Proving Distributed Algorithms by Graph Relabelling Systems: Examples of Trees in Networks with Processor Identities *

Y. Métivier, M. Mosbah, and A. Sellami

LaBRI
Université Bordeaux1 - ENSEIRB
351 Cours de la Libération
33405 - Talence, France.
{metivier, mosbah, sellami}@labri.fr

February, 2002

Abstract

In this paper, we argue that graph relabelling systems are well suited for the representation of distributed algorithms as well as for correctness and termination proofs. Various algorithms for computing distributed spanning trees serve as examples. The proofs use techniques that benefit from the high level encoding of distributed algorithms in form of rewriting systems. The result is a formal approach to describe and study distributed algorithms in a unified and simple way.

1 Introduction

Graph Relabelling Systems (GRS) have been introduced in [?] as a suitable tool for encoding distributed algorithms, for proving their correctness and for understanding their power. These systems do not modify the underlying structure of the graph on which they are applied, by only modifying the labels of vertices or edges. Each relabelling step is local, that is on a subgraph of fixed size according to some rules depending on such a subgraph.

In this model, a network is represented by a graph which vertices denote processors, and edges denote communication links. The local state of a processor (resp. link) is encoded by the labels attached to the corresponding vertex (resp. edge). Processors communicate only by asynchronous message passing. A relabelling rule is a rewriting rule which has the same underlying fixed graph for its left-hand side and its right-hand side, but with an update of the labels. Relabelling rules can be applied in parallel on the graph unless they share common edges or vertices for their left-hand sides. The relabelling is performed until no rule can be applied.

The power of relabelling systems can be increased using priorities (PGRS) or forbidden contexts (FCGRS). A priority relation (see [?, ?, ?]) is a partial order on the set of relabelling rules. In PGRS systems, the priority relation determines the order in which rules have to be applied if they overlap. In FCGRS [?], relabelling rules are equipped with forbidden context, that is graphs which disable the application of the corresponding rule if they are included in the context. All the previous examples on relabelling systems consider anonymous networks.

*This work has been supported by the TMR research network GETGRATS and by the "Conseil Régional d'Aquitaine"

In this paper, we deal with relabelling systems for named networks. Processors are assumed to have distinct identities. We show that relabelling systems can be applied for named networks. We consider the examples of computing spanning trees in such networks. By applying various types of relabelling systems, we capture various paradigms of the distributed computation of a spanning tree. These paradigms include sequential or distributed computation, local detection of the global termination, and correctness of computation.

One fundamental problem in distributed computing is the termination detection property [**?**]. That is, the existence of a vertex which decides whether the algorithm has globally terminated. On one hand, we will give examples of relabelling systems which have the termination detection property, on the other hand we will give a method which transforms a relabelling system which is initialized by exactly one vertex into anther one with the termination detection property. For the latter, we will use the well-known Dijkstra-Scholten algorithm [**?**] to detect termination. Such an algorithm fits in our framework, and can be encoded by a graph relabelling system. We exploit this simple representation to give an alternative proof of Dijkstra-Scholten algorithm. Moreover, by combining the relabelling system encoding the Dijkstra-Scholten algorithm with another representing a given distributed algorithm, we obtain a graph relabelling system encoding the latter algorithm which has the termination detection property. We illustrate this method by an example which computes a spanning tree.

We base our examples on the computation of a spanning tree in a network, because it is among the important problems in distributed computing. Trees are essential structures in various communication protocols (synchronization, deadlock resolution, information broadcasting). This problem is also closely related to the election problem. The aim of an election in a network of processors is to choose exactly one element in the set of processors : this element is called elected or leader. The leader can be used subsequently to make decisions or to centralize some information. In fact, the problem of finding a leader is reducible to the problem of finding a spanning tree. For anonymous networks, computing spanning trees by graph rewriting techniques have already been investigated in [**?**]. We focus in this paper on networks with processor identities. Let us note that all algorithms presented in this paper have been implemented and tested [**?**, **?**, **?**].

The paper is organized as follows. Section 2 recalls a few definitions and notations by giving examples. Section 3 presents three relabelling systems to compute spanning trees in a network with processor identities. In Section 4, we show how to add termination detection property to a relabelling system which is initialized by exactly one vertex.

2 Graph Relabelling Systems for Encoding Distributed Computation: Examples

In this section, we illustrate, in an intuitive way, the notion of graph relabelling systems by showing how some algorithms on networks of processors may be encoded within this framework [**?**]. Formal definitions are given in the Appendix and can be found in [**?**]. As usual, such a network is represented by a graph whose vertices stand for processors and edges for (bidirectional) links between processors. At every time, each vertex and each edge is in some particular state and this state will be encoded by a vertex or edge label. According to its own state and to the states of its neighbours, each vertex may decide to realize an elementary *computation step*. After this step, the states of this vertex, of its neighbours and of the corresponding edges may have changed according to some specific *computation rules*. Let us recall that graph relabelling systems satisfy the following requirements:

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

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

For such systems, the distributed aspect comes from the fact that several relabelling steps can be performed simultaneously on "far enough" subgraphs, giving the same result as a sequential realization of them, in any order. A large family of classical distributed algorithms encoded by graph relabelling systems is given in [?, ?]. In order to introduce definitions, we will recall in the following the relabelling systems for encoding algorithms which compute a spanning tree in an anonymous network.

2.1 Termination Detection Property

Consider a distributed system where processors communicate only by message passing. In such a system, a processor exchanges messages exclusively with its neighbours. Therefore, it is not trivial for a processor to have an up to date knowledge on the global state of the system, and to decide whether a distributed computation has finished. There are two types of termination. In an implicitly terminating algorithm, each execution is finite and in the last state of the execution each node has the correct result. However, the nodes are not aware that their state is the last one in the execution. Termination is said to be explicit in a process if that process is in a terminal configuration and its state is terminal. There were many proposals for *termination detection* algorithms: such algorithms transform implicitly into explicitly terminating algorithms. Several conditions were found to allow such algorithms and for each of these conditions a specific algorithm was given (see [?]). In this paper, by termination we mean explicit termination.

2.2 Sequential Computation of a Spanning Tree (priority relation)

We consider here the problem of building a spanning tree in a graph using the depth-first search algorithm. This algorithm is described in [?, Chap6, p213]. 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 label N) except exactly one vertex which is in an active state (encoded by label A) and that all edges have 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 relabelling 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).

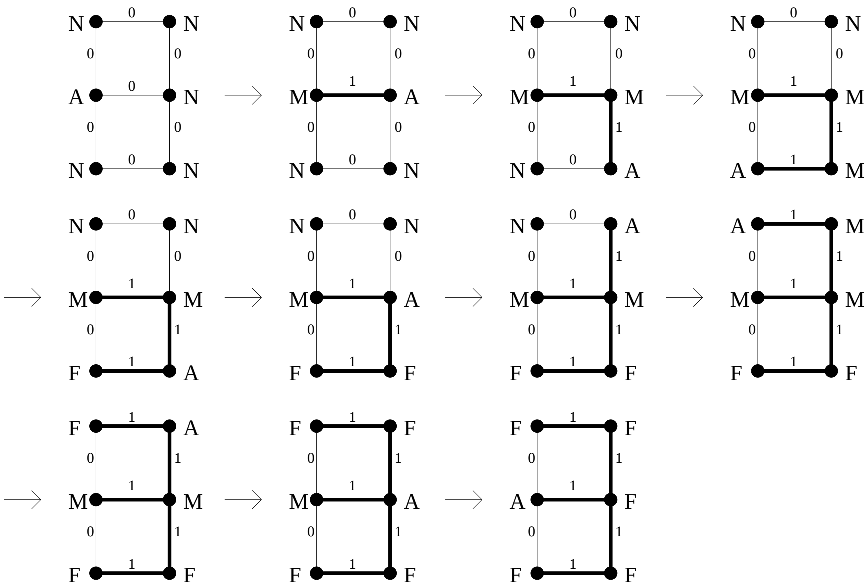
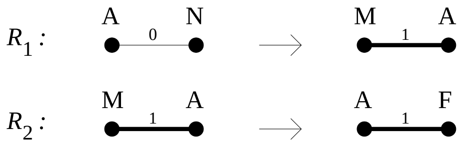


Figure 1: Sequential computation of a spanning tree

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 $L_2 = \{N, A, A', F, 0, 1\}$, $I_2 = \{N, A, 0\}$, $P_2 = \{R_1, R_2\}$ where R_1 and R_2 are the following relabelling rules:



with the priority relation: $R_1 > R_2$.

