\eta')$) if the following conditions are satisfied, with v_0 denoting the center of B_r:

1. φ induces both an isomorphism from (B_r, λ_r) to $B_{(H,\eta)}(\varphi(v_0))$ and from (B_r, λ'_r) to $B_{(H,\eta')}(\varphi(v_0))$,
2. $\eta'(x) = \eta(x)$ for all $x \in (V(H) \setminus V(B_H(\varphi(v_0)))) \cup (E(H) \setminus E(B_H(\varphi(v_0))),$

In this case we also say that φ is an occurrence of the rule r in (H, η) and the image of B_r under φ is called the image of r under φ.

The relabelling relation $\underset{r}{\Longrightarrow}$ induced by the rule r is defined by letting $(H,\eta)\underset{r}{\Longrightarrow}(H,\eta')$ if there exists an occurrence φ of r in (H, η) with $(H,\eta)\underset{r,\varphi}{\Longrightarrow}(H,\eta')$.

Let $r = (B_r, \lambda_r, \lambda'_r)$ and $s = (B_s, \lambda_s, \lambda'_s)$ be two (not necessary distinct) relabelling rules and let

$$\varphi_r : \quad (B_r, \lambda_r) \quad \hookrightarrow \quad (H, \eta), \quad \varphi_s : \quad (B_s, \lambda_s) \quad \hookrightarrow \quad (H, \eta)$$

be two occurrences of r and s respectively in (H, η). We say that these two occurrences *overlap* if
(i) the images of B_r by φ_r and B_s by φ_s have a common vertex, and
(ii) either $r \neq s$ or ($r = s$ and $\varphi_r \neq \varphi_s$).

Graph Relabelling Systems. A *graph relabelling system* is a recursive set R of graph relabelling rules, such that the set of labelled star-graphs that are preconditions of a rule in R is also recursive.

The relabelling relation $\underset{R}{\Longrightarrow}$ is defined by $(G, \lambda) \underset{R}{\Longrightarrow} (G, \lambda')$ if there is a rule $r \in R$ such that $(G, \lambda) \underset{r}{\Longrightarrow} (G, \lambda')$.

Examples of graph relabelling systems are presented in [LMS99,LMZ95].

Clearly, graph relabelling systems represent locally generated relabelling relations. Conversely, any locally generated relabelling relation can be represented by a graph relabelling system.

Proposition 5. *Let $\mathcal{R}$ be a relabelling relation. Then $\mathcal{R}$ is both locally generated and a recursive relation such that the set of irreducible graphs is recursive if and only if there exists a graph relabelling system R such that $\mathcal{R}$ equals $\underset{R}{\Longrightarrow}$.*

Proof. Given a locally generated relabelling relation $\mathcal{R}$, we have to find a graph relabelling system R that generates $\mathcal{R}$.

We define

$$R = \{(B, \lambda, \lambda') \mid B \text{ is a star-graph, } (B, \lambda)\ \mathcal{R}\ (B, \lambda')\}$$

First, R is obviously recursive since $\mathcal{R}$ is. The set of preconditions of R is also recursive, since one can check wether a precondition does not belong to the set of $\mathcal{R}$-irreducible graphs. It is then straightforward to verify that R generates exactly $\mathcal{R}$ from Definition 4.

In the following, we do not discriminate between a locally generated relabelling relation and a graph relabelling system that generates it.

Generic Rules. We explain here the convention under which we will describe graph relabelling systems later. If the number of rules is finite then we will describe all rules by their preconditions and relabellings. We will also describe a family of rules by a generic rule ("meta-rule"). In this case, we will consider a generic star-graph of generic center v_0 and of generic set of vertices $B(v_0)$. Within these conventions, we will refer to a vertex v of the star graph by writing $v \in B(v_0)$. If $\lambda(v)$ is the label of v in the precondition, then $\lambda'(v)$ will be its label in the relabelling. We will omit in the description labels that are not modified by the rule. This means that if $\lambda(v)$ is a label such that $\lambda'(v)$ is not explicitly described in the rule for a given v, then $\lambda'(v) = \lambda(v)$. In all the examples of graph relabelling systems that we consider in this paper the edge labels are never changed.

