Self-stabilizing global computations with r-operators

Bertrand Ducourthial and Sébastien Tixeuil
Laboratoire de Recherche en Informatique (bât. 490)
Université Paris-Sud, 91405 ORSAY Cedex FRANCE
email: {ducourth,tixeuil}@lri.fr

July 9, 1998

Abstract:

This papers describes a parametrized distributed algorithm applicable to any directed graph topology. The function parameter of our algorithm is instantiated to produce distributed algorithms for both fundamental and high level applications, such as shortest path calculus and depth-first-search tree construction. Due to fault resilience properties of our algorithm, the resulting protocols are self-stabilizing at no additional cost. Self-stabilizing protocols can resist transient failures and guarantee system recovery in a finite time. Since the condition on the function parameter (being a strictly idempotent r-operator) permits a broad range of applications to be implemented, we believe the solution presented in our paper can be quite useful for a large class of distributed systems.

Keywords: r-operator, distributed computation, self-stabilization

Résumé :

Dans cet article, nous décrivons un algorithme réparti et paramétré qui est applicable à tout type de graphe orienté. La fonction paramètre de notre algorithme est instanciée de manière à produire plusieurs algorithmes résolvant à la fois des tâches fondamentales et des applications de haut niveau dans le cadre des systèmes répartis, telles que le calcul des plus courts chemins ou la construction d'un arbre en profondeur. Compte tenu des propriétés de résistance aux défaillances de notre algorithme, les protocoles obtenus sont auto-stabilisants sans surcoût additionel. Les protocoles auto-stabilisants peuvent résister à des défaillances transitoires en garantissant un retour à la normale du système en temps fini. Comme la condition sur la fonction passée en paramètre de notre algorithme (celle de pouvoir être réduite à un r-opérateur strictement idempotent) rend possible le codage d'un vaste champ d'applications réparties, nous pensons que la solution présentée dans cet article est d'une utilité certaine pour une large classe de systèmes répartis.

Mots clés: r-opérateur, calcul réparti, auto-stabilisation

1 Introduction

Robustness is one of the most important requirements of modern distributed systems. Various types of faults are likely to occur at various parts of the system. These systems go through the transient faults because they are exposed to constant change of their environment.

Self-stabilization. One of the most inclusive approaches to fault tolerance in distributed systems is *self-stabilization* [D74, S93c]. Introduced by Dijkstra in [D74], this technique guarantees that, regardless of the initial state, the system will eventually converge to the intended behavior or the set of *legitimate* states. Since most self-stabilizing fault-tolerant protocols are non-terminating, if the distributed system is subject to transient faults corrupting the internal node state but not its behavior, once faults cease, the protocols themselves guarantee to recover in a finite time to a safe state without the need of human intervention. This also means that the complicated task of initializing distributed systems is no longer needed, since self-stabilizing protocols regain correct behavior regardless of the initial state. Furthermore, note that in practice, the context in which we may apply self-stabilizing algorithms is fairly broad since the program code can be stored in a stable storage at each node so that it is always possible to reload the program after faults cease or after every fault detection.

Related Work Silent tasks [DGS96] are tasks where the communication between the processors is fixed from some point of the execution. In our model, registers are used for communication between processors. Then a system solving a silent task has the property that the contents of the communication registers is not changed after some point in the execution.

Historically, research in self-stabilization over general networks has mostly covered undirected networks where bidirectional communication is feasible [DIM93, AKY90], or directed graphs which are strongly connected [AHOPV94]. In general, this allows to build a virtual well known topology (tree or ring) on which the self-stabilizing algorithm may be run ([ABDT98]). Those restriction are reasonable when the task to be solved is dynamic and the system is asynchronous: *e.g.* for traversal algorithms, a token has to be able to pass through any node infinitely often.

Silent tasks have been solved in a self-stabilizing way on directed graphs that are not strongly connected in [BDDN95], but the underlying network was assumed having no cycles (DAG).

Our Contribution. In this paper, we concentrate on solving silent tasks in a self-stabilizing way on a truly general network, where no hypothesis are made about the strong connectivity or the presence of cycles. Our parameterized algorithm enables a set of silent tasks to be solved self-stabilizing provided that these tasks can be expressed through local calculus operations called r-operators. The r-operators are general enough to permit applications such as shortest path calculus and depth-first-search tree construction, to be solved on arbitrary graphs while remaining self-stabilizing.

Outline of the paper. The rest of the paper is organized as follows. In Section 2, we give some definitions pertinent to the protocols and proofs. The self-stabilizing parameterized protocol is presented in Section 3. The correctness reasoning for the parameterized protocol is given in Section 4. Application to fundamental problems in distributed computing area are presented in Section 5. We discuss the extension of our ideas and make some concluding remarks in Section 6.