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

2.3 Distributed Computation of a Spanning Tree Without Local Detection of the Global Termination (no local control)

In the previous example, the spanning tree is computed in a strictly sequential way since at any time at most one vertex is active (with label A). We will give here a second version of this algorithm that will run in a more distributed way.

As before, we assume that a unique vertex has initially label A, all other vertices having label N and all edges having label 0.

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. As before, 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:

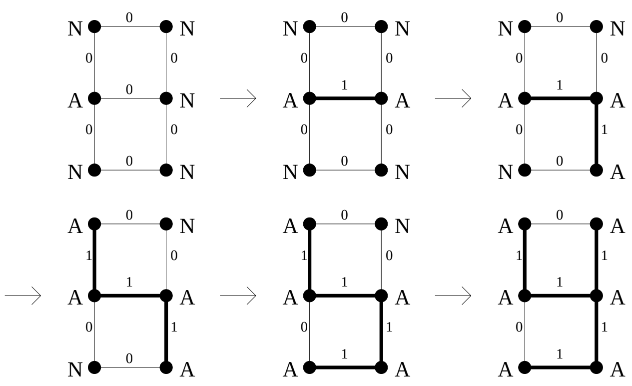


Figure 2: Distributed computation of a spanning tree

Figure 2 describes a sample computation using this algorithm. According to the previous discussion, the reader should keep in mind that some of the relabelling steps *may* be applied concurrently.

2.4 Distributed Computation of a Spanning Tree With Local Detection of the Global Termination (forbidden context)

The sequential algorithm described before is such that the active vertex is able to *locally* detect the global termination of the algorithm: if all the neighbours of the active vertex are F-labelled, then the computation is terminated. However, it is not difficult to observe that the distributed version of this algorithm does not have this property. We will give here a new distributed version of this algorithm having the property of local detection of the global 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.

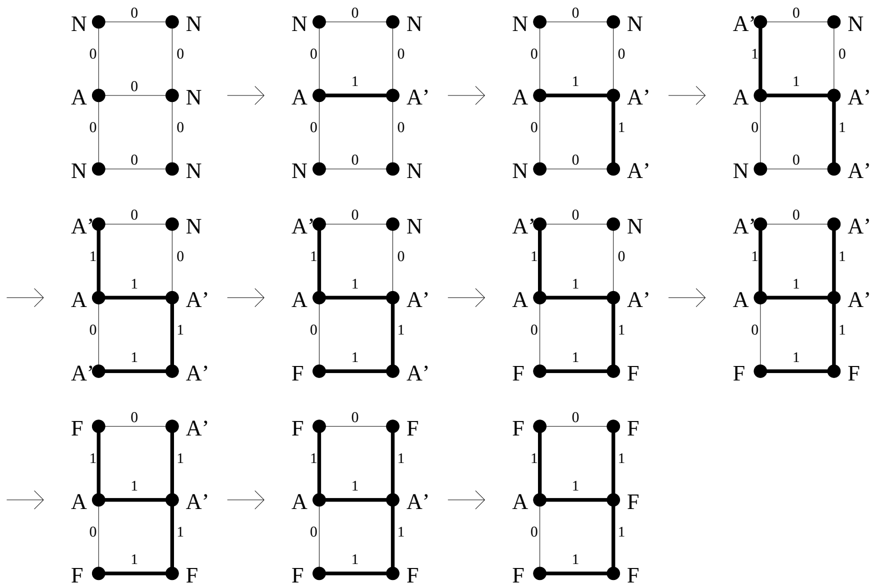
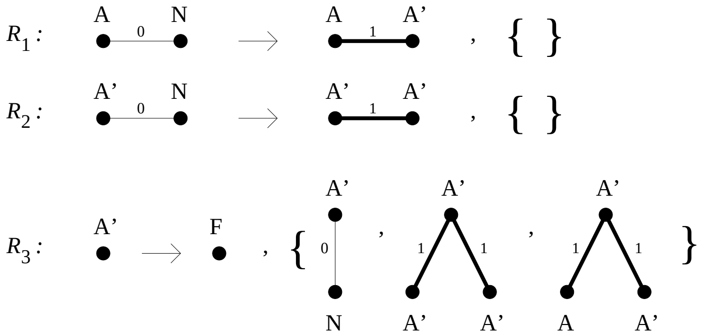


Figure 3: Distributed computation of a spanning tree with local detection of the global termination

Here we use another local control mechanism : *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 some of its forbidden contexts.

The algorithm may be encoded by the graph rewriting system with forbidden context (FCGRS for short) $\mathcal{R}_3 = (L_3, I_3, P_3)$ defined by $L_3 = \{N, A, M, F, 0, 1\}$, $I_3 = \{N, A, 0\}$, $P_3 = \{R_1, R_2, R_3\}$ where R_1, R_2 and R_3 are the following relabelling rules with forbidden contexts:



Rules R_1 and R_2 have not forbidden contexts: there is no restriction for the applications. Rule R_3 has 3 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.

Figure 3 describes a sample computation using this algorithm.

3 Computation of a Spanning Tree in a Network with Processor Identities

3.1 Techniques for Proving Distributed Computing

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 $x_1 > x_2 > \cdots > x_n > \ldots$ in S is finite), we say that $<$ is a *noetherian order*. It is compatible with $\mathcal{R}$ if there exists a mapping f from $\mathcal{G}_L$ to S such that for every $\mathcal{R}$-relabelling step $(G, \lambda) \rightarrow (G, \lambda')$ 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. For our examples, the set S will be a set $\mathbb{N}^p$ where p is an integer and the ordering relation is defined by $(x_1, \ldots, x_p) >_p (y_1, \ldots, y_p)$ which means that there exists an integer j such that $x_1 = y_1, \ldots, x_{j-1} = y_{j-1}$, and $x_j > y_j$.

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 for the computation of a spanning tree in a network with processor identities.

3.2 Distributed Computation of a Spanning Tree

Let G be a graph with n vertices. Each vertex has a unique identity; for simplicity we assume that there is a unique integer from 1 to n. Consider a labelling function $\lambda : V \cup E \longrightarrow [0..n]$, which is initialized to the identities for vertices and to 0 for edges. In order to compute a spanning tree, consider the algorithm encoded by the graph relabelling system $\mathcal{R}_4 = (L_4, I_4, P_4)$, defined by $L_4 = [0..n]$, $I_4 = [0..n]$, and $P_4 = \{R\}$ where R is the following relabelling rule:

$$R: \quad \overset{i}{\bullet}\!\overset{\alpha}{\text{———}}\!\overset{j}{\bullet} \quad \longrightarrow \quad \overset{i}{\bullet}\!\overset{i}{\text{———}}\!\overset{i}{\bullet} \quad ; \text{if } j < i$$

We prove now that this rewriting system computes a spanning tree. We show that it terminates and is correct.