With these conventions, the only point we have to care about is to verify that the set of graph relabelling rules and the set of preconditions described by the generic rule are recursive.

3.1 An Example: Computing a Spanning Tree

We consider here the problem of building a spanning tree in a graph. We assume that there exists a distinguished vertex, all vertices are initially in some neutral state (encoded by the label N) except exactly one vertex which is in an active state (encoded by the label A), and we assume that all edges have the label 0.

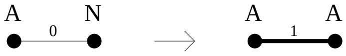
The construction of a spanning tree for a rooted network is among the most fundamental tasks to be performed. The spanning tree may be used subsequently for performing broadcast and convergecast communications.

Distributed Computation of a Spanning Tree Without Explicit Termination. At each step of the computation, an A-labelled vertex u may activate any of its neutral neighbours, say v. In that case, u keeps its label, v becomes A-labelled and the edge $\{u, v\}$ becomes 1-labelled.

Hence, several vertices may be active at the same time. Concurrent steps will be allowed provided that two such steps involve distinct vertices. The computation stops as soon as all the vertices have been activated. The spanning tree is given by the 1-labelled edges.

The algorithm may be encoded by the graph relabelling system $\mathcal{R}_1 = (L_1, I_1, P_1)$ defined by $L_1 = \{N, A, 0, 1\}$, $I_1 = \{N, A, 0\}$, and $P_1 = \{R\}$ where R is the following relabelling rule:

R :



In other words:

Spanning Tree1: **Without explicit termination**

<u>*Precondition:*</u>
- $\lambda(v_0) = A$,
- $\exists\, v \in B(v_0, 1), v \neq v_0, \lambda(v) = N$.

<u>*Relabelling:*</u>
- $\lambda'(v) := A$,
- $\lambda'(\{v_0, v\}) := 1$.

Figure 1 describes a sample computation using this algorithm. It is not difficult to observe that

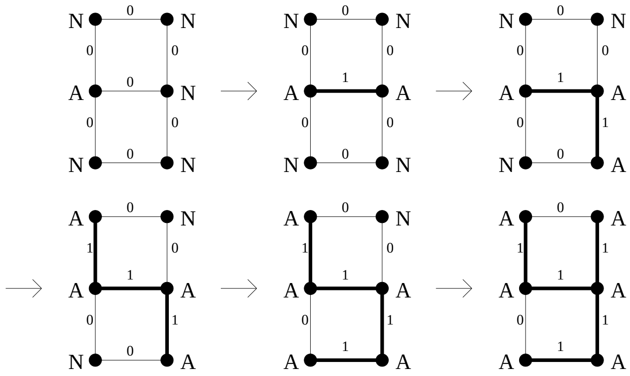


Fig. 1. Distributed computation of a spanning tree

no vertex can detect the end of the computation of the spanning tree: termination is implicit. We give in the next section a new version of this algorithm having the property of local detection of the global termination.

Distributed Computation of a Spanning Tree With Explicit Termination. As before, we assume that a unique vertex has initially label A, all other vertices having label N and all edges having label 0. The main idea is that the unique initially A-labelled vertex will keep its label until the end of the computation, while other activated vertices will be A'-labelled. As soon as an A'-labelled vertex is no longer "useful" for the computation, it will reach its final state (with label F). More precisely, we will use the following computation rules:

At each step of the computation, an active vertex (with label A or A'), say u, will act as follows:

1. If u has a N-labelled neighbour v, then u will activate this neighbour: u keeps its label, v becomes active (with label A') and the edge $\{u, v\}$ becomes 1-labelled.

2. If u is A'-labelled, has no N-labelled neighbour and is such that all its neighbours to which it is linked by a 1-labelled edge except one of these neighbours are F-labelled, then u becomes F-labelled.

At any time, the subgraph induced by the 1-labelled edges and the A- or A'-labelled vertices is a tree. Intuitively speaking, the second rule means that the vertex u is a leaf in this tree.