2 Model

2.1 Distributed System.

Graph modelling A distributed system $\mathcal{S}$ is a collection of processors linked with communication media allowing them to exchange information. Such a system is modeled by a *directed graph* (also called *digraph*) $G(V, E)$, defined by a set of vertices V and a set E of edges (v_1, v_2), which are ordered[1] pairs of vertices of V $(v_1, v_2 \in V)$. Each vertex u in V represents a processor P_u of the system $\mathcal{S}$. Each edge (u, v) in E, represents a communication link from P_u to P_v in $\mathcal{S}$. We give now some graph definitions.

The *in-degree* of a vertex v of G, denoted by $\delta^-(v)$ is equal to the number of vertices u such that the edge (u, v) is in E. The incoming edges of each vertex v of G are numbered from 1 to $\delta^-(v)$. We denote by $\mathbf{ind}(u)$ the index of the incoming edge (u, v) of v.

A *directed path* from a vertex v_0 to a vertex v_k in a digraph $G(V, E)$ is a list of consecutive edges of E, $(v_0, v_1), (v_1, v_2), \ldots, (v_{k-1}, v_k)$. The length of this path is k. If each v_i is unique in the path, the path is *elementary.* A *cycle* is a directed path where $v_0 = v_k$. A digraph without any cycle is called a *directed acyclic graph (DAG).* The *sources* of the digraph G are the vertices s which have no ancestor. The *main sources* of the digraph G are the vertices s such that there exists a path from s to all other vertices.

The *distance* between two vertices u, v of a digraph G, denoted by $\mathrm{d}_G(u, v)$, is the minimum of the lengths of all directed paths from u to v. The *diameter* of a digraph G, denoted by $\mathrm{Diam}(G)$, is the maximum of the distances between all couples of vertices in G. The *strongly connected component* of a vertex v in a digraph $G(V, E)$ is the set of all vertices w of V such that there exists a directed path from v to w <u>and</u> a directed path from w to v. G is *strongly connected* if it has exactly one strongly connected component.

The *direct descendants* of a vertex v of a digraph $G(V, E)$ are all the vertices w of G such that the edge (v, w) is in E. Their set is denoted by $\Gamma_G(v)$. The *descendants* of v are all the vertices w such that there exists a path from v to w. Their set is denoted by $\widehat{\Gamma_G}(v)$. In the same way, the *direct ancestors* of a vertex v of G are all the vertices u of G such that the edge (u, v) is in E. Their set is denoted by $\Gamma_G^{-1}(v)$. The *ancestors* of v are all the vertices u such that there exists a path from u to v. Their set is denoted by $\widehat{\Gamma_G^{-1}}(v)$. Finally, $\Gamma_G^{-k}(v)$ denotes the set of all the ancestors u of G such that $\mathrm{d}_G(u, v) = k$, while $\Gamma_G^k(v)$ denotes the set of all the descendants w of v such that $\mathrm{d}_G(v, w) = k$.

Communications. A communication from processor P_u to processor P_v is only feasible if the vertex u is a direct ancestor of the vertex v in G (*i.e.* (u, v) is an edge of G). Such a communication is performed through a communication register. Processor P_u writes the datum to be sent to P_v into its dedicated register reg_u. Then P_v is able to read the datum in reg_u and uses it.

The processors maintain two types of variables: *local variables* and *field variables.* The field variables are part of the shared register which is used to communicate with the neighbors. The local variables defined in the program of a processor cannot be accessed by its neighbors and is used for local computations only. A processor may only write into its own shared register and can only read shared registers owned by its direct ancestor processors or itself. So, the field variables of a processor can be accessed by the processor itself and also by its direct descendants.

[1] $(v_1, v_2) \neq (v_2, v_1)$.

Processor P_v performing a call to the `read`(reg_α) function ($\alpha \in \Gamma_G^{-1}(v)$) atomically reads reg_α and obtains the list of field variables that are stored at this register. Processor P_u performing `write`(reg_u,*list*) atomically writes *list* to the corresponding fields of reg_u.

Processors. A processor is a deterministic sequential machine that runs a single process (hence, we indifferently speak about processor and process).

The *state* of a processor is defined by the values of its local variables. The state of a link (u, v) of E is defined by the values of the field variable reg_u.

A processor *action* (or step) consists of an internal computation followed by either a `read` or a `write` action.

Configuration and computations. A *configuration* of a distributed system $\mathcal{S}$ is an instance of the states of its processors and links. The set of configurations of $\mathcal{S}$ is denoted as $\mathcal{C}$. Processor actions change the global system configuration.