Termination: Let f be the mapping from $\mathcal{G}_{L_4}$ to the set of natural integers $\mathbb{N}$ which associates with each L_4-labelled graph the sum $\sum_{\substack{v \in V, \\ \lambda(v)=k}} (n-k)$. Observing that this non-negative number strictly decreases when we apply the relabelling rule R_1 we get that $(\mathbb{N}, >)$ is a noetherian order compatible with the system $\mathcal{R}_4$. Thus, we have the following lemma:

Lemma 3.1 *The relabelling system $\mathcal{R}_4$ is noetherian.*

Correctness: It is obtained by the following lemma:

Lemma 3.2 *Let $G(V, E, \lambda)$ be a connected labelled graph such that every vertex is labelled by its identity (a unique integer of $[1, n]$ where n is the size of the graph), and every edge is labelled 0. Let $G'(V, E, \lambda')$ be a graph such that: $G(V, E, \lambda) \xrightarrow[\mathcal{R}_4]{*} G'(V, E, \lambda')$. Then the graph $G'(V, E, \lambda')$ satisfies:*

1. *All edges incident to an i-labelled vertex have labels lower or equal than i.*
2. *All vertices labelled n are connected by edges labelled n.*
3. *The subgraph induced by the i-labelled edges has no cycle ($i > 0$).*
4. *If G' is an irreducible graph obtained from G, then all vertices are labelled n.*

Proof:

1. Initially, it is true since all vertices have their identities as labels and all edges have zero as labels. Suppose that we have applied the rule k times and that property is still true. if we apply the rule for the $(k+1)^{th}$ step, only one vertex v and one edge e, say labelled j and α respectively, will change their labels to i. Since by induction, all edges incident to v have labels lower than j, and since $i > j$, then all edges, except e, have labels strictly lower than i. The new label of the edge e is i, which is equal to the labels of the vertices it links. Hence, the property is true after the rewriting step.

2. The proof is by induction. Initially, it is true since there exists only one vertex labelled n. Suppose that after k applications of the rule, the property is true. So we have a subgraph H connecting all vertices and edges labelled n. Now, suppose that we apply the rule for the $(k+1)^{th}$ step to set to n the label of a vertex (say v). It is clear that in this case, the vertex v has already a neighbour labelled n which belongs to H and that the edge linking them will be labelled n. Since, by induction, all vertices of H are labelled n and are connected by edge labelled n, vertex v, whose new label is n, will be connected to H by an edge labelled n. Hence, the property is true.

3. At the beginning, all the edges are labelled 0. We suppose that the property is true after k applications of the rule, we will show that it is true after the $(k+1)^{th}$ step. When we apply the rule to set the label of a vertex j to i, we will change the label of v and its adjacent edge to i. To apply the rule, the label of v must be lower than i and by the property 1 all its adjacent edges are lower than i so no cycle is created.

4. Assume there is a vertex with the label p, $p < n$. There is at least one vertex with the label n. As the graph is connected, there is a path linking the vertices labelled n and p. This path contains at least an edge linking a vertex labelled n with another labelled $k < n$. In such a case, the rule can be applied. This yields a contradiction.

With all these properties, we have the following result:

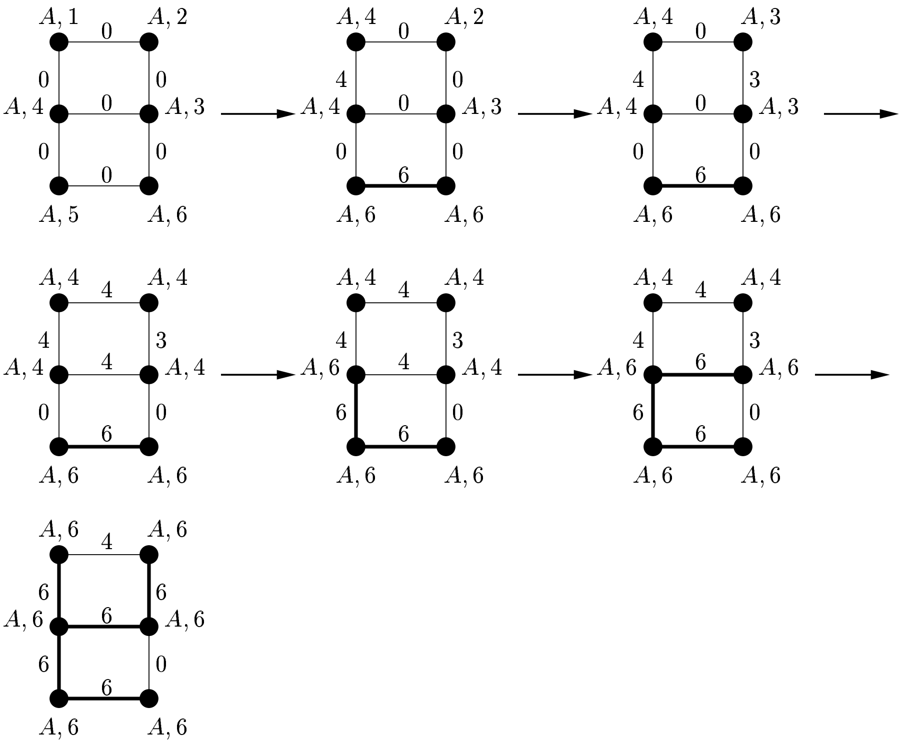
Theorem 3.1 *The relabelling system $\mathcal{R}_4$ given above encodes a distributed algorithm generating a spanning tree.*

Proof : From the previous lemmas, the tree formed by the edges labelled n is a spanning tree of the original graph.

Note that, although the algorithm finishes, it does not have the *local detection property.* That is, at the end, every vertex is labelled n and hence locally computation has terminated, but no one can decide whether the global algorithm is finished. We shall present in the following sections two other relabelling systems which have such property.

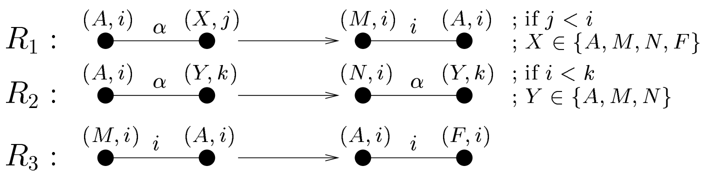
Example :

In the following, we give an example of a rewriting process that generates a spanning tree on a graph with 6 vertices and 7 edges, using the system $\mathcal{R}_4$. The bold edges are the those of the final spanning tree.



3.3 Sequential Computation of a Spanning Tree

In this section, we present a relabelling system to describe a distributed algorithm computing a spanning tree in a network with processor identities. We show that it has the termination detection property: i.e. there is at least one vertex who detect the global termination. Let G be a graph with n vertices. Each vertex has a unique identity; i.e. a unique number between 1 an n. Consider a labelling function λ, where $\lambda^{(V)} : V \to \{A, M, N, F\} \times [1..n]$ and $\lambda^{(E)} : E \to [0..n]$. Initially all edges are labelled 0 and each vertex is labelled (A, i) where i is its identity. Consider the graph relabelling system with priorities $\mathcal{R}_5 = (L_5, I_5, P_5, >)$ defined by $L_5 = \{\{A, M, N, F\} \times [1..n] \cup [0..n]\}$, $I_5 = \{\{A\} \times [1..n] \cup \{0\}\}$, $P_5 = \{R_1, R_2, R_3\}$ where R_1, R_2 and R_3 are the following relabelling rules:



with the priority relation : $R_1, R_2 > R_3$.