Thus, this algorithm runs in two phases (that may overlap): in the first phase, the tree is growing until all vertices are reached; in the second phase, it will decrease (by loosing its leaves) until it is reduced to the initially A-labelled vertex. This vertex is then able to detect that the algorithm has terminated since all its neighbours are F-labelled: in this case it takes the label END.

Figure 2 describes a sample computation using this algorithm. A complete proof of this system can be found in [LMS99].

Spanning tree1: **Adding a vertex**

Precondition:

- $\lambda(v_0) = A$,
- $\exists\, v \in B(v_0, 1), v \neq v_0, \lambda(v) = N$.

Relabelling:

- $\lambda'(v) := A'$,
- $\lambda'(\{v_0, v\}) := 1$.

Spanning tree2: **Adding a vertex**

Precondition:

- $\lambda(v_0) = A'$,
- $\exists v \in B(v_0, 1), v \neq v_0, \lambda(v) = N$.

Relabelling:

- $\lambda'(v) := A'$,
- $\lambda'(\{v_0, v\}) := 1$.

Spanning tree3: **Acknowledgement**

Precondition:

- $\lambda(v_0) = A'$,
- $\forall v \in B(v_0, 1), \lambda(v) \neq N$,
- $\forall v_1, v_2 \in B(v_0, 1)$ if $\lambda(v_1) = A'$ and $\lambda(\{v_0, v_1\}) = 1$ and $\lambda(\{v_0, v_2\}) = 1$ then $\lambda(v_2) \neq A$ and $\lambda(v_2) \neq A'$.

Relabelling:

- $\lambda'(v) := F$.

Spanning tree4: **Termination detection**

Precondition:

- $\lambda(v_0) = A$,
- $\forall v \in B(v_0, 1), \lambda(v) = F$.

Relabelling:

- $\lambda'(v_0) := END$.

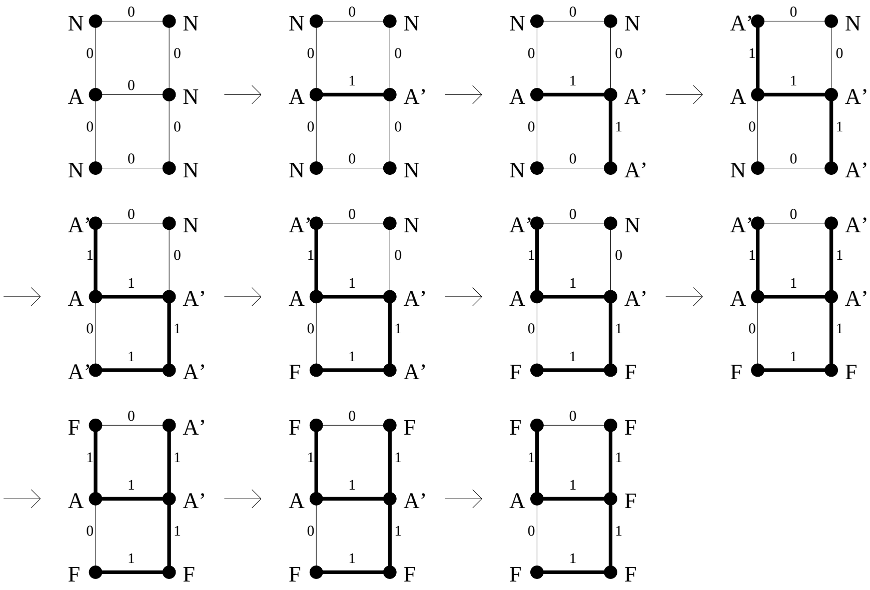


Fig. 2. Distributed computation of a spanning tree with explicit termination

.