A *computation* e (also called an *execution*) is a sequence of configurations $c_1, c_2, \dots$ such that for $i = 1, 2, \dots$, the configuration c_{i+1} is reached from c_i by a single step of one processor. c_1 is called the *initial configuration* of c.

This scheduling policy is known as the *read-write* atomicity model (see [DIM93]). It should be noted that *read* and *write* actions are executed asynchronously,

All computations considered in this paper are assumed to be *fair* and *maximal*. *Fairness* of the sequence means that if some action in the protocol $\mathcal{P}$ is continuously enabled along the sequence, it is eventually chosen for execution. *Maximal* means that the sequence is either infinite, or it is finite and no action is enabled in the final configuration.

The set of computations in the distributed system $\mathcal{S}$ starting with a particular initial configuration $c_1 \in \mathcal{C}$ is denoted by $\mathcal{E}_{c_1}$. Every computation $e \in \mathcal{E}_{c_1}$ is of the form $c_1, c_2, \dots$. The set of computations in the system $\mathcal{S}$ whose initial configurations are all elements of $\mathcal{C}_1 \subset \mathcal{C}$ is denoted as $\mathcal{E}_{\mathcal{C}_1}$. Thus, $\mathcal{E} = \mathcal{E}_{\mathcal{C}}$ is the set of all possible computations.

2.2 Self-stabilization

A *specification* is a predicate on computations that are admissible for a distributed system. A system *matches its specification* if all its possible computations match the specification. If we consider only *static* problems (*i.e.*, problems whose solutions consist of computing some global calculus), the specification can be given in terms of a set of configurations. Every computation matching the specification would be a sequence of such configurations. The set of configurations that matches the specification of static problems is called the set of *legitimate* configurations (denoted as $\mathcal{L}$), while the remainder $\mathcal{C} \setminus \mathcal{L}$ denotes the set of *illegitimate* configurations.

We need to introduce the concept of an attractor to define self-stabilization. Intuitively, an attractor is a set of configurations of the system $\mathcal{S}$ that "attracts" another set of configurations of $\mathcal{S}$ for any computation in $\mathcal{E}$, the set of all possible computations. In addition, if the attractor is closed, then any subsequent computation of the algorithm remains in the same set of configurations.

Definition 2.1 (Closed Attractor) *Let $\mathcal{C}_1$ and $\mathcal{C}_2$ be subsets of $\mathcal{C}$. $\mathcal{C}_1$ is an attractor for $\mathcal{C}_2$ if and* *only if for any initial configuration* c_1 *in* $\mathcal{C}_2$*, for any execution* e *in* $\mathcal{E}_{c_1}$*,* $(e = c_1, c_2, \ldots)$*, there exists* $i \geq 1$ *such that for any* $j \geq i$*,* $c_j \in \mathcal{C}_1$*.*

Definition 2.2 (Self-stabilization) *A system* $\mathcal{S}$ *is called self-stabilizing if and only if there exists a non-empty subset* $\mathcal{L} \subset \mathcal{C}$ *of legitimate configurations such that* $\mathcal{L}$ *is a closed attractor for* $\mathcal{C}$*.*

In the usual (*i.e.* non-stabilizing) distributed systems, possible computations can be restricted by allowing the system to start only from some well-defined *initial* configurations. On the other hand, in stabilizing systems, problems can not be solved using this convenience, since all possible system configurations are admissible initial configurations.

2.3 r-operators

In this section we present the r-operators. Operators considered in this section are defined on a set A. We assume that for any operator, a neutral element e exists in A. This is not a restrictive hypothesis since such an element can always be added to A. Additional information and proofs can be found in [Duc98b, Duc98a].

We first define s-operators, on which r-operators are based on.

Definition 2.3 (s-operator) *A binary operator* $\star$ *is an s-operator on* A *if and only if* $\star$ *verifies the following properties for any* x*,* y *in* A*:*

$$\begin{array}{llll} (x \star y) \star z & = & x \star (y \star z) & \textit{associativity} \\ x \star y & = & y \star x & \textit{commutativity} \\ x \star x & = & x & \textit{idempotency} \end{array}$$

Defining an s-operator induces a partial ordering relation[2] $\preceq_\star$ defined by $x \preceq_\star y$ if and only if $x \star y = x$ (s holds for *sort* in the term s-operator). The neutral element e defined by $x \star e = x$ is the single element of A verifying $x \preceq_\star e$ for all x in A.

We now present binary r-operators.