The proof of termination and of correctness are obtained by the following lemmas, where n is the size of G.

Lemma 3.3 *The relabelling system $\mathcal{R}_5$ is noetherian.*

Proof : This system is noetherian because executing a rewriting step implies that $(\sum_{\substack{v \in V, \\ \lambda(v)=(A,k)}} (n - k), |G|_A, |G|_M)$ is decreasing for $>_3$, where $|G|_A$ (resp. $|G|_M$) is the number of vertices with the label A (resp. M) in G.

Correctness is stated by the following lemma:

Lemma 3.4 *Let $G(V,E,\lambda)$ be a connected labelled graph such that every vertex is labelled $(A, ident)$ and every edge is labelled 0. We assume that ident is a unique integer in $[1..n]$. Let $G'(V,E,\lambda')$ be a graph such that: $G(V,E,\lambda) \xrightarrow[\mathcal{R}_5]{*} G'(V,E,\lambda')$ and let $X \in \{A,M,F\}$. Then the graph $G'(V,E,\lambda')$ satisfies:*

(I_1) *The labels of the edges incident to an (X,i)-labelled vertex are less or equal than i.*

(I_2) *There exists at most one vertex labelled (A,i), for a fixed $i > 0$.*

(I_3) *There exists at least one vertex labelled (X,j) neighbour of a (N,i)-labelled vertex with $j > i$.*

(I_4) *A (F,i)-labelled vertex has no (X,j)-labelled vertex as a neighbour, where $j \neq i$.*

(I_5) *All the (X,n)-labelled vertices are connected by n-labelled edges.*

(I_6) *The subgraph of $G(V,E)$ induced by the i-labelled edges has no cycle ($i > 0$).*

(I_7) *The (M,n)-labelled vertices form a simple path which edges are labelled n. Moreover, one of the end-vertices of this path is connected to a vertex labelled (A,n) by an edge labelled n.*

(I_8) *Let G' be an irreducible graph obtained from G, only one vertex has (A,n) as label and all the others are labelled (F,n).*

Proof :

(I_1) Let us prove this property by induction on the size of a relabelling sequence. Initially, it is true since all edges are labelled 0. Now, assume that the property is true for a graph $G(V,E,\lambda)$ obtained from the initial graph after k applications of the rules R_1 or R_2 or R_3. Let us show that the property remains true after applying R_1 or R_2 or R_3 on $G(V,E,\lambda)$, i.e. at step $k+1$. The rules R_2 and R_3 do not change neither the identities of vertices, nor the labels of edges. The rule R_1 sets the label of the edge on which it is applied to the highest identity of the incident vertex labelled A. Since all edges incident to such edge have labels lower than i (by the hypothesis of induction). The property remains true after applying R_1.

(I_2) The property is true initially. We assume it is true after k steps. if we apply R_1 for the $(k+1)^{th}$ step, we have only one (A,i), the first will change to (M,i); similarly, if we apply R_3 for the $(k+1)^{th}$ step, the first (A,i) will change to (F,i) and the (M,i) will change to (A,i). We will have only one (A,i)-labelled vertex. If we apply R_2 for the $(k+1)^{th}$ step, an (A,i) will change to a neutral state (N,i) so the property is still true after applying R_2. Hence, the property remains true.

(I_3) Initially, I_3 is true. Suppose that it is true after k steps. In the $(k+1)^{th}$ step, if we apply R_3 the property is still true since we do not change the identities of the vertices; if we apply R_1 on a vertex labelled (A,i) and on a vertex with the label (N,j), with $j < i$, the second label will change to (A,i) and the first will change to (M,i), so we do not have a (N,j)-labelled vertex. If (X,j) is a neighbour of a (N,k)-labelled vertex, as $j > k$ in the k^{th} step and as $i > j$ so $i > k$ and the property remains true after the application of the rule R_1. Applying R_2 , a vertex labelled (A,i) will change to (N,i). As (A,i) is a neighbour of a vertex with the label (Y,k), where $k > i$, the property is still true.

(I_4) It is a direct consequence of the priority rule: if a rewriting step labels a vertex (A,i) with (F,i), the priority rule implies that (A,i) has no (X,j)-labelled neighbours where $j \neq i$.

(I_5) Initially the property is true since there is only one vertex labelled (A, n). We suppose that it is true after k rewritings, we will prove that it is true for the $(k+1)^{th}$ step. After k times, all (X, n) vertices are connected by n-labelled edges. Only the application of R_1 on an edge incident to a vertex v_1 labelled (A, n) will add a new (A, n)-vertex v_2 that is v_1 will became (M, n)-labelled, $v_2 : (A, n)$-labelled and the edge between will be n-labelled. Since by induction v_2 is connected to other (X, n)-labelled vertices only by n labelled edges, and since the new label of the edge (v_1, v_2) is n, then the property is true after applying rule R_1. Note that in all other cases, the edge labels do not change.

(I_6) Initially, all the edges are labelled 0 so there is no cycle. We suppose that the property is true after k applications of the rule, we will show that it is true after the $(k+1)^{th}$ step. When we apply the rule R_2 or R_3, we will not change the labels of edges so the property remains true. Applying R_1 we will change the label of a vertex v with a label lower than i and his adjacent edge to i. By the property (I_1) all the adjacent edges of v are lower than i so we will not create a cycle. Hence, the property is true.

(I_7) At the beginning the property is true since no vertex is labelled (M, n) and only one vertex is labelled (A, n). Now, suppose the property true at the k^{th} step of the rewriting sequence and let us prove that it is true at the $(k+1)^{th}$ step. If the rule R_1 is applied on the (A, n)-labelled vertex then this vertex will be labelled (M, n). The neighbour involved by the application of this rule will be labelled (A, n). By induction, all the (M, n)-labelled vertices form a path in the k^{th} step, hence the property remains true in the $(k+1)^{th}$ step. If the rule R_3 is applied on the vertices labelled (A, n) and (M, n), the former will be labelled (A, n) and the latter (F, n). Again, by induction the property remains true. All other cases of applying rules R_1 and R_2 will not generate (M, n) or (A, n) labelled vertices. The application of R_2 will no change the vertices labelled (M, n) nor the (A, n)-labelled vertex since there is no identity greatest than n. Hence, the property remains true.

(I_8) By (I_7), the irreducible graph G' has no (M, n)-labelled vertex. The graph G' has at least one (F, n)-labelled vertex adjacent to (A, n)-labelled vertex which is unique. As G' is irreducible, the vertex labelled (A, n) has no (X, i)-labelled vertices as neighbours. The result is now a consequence of (I_4).