3.2 Graph Relabelling Systems with Forbidden Contexts

The idea we develop here is to prevent the application of a relabelling rule whenever the corresponding occurrence is "included" in some special configuration, called a context. More formally, we have:

Definition 6. *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).*

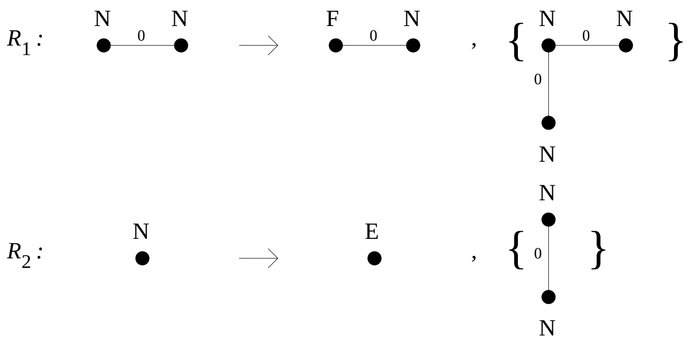
Definition 7. *A* relabelling rule with forbidden contexts *is a 4-tuple $R = (G_R, \lambda_R, \lambda'_R, F_R)$ such that $(G_R, \lambda_R, \lambda'_R)$ is a relabelling rule and F_R is a finite set of contexts of (G_R, λ_R).*

Definition 8. *A* graph relabelling system with forbidden contexts *(FCGRS for short) is a triple $\mathcal{R} = (L, I, P)$ defined as a GRS except that the set P is a set of relabelling rules with forbidden contexts.*

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 any of its forbidden contexts. More formally:

Definition 9. *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 $\varphi_i(\psi_i(G_R, \lambda_R)) = \varphi(G_R, \lambda_R)$.*

Example 10. In this example, we give a relabelling system encoding a tree election algorithm. Let $\mathcal{R}_3 = (L_3, I_3, P_3)$ be the FCGRS defined by $L_3 = \{N, F, E, 0\}$, $I_3 = \{N, 0\}$ and $P_3 = \{R_1, R_2\}$ 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 (since the forbidden context ensures that this vertex has no other $N-$labelled neighbour) by giving it a $F-$label. Thus, if (G, λ) is a labelled tree where all vertices are $N-$labelled and all edges are $0-$labelled then this cutting 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.

3.3 Graph Relabelling Systems with Priorities

The first mechanism we will consider is obtained by introducing some priority relation on the set of relabelling rules. It is obtained by introducing some priority relation, denoted $>$, on the set of relabelling rules. A relabelling rule R is applied if there exists no occurrence of a relabelling rule R' with $R' > R$ which intersects R (i.e. R and R' are applied on subgraphs having vertices in common).

Definition 11. *A* graph relabelling system *with* priorities *(PGRS for short) is a 4-tuple* $\mathcal{R} = (L, I, P, >)$ *such that* (L, 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.

We consider here the problem of building a spanning tree in a graph using the depth-first search algorithm. This algorithm is described in [Tel00, Chap. 6, p. 213]. We will give in this section a formulation using graph relabelling systems.

Suppose that all the vertices are initially in some neutral state (encoded by the label N) except exactly one vertex which is in an active state (encoded by the label A) and that all edges have the label 0. The algorithm will run in such a way that at every time exactly one vertex will be A-labelled.

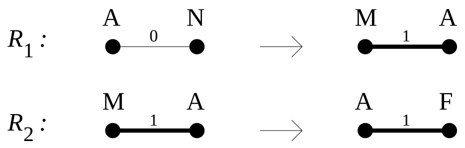
At each step of the computation, the A-labelled vertex, say u, will act as follows:

1. If u has a N-labelled neighbour v, then u will activate this neighbour: u becomes marked (with label M), v becomes active (with label A) and the edge $\{u, v\}$ becomes 1-labelled.
2. If u has no N-labelled neighbour and has a (unique) M-labelled neighbour w then u will reactivate this neighbour: u enters a final state (with label F) and w becomes A-labelled.

The computation stops as soon as none of the above computation rules may be applied (in that case, all the neighbours of the A-labelled vertex are F-labelled). The spanning tree is then given by the set of all 1-labelled edges.

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 relabellings rules. The first mechanism is obtained by introducing some priority relation, denoted $>$, on the set of relabelling rules. A relabelling rule R is applied if there exists no occurrence of a relabelling rule R' with $R' > R$ which intersects R (i.e. R and R' are applied on subgraphs having vertices in common).