Definition 2.4 (binary r-operator) *A binary operator* $\triangleleft$ *is an r-operator on* A *if there exists a bijective mapping* r*, called an* r-mapping*, such that* $\triangleleft$ *verifies the following properties for any* x*,* y*,* z *in* A*:*

$$\begin{array}{llll} (x \triangleleft y) \triangleleft r(z) & = & x \triangleleft (y \triangleleft z) & r\textit{-associativity} \\ r(x) \triangleleft y & = & r(y) \triangleleft x & r\textit{-commutativity} \\ r(x) \triangleleft x & = & r(x) & r\textit{-idempotency} \end{array}$$

Note that r holds for *reduce* in the term r-operator (it was defined to reduce some expressions in ([Duc98b]). For any r-operator, there exists an s-operator $\star$ such that for any x and y in A, $x \star y = x \triangleleft r^{-1}(y)$. The neutral element e of $\star$ is the right neutral element of $\triangleleft$. Moreover the r-mapping of any r-operator is an isomorphism of $(A, \triangleleft)$ and $(A, \star)$. There are as many r-operators on A as there exists some couples of s-operators $\star$ and isomorphisms r of $(A, \star)$.

[2]This result is known as the Infimum Theorem in [Tel94].

When the r-operator $\triangleleft$ verifies $x \triangleleft x = x$, it is said idempotent. To obtain an idempotent r-operator $\triangleleft$ from an s-operator $\star$ and an isomorphism r of $(A, \star)$, it suffices that $\star$ and r verify, for all x in A, $x \preceq_\star r(x)$.

The n-ary r-operators are built following construction of binary r-operators from s-operators.

Definition 2.5 (n-ary r-operator) *A mapping $\bigtriangledown$ from A^n into A is an n-*ary r-operator *if there exists an s-operator $\star$ on A and $n-1$ isomorphisms $r_1, \ldots, r_{n-1}$ of $(A, \star)$ such that for any x in A, $\bigtriangledown(x_0, \ldots, x_{n-1}) = x_0 \star r_1(x_1) \star \cdots \star r_{n-1}(x_{n-1})$.*

In other words, an n-ary r-operator is built with $n-1$ r-operators $\triangleleft_1, \ldots, \triangleleft_{n-1}$ based on the same s-operator and the r-mappings $r_1, \ldots, r_{n-1}$. If all of these binary r-operators are idempotent ($\forall i \in \{1, \ldots, n-1\}, x \triangleleft_i x = x \star r_i(x) = x$), the resulting n-ary r-operator is idempotent.

We now define strict idempotency for n-ary r-operators.

Definition 2.6 (strict idempotency) *An n-ary r-operator $\bigtriangledown$ based on the n r-mappings $r_1, \ldots, r_n$ and the s-operator $\star$ is* strictly idempotent *if for any x in A, $x \neq e$ (e is the neutral element), for any i in $\{1, \ldots, n\}$, $x \prec_\star r_i(x)$, where $\prec_\star$ is a strict partial order relation defined from the partial order relation $\preceq_\star$ by $x \prec_\star y \equiv (x \star y = x$ and $x \neq y)$.*

Consider a distributed system $\mathcal{S}$ where each processor P_v uses a $(\delta^-(v)+1)$-ary r-operator to compute its current result. Then, for each edge (u, v) of the digraph $G(V, E)$ that models the system, there exists an r-mapping that we denote by r_v^u. Similarly, a mapping can be defined for any path in G.

Definition 2.7 (r-path mapping) *Let $P = (v_0, v_1) \ldots (v_{k-1}, v_k)$ be a directed path of the digraph $G(V, E)$ that models $\mathcal{S}$. The mapping r_P from A into A defined by*[3] *$r_P = r_{v_k}^{v_{k-1}} \circ \cdots \circ r_{v_1}^{v_0}$ is called the* r-path-mapping *of P.*

Definition 2.8 *An* r-augmented variable *is the image of a variable x through an r-mapping r ($r(x)$) or by an r-path-mapping ($r_P(x)$).*

3 Parametric Algorithm

In this section, we describe a simple parametric algorithm that simply gets input data from its incoming neighbors, computes a local function $\mathcal{F}_v$ and finally makes the result of this function available to its outgoing neighbors.

3.1 Programming Notation

The program for each protocol consists of a sequence of actions: $<$ *action* $>$ $\cdots$ $<$ *action* $>$. Each *action* has the form: $<$ *guard* $>\longrightarrow<$ *statement* $>$. A *guard* is a boolean expression over the local variables of a processor and the field variables of its immediate ancestors. A *statement* is allowed to

[3] French composition of the applications: $f \circ g(x) = f(g(x))$.

update the variables of the processor only. Any action whose guard is *true* is said to be *enabled.* A processor with one or more enabled actions is said to be *privileged* and may make a *move* executing the statement corresponding to the chosen enabled guard.