Theorem 3.2 *The system $\mathcal{R}_5$ computes a spanning tree. This algorithm has the local detection property.*

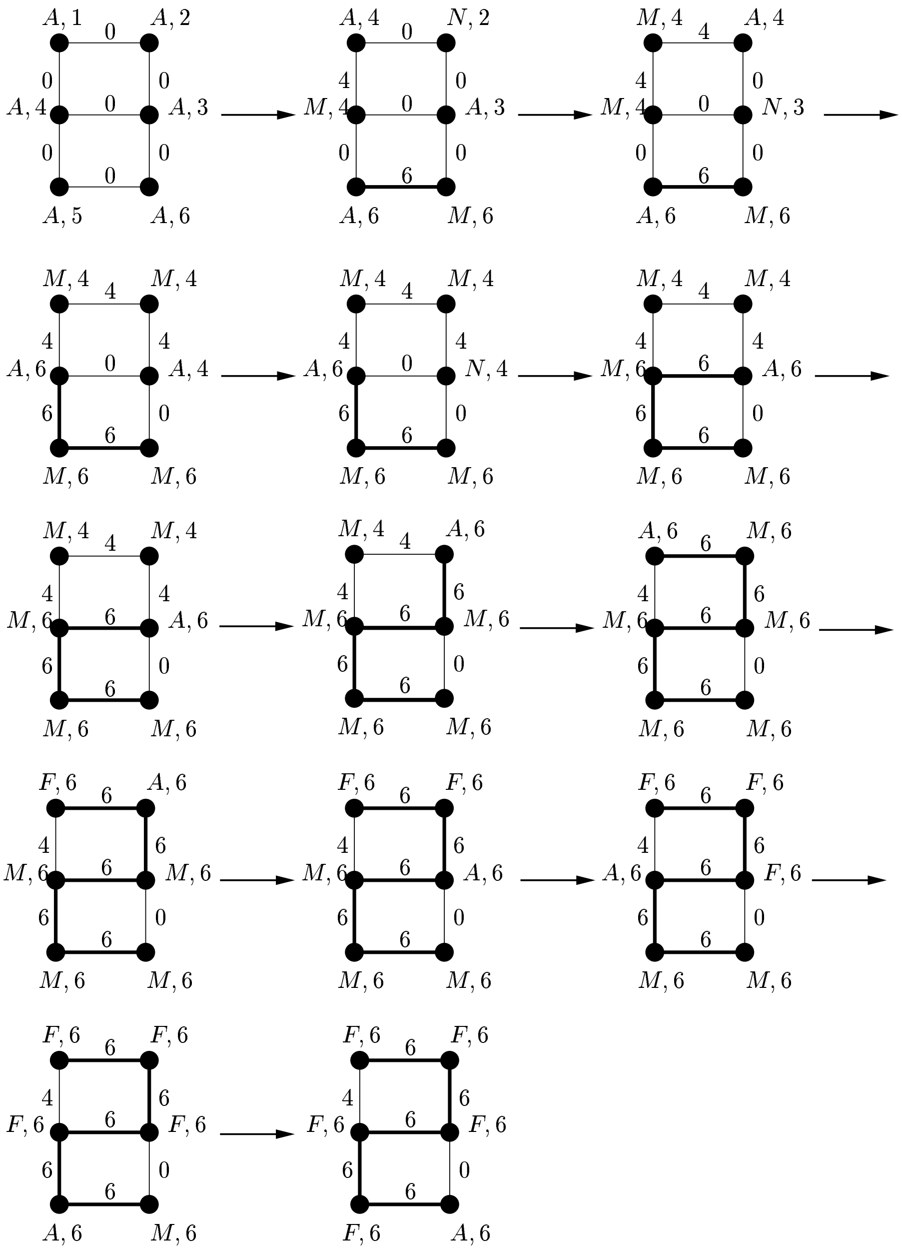
With the properties $(I_5), (I_6), (I_7)$ and (I_8), we deduce that at the end, we have a spanning tree with (A, n) as root and all the other vertices are labelled with (F, n). The edges of this tree are labelled n.

Remark :

In this algorithm, many trees are computed distributively. However, each of these trees is computed in a sequential way.

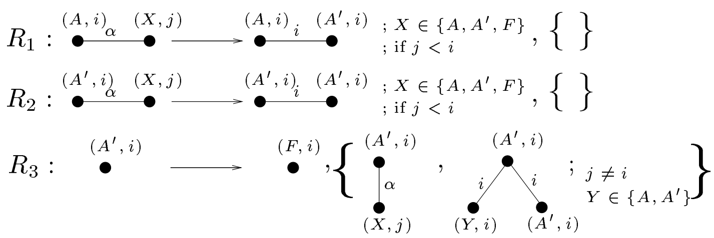
Example:

In the following, we give the same initial graph as the previous section, but executing the rules R_1, R_2 and R_3. Note that, from step eight, the relabelling becomes sequential; only one vertex is labelled $(A, 6)$.



3.4 Distributed Computation of a Spanning Tree with Local Detection of the Global Termination

Now, we present a relabelling system encoding an algorithm computing a spanning tree which has both properties: distribution and local detection. We use the forbidden context local control mechanism. Let G be a graph with n vertices. Each vertex has a unique identity; i.e. a unique number between 1 an n. Consider a labelling function λ, where $\lambda^{(V)} : V \to \{A, A', F\} \times [1..n]$ and $\lambda^{(E)} : E \to [0..n]$. Initially all edges are labelled 0 and each vertex is labelled (A, i) where i is its identity. The graph rewriting system is $\mathcal{R}_6 = (L_6, I_6, P_6)$ defined by $L_6 = \{\{A, A', F\} \times [1..n] \cup [0..n]\}$, $I_6 = \{\{A\} \times [1..n] \cup \{0\}\}$, $P_6 = \{R_1, R_2, R_3\}$ where R_1, R_2 and R_3 are the following relabelling rules with forbidden contexts:



Rules R_1 and R_2 have not forbidden contexts: there is no restriction for the applications. Rule R_3 has 3 forbidden contexts (the subgraphs between brackets on the left-hand side). The unique vertex of the left-hand side of the rule R_3 is associated with the top vertex of its forbidden contexts. We will show that this rewriting system terminates and is correct. Termination is expressed by the lemma:

Lemma 3.5 *The rewriting system $\mathcal{R}_6$ defined above is noetherian.*

Proof: consider this couple $(\sum_{\substack{v \in V, \\ \lambda(v)=(X,k)}} (n-k), |G|_{A'})$, where $X \in \{A, A'\}$. It is decreasing for $>_2$, so this system is noetherian.

Correctness is given by:

Lemma 3.6 *Let $G(V, E, \lambda)$ be a connected labelled graph such that every vertex v is labelled (A, i) and every edge is labelled 0, where $i \in [1..n]$ is the identity of v, n is the size of G. Let $G'(V, E, \lambda')$ be a graph such that: $G(V, E, \lambda) \xrightarrow[\mathcal{R}_6]{*} G'(V, E, \lambda')$ and let $X \in \{A, A', F\}$. Then the graph $G'(V, E, \lambda')$ satisfies:*

(I_1) *The labels of the edges incident to an (X, i)-labelled vertex are less or equal than i.*

(I_2) *There exists at most one vertex labelled (A, i), $i > 0$.*

(I_3) *There is exactly one vertex with the label (A, n).*

(I_4) *A (F, i)-labelled vertex has only (X, i)-labelled vertices as neighbours.*

(I_5) *All the (X, n)-labelled vertices are connected by n-labelled edges.*

(I_6) *The (A, n) and (A', n) labelled vertices are connected by n-labelled edges.*

(I_7) *The subgraph of $G(V, E)$ induced by the i-labelled edges has no cycle.*

(I_8) *Let G' be an irreducible graph obtained from G, only one vertex has (A, n) as label and all the others are labelled (F, n).*

Proof :

(I_1) The proof is by induction. I_1 is true initially. Suppose that it is true after k steps. In the $(k+1)^{th}$ step, if we apply R_3 there is no change of the labels of edges so the property is still true; if we apply R_1 or R_2 , an edge label will change to i, the incident vertices are labelled (X, i) and the others will not change. Therefore, the property is still true.

(I_2) Initially, it is true. Assume that it is true after k steps. In the $(k+1)^{th}$ step, if we apply any of the rules, no new labels (A, i) appears. Hence the property is true.