The algorithm may be encoded by the graph relabelling system with priorities (PGRS for short) $\mathcal{R}_2 = (L_2, I_2, P_2, >)$ defined by $L_2 = \{N, A, M, F, 0, 1\}$, $I_2 = \{N, A, 0\}$, $P_2 = \{R_1, R_2\}$ where R_1 and R_2 are the following relabelling rules:



with the priority relation: $R_1 > R_2$.

Figure 3 describes a sample computation using this algorithm (edges with label 1 are drawn as thick edges).

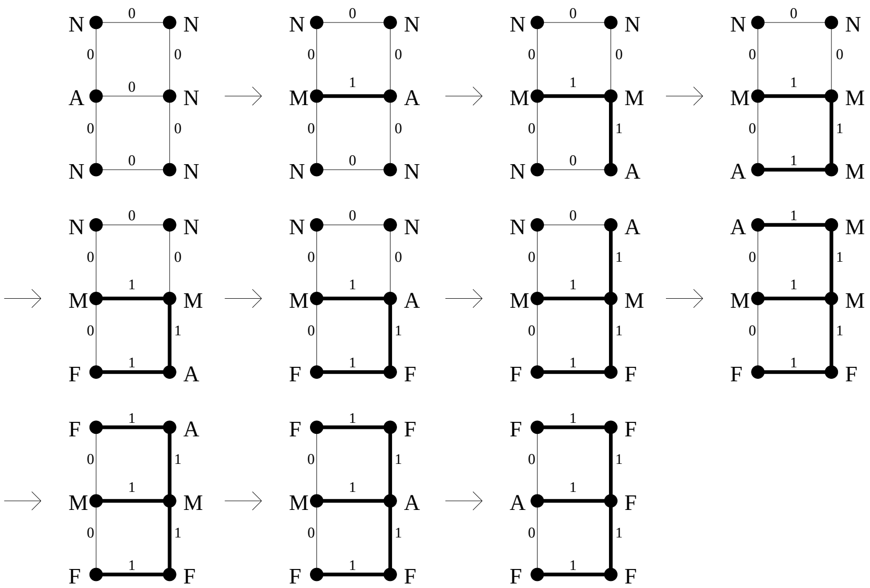


Fig. 3. Sequential computation of a spanning tree

The comparison between the expressive power of PGRSs and FCGRSs, together with some other types of GRSs, has been done in [LMS95]. 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.

3.4 Distributed Computations of Local Computations

The notion of relabelling sequence defined above obviously corresponds to a notion of *sequential* computation. Clearly, a locally generated relabelling relation allows parallel relabellings too, since non-overlapping balls may be relabelled independently. Thus we can define a distributed way of computing by saying that two consecutive relabelling steps with disjoint supports may be applied in any order (or concurrently). More generally, any two relabelling sequences such that one can be obtained from the other by exchanging successive concurrent steps, lead to the same result.

Hence, our notion of relabelling sequence associated to a locally generated relabelling relation may be regarded as a *serialization* [Maz87] of a distributed computation. This model is asynchronous, in the sense that several relabelling steps *may* be done at the same time but we do not require that all of them have to be performed. In the sequel we will essentially handle sequential relabelling sequences, but the reader should keep in mind that such sequences may be done in parallel.

References

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

[LMS99] I. Litovsky, Y. Métivier, and E. Sopena. Graph relabelling systems and distributed algorithms. In H. Ehrig, H.J. Kreowski, U. Montanari, and G. Rozenberg, editors, *Handbook of graph grammars and computing by graph transformation*, volume 3, pages 1–56. World Scientific, 1999.

[LMZ95] I. Litovsky, Y. Métivier, and W. Zielonka. On the Recognition of Families of Graphs with Local Computations. *Information and Computation*, 118(1):110–119, 1995.

[Maz87] A. Mazurkiewicz. Trace theory. In W. Brauer et al., editor, *Petri nets, applications and relationship to other models of concurrency*, volume 255 of *Lecture notes in computer science*, pages 279–324. Spinger-Verlag, 1987.

[Ros00] K. H. Rosen, editor. *Handbook of discrete and combinatorial mathematics.* CRC Press, 2000.

[Tel00] G. Tel. *Introduction to distributed algorithms.* Cambridge University Press, 2000.