When no rule is enabled on any node in the distributed system $\mathcal{S}$, the corresponding configuration is called *terminal.* If for any initial configuration, and any subsequent execution, a terminal configuration is reached, the system $\mathcal{S}$ is *silent.*

3.2 Formal Description

Each processor P_v has two local constants stored in Read Only Memory: the *initial datum*, $M_v[0]$, and the set of its direct ancestors $\Gamma_G^{-1}(v)$.

To replicate values from its direct ancestors and ensure `read/write` atomicity, P_v also has $\delta^-(v)$ local variables $M_v[1]$, ..., $M_v[\delta^-(v)]$, the *incoming variables.* To store the result of the local computation, a single field variable is used at P_v: $M_v[\delta^-(v)+1]$, the *outgoing variable.*

In addition to the above, the protocol maintains on each processor P_v a function $\mathcal{F}_v$ which is defined as:

$$\begin{array}{rlcl} \mathcal{F}_v: & \mathtt{t}_{\mathtt{M_v}} & \rightarrow & \mathtt{t}_{\mathtt{M_v[.]}} \\ & M_v[1..\delta^-(v)] & \mapsto & \mathcal{F}(M_v[1..\delta^-(v)]) \end{array}$$

The function $\mathcal{F}_v$ implements the calculation of a strictly idempotent $(\delta^-(v)+1)$-ary r-operator. It is a well-defined local function. The input to $\mathcal{F}_v$ are the initial datum $M_v[0]$ of v and the values of the local variables copied from the direct ancestors into $M_v[1..\delta^-(v)]$. $\mathcal{F}$ outputs the current state of the processor computing the function. This result is stored in $M_v[\delta^-(v)+1]$.

The dynamics of each processor P_v is as follows:

(1) reads data in the registers of RR_v and copies them into its incoming variables $M_v[1]\ldots M_v[\delta^-(v)]$,

(2) performs a local computation using function $\mathcal{F}_v$, its initial datum $M_v[0]$ and the received data $M_v[1]\ldots M_v[\delta^-(v)]$, and stores the result into its outgoing variable $M_v[\delta^-(v)+1]$:

$$M_v[\delta^-(v)+1] \quad \leftarrow \quad \mathcal{F}_v\left(M_v[0], \ldots, M_v[\delta^-(v)]\right)$$

More formally, the parameterized algorithm consists of two guarded rules instantiated by $\mathcal{F}_v$ and is shown in Algorithm 3.1.

Implementation notes. Having each node v disposing of the $\Gamma_G^{-1}(v)$ is only convenient when writing the algorithm. In an actual implementation, this set can be efficiently replaced with the list of v in-port hardware addresses.

The first rule, called *Copying rule* and denoted by $R_1|\mathcal{F}_v$, copies the direct ancestor field variables into the processor's local variables, so that they can be used at a later time.

It uses a local variable α to avoid reading the field variable $M_u[\delta^-(v)+1]$ twice, and thus, implements the `read/write` atomicity. In $R_1|\mathcal{F}_v$, an expression of the form $<$ *left* $>:=<$ *right* $>$ is used. The first operand *left* must be a variable while the second operand *right* may be a constant or a variable. The operator $:=$ assigns the value of *right* to *left* and always returns *true*.

The second rule, called *Computing rule* and denoted by $R_2|\mathcal{F}_v$, computes the $\mathcal{F}_v$ function with the previously copied variables as input and stores the result into $M_v[\delta^-(v)+1]$.

Algorithm 3.1 The $\mathcal{PA}$ algorithm

	(Copying)
$R_1\vert\mathcal{F}_v$:	$\left(P_u \in \Gamma_G^{-1}(v) : (\alpha := \texttt{read}\,(M_u[\delta^-(u)+1])) \wedge M_v[\mathbf{ind}(u)] \neq \alpha\right) \longrightarrow$
	$M_v[\mathbf{ind}(u)] := \alpha;$
	(Computing)
$R_2\vert\mathcal{F}_v$:	$(\texttt{read}\,(M_v[\delta^-(v)+1]) \neq \mathcal{F}_v\,(M_v[0..\delta^-(v)])) \longrightarrow$
	$\texttt{write}\,(M_v[\delta^-(v)+1], \mathcal{F}_v\,(M_v[0..\delta^-(v)]))\,;$

4 Proof of Correctness

In this section we assume our algorithm is instantiated with one strictly idempotent n-ary r-operator as $\mathcal{F}_v$ for each processor v. We do not require that all those r-operators are the same for each node. But we assume that they are all based on the same s-operator $\star$, such that the ordering relation defined by $x \preceq_\star y \equiv x \star y = x$ is a total order relation. In this case, for any set of data $\{x_0, \ldots, x_k\}$, we have $\star\{x_0, \ldots, x_k\} = x_0 \star \cdots \star x_k \in \{x_0, \ldots, x_k\}$, which is not always true when the ordering relation $\preceq_\star$ is not total.