(I_3) At the beginning the property is true since only one vertex is labelled (A, n). We suppose that the property is true after k relabelling steps. We will prove that the property remains true at the $(k+1)^{th}$ step. In the k^th step, we have exactly one vertex labelled (A, n). If we apply R_1 on this vertex, we can apply it only to change the neighbours of this node since it has the greatest value which is n. Hence, the property is still true. For the rule R_2, we

can not apply them on this vertex for the same reason. So the property remains true.
As n is the size of the graph so the greatest identity, we can not apply the rule R_1 nor R_2 to change the label of the vertex (A, n). Hence, there is usual one vertex with the label (A, n).

(I_4) It is the consequence of the forbidden contexts; we can not apply R_3 since there is a neighbour that is labelled with (X, j) where $j \neq i$.

(I_5) The proof is by induction. Initially, the property is true since only one vertex is labelled (A, n). Suppose the property true after k relabelling steps and assume that G_k is the subgraph of G induced by the (X, n)-labelled vertices. We will prove that the property remains true at the $(k+1)^{th}$ step. Without loss of generality, we consider the case where a new label (X, n) appears, because in all other cases the property is true. Thus, if the rule R_1 or R_2 is applied on some vertex u of G_k and on a neighbour v not in G_k, the new label of v will be (A', n). The edge linking u and v will have n as label. Hence, the property is still true.

(I_6) By (I_5), the vertices labelled $(A, n), (A', n)$ and (F, n) are connected by n-labelled edges. Let us prove now that (A, n) and (A', n) are connected by n-labelled edges. Initially, it is true since there exist only one vertex labelled (A, n). Suppose the property true after k relabelling steps and assume that G_k is the subgraph of G induced by (A, n) and (A', n) labelled vertices. We will prove that the property remains true at the $(k+1)^{th}$ step. If one of the rules R_1 or R_2 is applied on some vertex u of G_k and on a neighbour v not in G_k, the new label of v will be (A', n). The edge linking u and v will have n as label. So the new vertex v belong to G_k. Applying R_3, we will remove a vertex u from G_k. The new graph G_{k+1} is connected since u is a leaf. Hence the property remains true.

(I_7) At the beginning the property is true since all the edges are labelled 0 so there is no cycle. We suppose that the property is true after k relabelling steps and let G_i be the subgraph of G induced by the i-labelled edges. We will prove that the property remains true at the $(k+1)^{th}$ step. The rule R_3 will not change the label of the edges so the property remains true. But if one of the rule R_1 or R_2 is applied on a vertex u labelled (A, i) or (A', i) of G_i and on a neighbour v labelled (X, j), where $X \in \{A, A', F\}$ and $j < i$, the label of the vertex v will be (A', i) and the edge linking u and v will have i as label. By the property (I_1), all the adjacent edges of v are lower than i at the k^{th} step so no edge is in G_i, therefore no cycle will be created. Hence, the property is still true.

(I_8) By (I_3), there is exactly one vertex labelled (A, n). Its neighbours are labelled (A', n) or (F, n) since if there is a vertex labelled (X, i) where $i < n$, we can apply the rule R_1 or R_2. We suppose that there is at least one vertex labelled (A', n), by (I_5) and (I_6), the graph G_n induced by the vertex labelled (A, n) and the vertices labelled (A', n) form a tree. This contradicts the fact that G' is an irreducible graph. So there is no (A', n)-labelled vertices. Hence, (A, n) has only (F, n)-labelled vertices as neighbours.

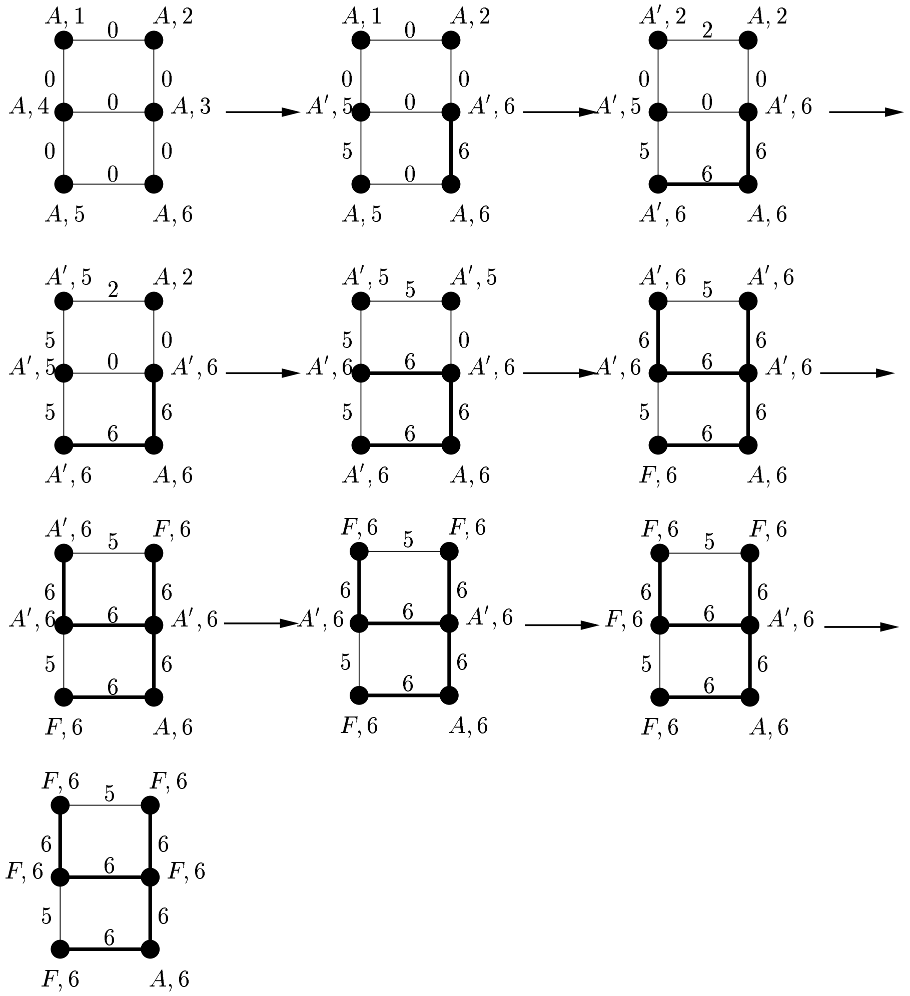
Finally, we have :

Theorem 3.3 *The system $\mathcal{R}_6$ encodes a distributed algorithm which computes a spanning tree. Moreover, it has the termination detection property.*

Proof: With properties (I_5), (I_7) and (I_8), we deduce that at the end, we have a spanning tree with (A, n) as root and all other vertices are labelled with (F, n). The edges of this tree are labelled with n. As we proved that only one vertex has (A, n) as label, it detects the global termination as soon as all its neighbours are labelled (F, n).

In the following, we give an example with the same initial labelled graph than the previous sections.

Example :



4 Termination Detection of a Graph Relabelling System

In this section, we give a solution to make a graph relabelling system detect locally the global termination. We use the well-known Dijkstra-Scholten termination algorithm [**?**]. More precisely, We show how to transform a graph relabelling system into another one which has the termination detection property. The method we present can be applied for any relabelling system which is initialized by exactly one vertex. We will illustrate it on the example of section 2.3. We start by a description of the termination algorithm [**?**]. Then, we express it by a relabelling system, and finally we use it as a tool to detect locally the global termination.