We first provide specification of legitimate configuration when dealing with algorithms based on n-ary r-operators. Then we prove that **(i)** if $\mathcal{F}_v$ is instantiated with a n-ary r-operator that is strictly idempotent, the resulting algorithm $\mathcal{A}|\mathcal{F}_v$ is self-stabilizing for the intended specification, while **(ii)** if $\mathcal{F}_v$ is instantiated with a n-ary r-operator that is not strictly idempotent, the resulting algorithm $\mathcal{A}|\mathcal{F}_v$ is not self-stabilizing. Finally, we investigate atomicity related issues proving the `read/write` atomicity compliance of our algorithm.

Notation 4.1 *We call r_v^i the i^{th} r-mapping in the node v, for i in $1, \ldots, \delta^-(v)$. To shorten our notations, we admit that, for any node v, there is also an r-mapping r_v^0 corresponding to the initial datum, and we state that $r_v^0 \equiv$* **Id**.

4.1 Legitimate configurations

We define the legitimate property for nodes through the legitimate property for node variables. We denote by $\mathcal{L}(x)$ the legitimate value of variable x.

Definition 4.1 (Legitimate variable)

$$\mathcal{L}\left(M_v[\delta^-(v)+1]\right) = \star\left\{\begin{array}{l} r_P(M_u[0]), \\ u \in \widehat{\Gamma_G^{-1}}(v), \\ P \text{ an elementary path from } u \text{ to } v \end{array}\right\}$$

For any node v, the outgoing variable $M_v[\delta^-(v)+1]$ is legitimate *if $M_v[\delta^-(v)+1] = \mathcal{L}\left(M_v[\delta^-(v)+1]\right)$, and an incoming variable $M_v[\mathbf{ind}(u)]$ is legitimate if $M_v[\mathbf{ind}(u)] = \mathcal{L}\left(M_u[\delta^-(u)+1]\right)$.*

For any node v, the initial datum $M_v[0]$ is legitimate in any configuration, since this datum is stored in Read Only Memory.

Definition 4.2 (Legitimate node) *A node v with a legitimate outgoing variable $M_v[\delta^-(v)+1]$ is* legitimate*.*

Given these definitions, a legitimate configuration is simply a configuration where all nodes are legitimate.

Definition 4.3 (Legitimate Configuration) *The set $\mathcal{L}$ of legitimate configurations for $\mathcal{S}$ is defined as: $\forall l \in \mathcal{L}$, $\forall v$ in l, v is legitimate.*

4.2 Strict idempotency of n-ary r-operators yields self-stabilization

The proof of correctness follows the following scheme:

(1) We prove that any computation has a finite number of steps,

(2) We prove that in any terminal configuration, all nodes of the graph G are legitimate.

Lemma 4.1 (Silent) *Algorithm $\mathcal{PA}$ is silent.*

Proof. We wish to prove that, starting from any initial configuration c_1, any execution is finite in the absence of new transient faults, *i.e.* after finite time, no rule (either $R_1|\mathcal{F}_v$ or $R_2|\mathcal{F}_v$) is enabled.

In any initial configuration c_1, there exists a finite number of data in the network, either stored in the Read Only Memory, in some incoming variables $M_v[i]$, $1 \leq i \leq \delta^-(v)$, for any v in the network, or in some outgoing variable $M_v[\delta^-(v)+1]$.

Without loss of generality, one can consider that all the results built by the processors of the distributed system are stored into a formal expression where the r-mapping are not computed. For example, if the processor P_v has x_0 in $M_v[0]$ and $x_1, \ldots, x_{\delta^-(v)}$ in $M_v[1..\delta^-(v)]$, then the information stored into $M_v[\delta^-(v)+1]$ after applying the computing rule $R_2|\mathcal{F}_v$ is $r_v^i(M_v[i])$ where

$$r_v^i\left(M_v[i]\right) = M_v[0] \star r_v^1\left(M_v[1]\right) \star \cdots \star r_v^{\delta^-(v)}\left(M_v[\delta^-(v)]\right)$$

instead of $y \in A$, with $y = r_v^i(M_v[i])$. This is possible because we supposed that the s-operator $\star$ induces a total order relation. Hence, there is no creation of new data in the network, but only creation of new expressions, composed of r-mappings — and composition of r-mappings (*i.e.* r-path-mappings) — and data which were in the network at the beginning. One can remark that there is always one datum per expression, and any expression is an r-augmented datum which was already in the system at configuration c_1.