4.1 The Dijkstra-Scholten Termination Detection Algorithm

Consider a network of processors modeled by a graph. Each processor, represented by a vertex, is always in one of two states *active* or *passive*. The processors communicate solely by messages (which are assumed to be transmitted instantaneously). Some of these messages are referred to as activation messages; they may set the receiver into an active state. An active processor maintains a count of the number of activations it has sent which have not yet been acknowledged. Initially, exactly one processor is active; then it activates its neighbours. The set of active nodes form a dynamic tree; where the root is the initial active node, and a node is the father of the nodes it activates. For each node, the transition from the active state to the passive state is an internal transition. A leaf node is deleted from the tree if it is passive and has no active descendants; i.e. it has received acknowledgments for the activation messages it has sent. In this case, it sends a signal to acknowledge the activation to its father who will decrease its own count of active sons. Note that the tree may grow and shrink dynamically. We will call this tree the *control tree*. The computation terminates when the root becomes passive and has no active descendants.

We give now a relabelling system to encode Dijkstra-Scholten algorithm. To do so, we equip each vertex of the graph with a counter (say sc) and with a flag describing its activity status (say a value in $\{Ac, Pa\}$). Hence, a vertex will be labelled by a triple (X, Y, sc), where X is its label N, A, or A'; Y is its status activity, and sc the counter described above. An edge will be labelled by a label in $\{Ac, Pa\}$ indicating whether on not it belongs to the control tree. Initially, each edge is labelled Pa and each node is labelled $(N, Pa, 0)$; a distinguished node is labelled $(A, Ac, 0)$ which will be the root of the control tree. This node initializes the algorithm. Let $\mathcal{R}_7 = (L_7, I_7, P_7)$ be the relabeling system defined by $L_7 = \{A, A', N\} \times \{Ac, Pa\} \times [0..n] \cup \{Pa, Ac\}$, $I_7 = \{(A, Ac, 0) \cup (N, Pa, 0) \cup \{Pa\}\}$, and P_7 consisting of the following relabelling rules:

$$R_1: \begin{array}{c}(X,Ac,sc)\\ \bullet\\ \Big|\, Pa\\ \bullet\\ (N,Pa,0)\end{array} \longrightarrow \begin{array}{c}(X,Ac,sc+1)\\ \bullet\\ \Big|\, Ac\\ \bullet\\ (A',Ac,0)\end{array} \quad ; X \in \{A, A'\}$$

$$R_2: \begin{array}{c}(X,Ac,sc)\\ \bullet\end{array} \longrightarrow \begin{array}{c}(X,Pa,sc)\\ \bullet\end{array} \quad ; X \in \{A, A'\}$$

$$R_3: \begin{array}{c}(X,Ac,sc)\\ \bullet\\ \Big|\, Pa\\ \bullet\\ (Y,Pa,sc')\end{array} \longrightarrow \begin{array}{c}(X,Ac,sc)\\ \bullet\\ \Big|\, Pa\\ \bullet\\ (Y,Ac,sc')\end{array} \quad \begin{array}{l}; X, Y \in \{A, A'\}\\ ; sc' \neq 0\end{array}$$

$$R_4: \begin{array}{c}(X,Y,sc)\\ \bullet\\ \Big|\, Ac\\ \bullet\\ (A',Pa,0)\end{array} \longrightarrow \begin{array}{c}(X,Y,sc-1)\\ \bullet\\ \Big|\, Pa\\ \bullet\\ (N,Pa,0)\end{array} \quad \begin{array}{l}; X \in \{A, A'\}\\ ; Y \in \{Ac, Pa\}\end{array}$$

$$R_5: \begin{array}{c}(A,Pa,0)\\ \bullet\end{array} \longrightarrow \begin{array}{c}End\\ \bullet\end{array}$$

Note that R_3 allows an internal node of the control tree which is in a passive state to be activated by one of its active neighbours. This is a part of the Dijkstra-Scholten algorithm.

Theorem 4.1 *The relabelling system $\mathcal{R}_7$ encodes Dijkstra-Scholten algorithm. If the label End appears, then each vertex is passive. Conversely, if each vertex is passive then eventually the label End will appear.*

Proof: The proof follows from the invariants given below.

(I_1) sc+1 is the number of edges labelled Ac incident to the vertex labelled (X, Y, sc), where $X \in \{A', N\}$ and $Y \in \{Ac, Pa\}$. sc is the number of edges labelled Ac incident to the vertex labelled (A, Y, sc), where $Y \in \{Ac, Pa\}$.

(I_2 All edges incidents to a $(N, Pa, 0)$-labelled vertex are labelled Pa.

(I_3) A (A', Y, sc)-labelled vertex has at least one Ac-labelled incident edge, where $Y \in \{Ac, Pa\}$.

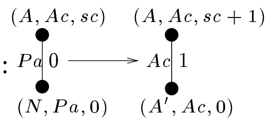
(I_4) The subgraph induced by (X, Y, sc)-labelled vertices is connected by Ac-labelled edges, where $X \in \{A, A'\}$ and $Y \in \{Ac, Pa\}$.

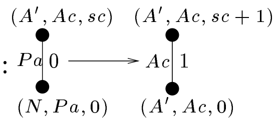
(I_5) The subgraph induced by the Ac-labelled edges has no cycle.

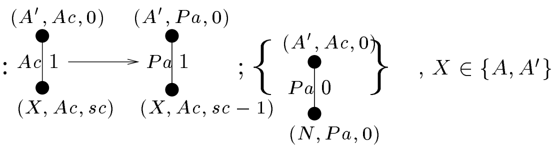
Remark. Encoding the Dijkstra-Scholten algorithm by a graph relabelling system makes it easy to read, to understand and particularly to prove in a unified and simple way as we showed for the previous algorithms. Moreover, expressing this algorithm in form of a graph relabelling system will be useful to apply the Dijkstra-Scholten termination technique to an algorithm encoded by a graph relabelling system. In fact, this turns out to be a combination of two graph relabelling systems as we will illustrate this idea by the following example.

4.2 Adding Dijkstra-Scholten Termination Detection Algorithm to a Graph Relabelling System

Now, given a relabelling system encoding a distributed algorithm which is initialized by a one vertex. To detect termination by the previous technique, one has to embed the relabelling computation into the control tree. We have just to combine the graph relabelling system with the previous one. A node will be passive if it has locally terminated (no rule can be applied on its context). The rules of the relabelling system are mapped using the new triple labelling. In order to keep up with the control tree of the termination algorithm, a relabelling rule encoding local termination and acknowledgment must be added. The relabelling system of Section 2.3 combined with Dijkstra-Scholten termination techniques becomes the relabelling system with forbidden contexts $\mathcal{R}_8 = (L_8, I_8, P_8)$, defined by $L_8 = \{A, N\} \times \{Ac, Pa\} \times [0..n] \cup \{0,1\} \times \{Ac, Pa\}$, $I_8 = \{(A, Ac, 0) \cup (N, Pa, 0) \cup \{(0, Pa)\}\}$, and P_8 consisting of the following relabelling rules:

R_1 : 

R_2 : 

R_3 : 

Note that we give here a simplified system of the combination of the relabelling system of Section 2.3 with that of the Dijkstra-Scholten relabelling system. It is a relabelling system with forbidden context. Rule R_1, which is similar to the original rule, maintains the count and the status of activated vertices. However, rule R_3 is added to collect information about nodes who locally terminated, and has received acknowledgments from all activated descendants ($sc = 0$). A node locally terminates if it has no neighbour labelled N which has not been yet active. Such a node, will be set in passive status, and will send an acknowledgment to its father if its counter is null. In the relabelling rule, we just decrement the counter sc of the father. The whole computation terminates if the root becomes passive and has no active descendants.