Any copying rule $R_1|\mathcal{F}_v$ moves the r-augmented datum of the outgoing variable $M_u[\delta^-(u)+1]$ of each direct ancestor u of v into the incoming variables $M_v[i]$ of v. This erases the previously stored expression in these incoming variables. Any computing rule $R_2|\mathcal{F}_v$ moves the r-augmented datum stored into $M_v[i]$ for a single i in $\{0, \ldots, \delta^-(v)\}$, in a new expression $r_v^i(M_v[i])$ which is stored into $M_v[\delta^-(v)+1]$. This erases the previously stored expression in $M_v[\delta^-(v)+1]$. Rule computation induces data progress in the network, and *vice versa*. Thus, if for any subsequent execution starting from configuration c_t, there is no progress of data, it means that no rule is enabled starting from c_t, that in turn c_t is a terminal configuration.

Suppose that the network is a DAG. In this case, there is no cycle, and the data movement may not occur forever. Hence, starting from any configuration, and for each subsequent computation, there exists a terminal configuration where no rule is enabled.

Now suppose that there are some cycles in the network. Some data may move forever, reaching at least one node in the network infinitely often. Let x be a datum present at node v in configuration c_i, inside the r-path-mapping r_P. Suppose that there exists a computation starting from c_i and such that x reaches again v at configuration c_j. In c_j, x has been r-augmented by an r-path-mapping corresponding to the list of nodes that x has visited between configuration c_i and configuration c_j. We denote this particular r-path-mapping using the configuration names: $r_{P_{c_i \rightarrow c_j}}$. By hypotheses, any r-mapping r verifies that, for any $y \in A$, $y \prec_\star r(y)$. Then any r-path-mapping r_P verifies also $y \prec_\star r_P(y)$, for any $y \in A$. Hence, in c_j, x is r-augmented by $r_{P_{c_i \rightarrow c_j}}$ and we have $r_P(x) \prec_\star r_{P_{c_i \rightarrow c_j}} \circ r_P(x)$. This means that the expression containing x is strictly increasing at each arrival of x at v. Suppose now that there exists a computation such that x reaches v infinitely often. In this case, there exists a configuration c_k such that $M_v[0] \prec_\star r_{P_{c_i \rightarrow c_k}} \circ r_P(x)$. In turn, this means that the expression containing x will no more be "chosen" by the computation of $\mathcal{F}_v$, and then x may not progress in the network.

Finally, no infinite computation (in the sense that some rules remain enabled) may occur in the distributed system we consider, and the algorithm is silent. □

We now want to prove that in any such terminal configuration, all nodes in the network are legitimate. Since we supposed that the s-operator $\star$ defines a total order, one can sort, for any given configuration, the legitimate incoming variables of all the nodes of the system.

Definition 4.4 (Λ) *Let c be a configuration, Λ denotes the set of all legitimate incoming variables in c. Using the total ordering relation $\prec_\star$ induced by $\star$, $\Lambda = \{\lambda_0, \lambda_1, \ldots, \lambda_k\}$ with $\lambda_i \prec_\star \lambda_j$ for $i < j$.*

Moreover, for any given configuration, one can sort out the nodes of the system into several distinct sets:

Definition 4.5 (Ψ_n) *Let c be a configuration, Ψ_n is the set of the nodes of the system verifying the following criteria:*

$$\Psi_n = \left\{ v \in V \ \ s.t. \begin{array}{c} \exists i, 0 \leq i \leq \delta_v^-, \\ M_v[i] \text{ is legitimate}, \\ r_v^i(M_v[i]) = \lambda_n, \\ \nexists n' < n, v \in \Psi_{n'} \end{array} \right\}$$

Lemma 4.2 *In any configuration, there is no legitimate outgoing variable smaller than λ_0.*