Theorem 4.2 *The relabelling system $\mathcal{R}_8$ encodes a distributed algorithm which computes a spanning tree. This algorithm has the property of local detection of the termination.*

Proof:

The proof is similar to the previous systems. We give the invariants to prove correctness and termination. Let $G(V, E, \lambda)$ be a connected labelled graph such that a distinguished vertex is labelled $(A, Ac, 0)$, all other vertices are labelled $(N, Pa, 0)$ and every edge is labelled $(Pa, 0)$. Let

$G'(V,E,\lambda')$ be a graph such that: $G(V,E,\lambda) \xrightarrow[\mathcal{R}_8]{*} G'(V,E,\lambda')$. Then the graph $G'(V,E,\lambda')$ satisfies:

1. All edges incident to a $(N,Pa,0)$-labelled vertex are labelled $(0,Pa)$.
2. A (A,X,sc)-labelled vertex has at least $(1,Y)$-labelled incident edge, ($sc \neq 0$, $X,Y \in \{Ac,Pa\}$).
3. The subgraph induced by the edges labelled $(1,X)$ has no cycle, ($X \in \{Ac,Pa\}$).
4. The (A,Ac,sc) and (A',Ac,sc') labelled vertices are connected by $(1,Ac)$-labelled edges.
5. More generally, all the (A,X,sc) and (A',Y,sc') labelled vertices are connected by $(1,Z)$-labelled edges, ($X,Y,Z \in \{Ac,Pa\}$).
6. A $(A,Pa,0)$-labelled vertex has no $(N,Pa,0)$-labelled neighbours.
7. If G' be an irreducible graph obtained from G then all vertices are $(A,Pa,0)$-labelled.

The detection of termination of the algorithm occurs as soon as the counter (sc) of the root becomes zero (0).

Let us note that all the algorithms presented in this paper have been implemented and tested in ViSiDiA[**?**, **?**]. ViSiDiA is a tool to simulate, visualize and animate distributed algorithms. It is based on graph relabelling systems.

References

[1] M. Bauderon, S. Gruner, Y. Métivier, M. Mosbah, and A. Sellami. Visualization of distributed algorithms based on labeled rewriting systems. In *Second International Workshop on Graph Transformation and Visual Modeling Techniques, Crete, Greece*, July 12-13, 2001.

[2] M. Bauderon, Y. Métivier, M. Mosbah, and A. Sellami. From local computations to asynchronous message passing systems. Technical Report RR-1271-02, LaBRI, 2002.

[3] M. Billaud, P. Lafon, Y. Métivier, and E. Sopena. Graph rewriting systems with priorities. *Lecture notes in computer science*, 411:94–106, 1989.

[4] E.W. Dijkstra and C.S. Sholten. Termination detection for diffusing computations. *Information Processing Letters*, 11(1):1–4, 1980.

[5] I. Litovsky and Y. Métivier. Computing trees with graph rewriting systems with priorities. *Tree automata and languages*, pages 115–139, 1992.

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

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

[8] 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.

[9] 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.

[10] M. Mosbah and A. Sellami. Visidia: A tool for the visualization and similation of distributed algorithms. *http://www.labri.fr/visidia/*.

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

A Formal definitions of graph relabelling systems

An L*-labelled graph* is a graph whose vertices and edges are labelled with labels from a possibly infinite alphabet L. It will be denoted by $G(V,E,\lambda)$, where G is a graph and $\lambda : V(G) \cup E(G) \to L$ is the *labelling function*. The graph G is called the *underlying graph* of $G(V,E,\lambda)$, and λ is a *labelling* of G. The class of L-labelled graphs will be denoted by $\mathcal{G}_L$.

Let $G(V,E,\lambda)$ and $G'(V',E',\lambda')$ be two labelled graphs; $G(V,E,\lambda)$ is a subgraph of $G'(V',E',\lambda')$, denoted by $G(V,E,\lambda) \subseteq G'(V',E',\lambda')$, if G is a subgraph of G' and λ is the restriction of λ' to $V(G) \cup E(G)$.
A mapping φ: $V(G) \cup E(G) \to V(G') \cup E(G')$ is a homomorphism of $G(V,E,\lambda)$ to $G'(V',E',\lambda')$ if φ is a homomorphism of G to G' which preserves the labelling, that is, $\lambda'(\varphi(x)) = \lambda(x)$ for every $x \in V(G) \cup E(G)$. An *occurrence* of $G(V,E,\lambda)$ in $G'(V',E',\lambda')$ is an isomorphism φ between $G(V,E,\lambda)$ and some subgraph $H(N,L,\eta)$ of $G'(V',E',\lambda')$.

A graph relabelling rule is a triple $R = (G_R, \lambda_R, \lambda'_R)$ such that $G_R(V_R,E_R,\lambda_R)$ and $G_R(V_R,E_R,\lambda'_R)$ are two labelled graph. The labelled graph $G_R(V_R,E_R,\lambda_R)$ is the *left-hand side* and the labelled graph $G_R(V_R,E_R,\lambda'_R)$ is the *right-hand side* of R. A graph relabelling system is a triple $\mathcal{R} = (L,I,P)$ where L is a set of labels, I a subset of 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 $\mathcal{P}$ and φ is both an occurrence of $G_R(V_R,E_R,\lambda_R)$ in $G(V,E,\lambda)$ and an occurrence of $G_R(V_R,E_R,\lambda'_R)$ in $G(V,E,\lambda')$. In other words, the labelling λ' of G is obtained from λ by modifying all the labels of the elements of $\varphi(G_R(V_R,E_R,\lambda_R))$ according to the labelling λ'_R. Such a relabelling step will be denoted by $G(V,E,\lambda) \underset{\mathcal{R}}{\longrightarrow} G(V,E,\lambda')$. The notion of computation then corresponds to the notion of relabelling sequence. A $\mathcal{R}-$*relabelling sequence* will be denoted by $G(V,E,\lambda) \xrightarrow[\mathcal{R}]{*} G(V,E,\lambda')$.

The computation stops when the graph is labelled in such a way that no relabelling rule can be applied. A labelled graph $G(V,E,\lambda)$ is said to be $\mathcal{R}-$*irreducible* (or irreducible when it is clear from the context) if there exists no occurrence of $G_R(V_R,E_R,\lambda_R)$ in $G(V,E,\lambda)$ for every relabelling rule R in P.

A graph relabelling system *with priorities* 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 $\mathcal{P}$ and φ is both an occurrence of $G_R(V_R,E_R,\lambda_R)$ in $G(V,E,\lambda)$ and an occurrence of $G_R(V_R,E_R,\lambda'_R)$ in $G(V,E,\lambda')$ 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.

Let $G(V,E,\lambda)$ be a labelled graph. A context of $G(V,E,\lambda)$ is a triple (H,μ,ψ) such that $H(N,L,\mu)$ is a labelling graph and ψ an occurrence of $G(V,E,\lambda)$ in $H(N,L,\mu)$.

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 $G_R(V_R,E_R,\lambda_R)$.

A graph relabelling system with forbidden contexts is a triple $\mathcal{R} = (L,I,P)$ defined as a graph relabelling system 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 and φ is both an occurrence of $G_R(V_R,E_R,\lambda_R)$ in $G(V,E,\lambda)$ and an occurrence of $G_R(V_R,E_R,\lambda'_R)$ in $G(V,E,\lambda')$, and for every context (H_i,μ_i,ψ_i) of $G_R(V_R,E_R,\lambda_R)$, there is no occurrence φ_i of $H_i(N_i,L_i,\mu_i)$ in $G(V,E,\lambda)$ such that $\varphi_i(\psi_i(G_R)) = \varphi(G_R)$. Formal definitions and examples on relabelling systems may be found in [**?**, **?**, **?**, **?**, **?**].