Proof. Any legitimate variable is equal to an expression built with initial data increased by strictly idempotent r-path-mappings. Suppose that

$$\mathcal{L}\left(M_v[\delta^-(v)+1]\right) \prec_\star \lambda_0$$

Then we have

$$\star \left\{ r_P\left(M_u[0]\right), u \in \widehat{\Gamma_G^{-1}}(v), P \text{ elementary path from } u \text{ to } v \right\} \prec_\star \lambda_0$$

and then there exists $u \in \widehat{\Gamma_G^{-1}}(v)$ and an r-path-mapping r_P such that

$$M_u[0] \prec_\star r_P\left(M_u[0]\right) \prec_\star \lambda_0$$

which is impossible, by definition of λ_0. □

Lemma 4.3 *All the nodes of Ψ_0 verify the following relation:*

$$\mathcal{L}(M_v[\delta^-(v)+1]) = \mathcal{L}(M_v[0]) = M_v[0] = \lambda_0$$

Proof. We have:

$$\mathcal{L}(M_v[\delta^-(v)+1]) = \star\left\{r_v^i\left(\mathcal{L}\left(M_v[i]\right)\right), 0 \leq i \leq \delta^-(v)\right\}$$

and since v is in Ψ_0,

$$\mathcal{L}(M_v[\delta^-(v)+1]) \prec_\star \lambda_0$$

But, following Lemma 4.2, no legitimate variable is smaller than λ_0, which means that

$$\mathcal{L}(M_v[\delta^-(v)+1]) = \lambda_0$$

Now suppose that there exists an integer q, $1 \leq q \leq \delta^-(v)$ such that

$$\mathcal{L}(M_v[\delta^-(v)+1]) = r_v^q\left(\mathcal{L}(M_v[q])\right)$$

Since the r-mappings are strictly idempotent, we have:

$$\begin{array}{rcl}\mathcal{L}(M_v[q]) & \prec_\star & r_v^q\left(\mathcal{L}(M_v[q])\right) \\ & \prec_\star & \lambda_0\end{array}$$

which is impossible by definition of λ_0. □

For a given configuration, we consider the following Assertion P_n^k.

$$\begin{array}{ll}P_n^k: & \exists v \in G, \exists n_v \geq n \text{ s.t. } v \in \Psi_{n_v}, \\ & \exists u_1 \in \Gamma_G^{-1}(v), \ldots, u_k \in \Gamma_G^{-k}(v), \\ & \forall i, 1 \leq i \leq k, \exists n_i \geq n, u_i \in \Psi_{n_i}, \\ & (u_k, u_{k-1}) \in E, \ldots, (u_2, u_1) \in E, (u_1, v) \in E \\ & M_{u_k}[\delta^-(u_k)+1] \prec_\star \ldots \prec_\star M_{u_1}[\delta^-(u_1)+1] \prec_\star \lambda_n\end{array}$$

P_n^k states that there exists a path of length k of illegitimate distinct vertices. Indeed, $v \in \Psi_n$ means that the smallest legitimate r-augmented incoming variable of v is λ_n. If the outgoing variable of v is smaller than λ_n, then this value has been given by another r-augmented incoming variable, smaller than λ_n. Hence, this incoming variable cannot be legitimate, and therefore v cannot be legitimate too.

We now wish to prove that in a terminal configuration c_t, all nodes of $\mathcal{P}$ are legitimate. The proof is by contradiction and multiple recurrence on P_n^k. We wish to prove that: if there exists a terminal configuration c_t such that there exists a node $v \in \mathcal{P}$ which is non-legitimate, then P_n^k holds.

1. First, we prove that if there exists a terminal configuration c_t such that there exists a node $v \in \Psi_0$ which is non-legitimate, then P_0^1 holds (Lemma 4.4).
2. Then, we prove by recurrence on k that if there exists a terminal configuration c_t such that if there exists a node $v \in \Psi_0$ which is non-legitimate, then P_0^k holds for $k \in \mathbb{N} \setminus \{0\}$ (Lemma 4.5).
3. Since the network is finite, P_0^k may not hold for $k \geq N+1$, where N is the size of the network. Thus, all nodes in Ψ_0 are legitimate (Lemma 4.6).
4. Next, we prove that if there exists a terminal configuration c_t such that all nodes $u \in \Psi_0 \cup \ldots \cup \Psi_{n-1}$ are legitimate and there exists a node v in Ψ_n which is non-legitimate, then P_n^1 holds (Lemma 4.7).
5. Then, we prove by recurrence on k that if there exists a terminal configuration c_t such that all nodes $u \in \Psi_0 \cup \ldots \cup \Psi_{n-1}$ are legitimate and there exists a node v in Ψ_n which is non-legitimate, then P_n^k holds for $k \in \mathbb{N} \setminus \{0\}$ and $n \in \mathbb{N}$ (Lemma 4.8).
6. Since the network is finite, P_n^k may not hold for $k \geq N+1$, where N is the size of the network. Thus, all nodes in Ψ_n are legitimate for all $n \in \mathbb{N}$, and then all nodes in $\mathcal{P}$ are legitimate (Lemma 4.9).

Lemma 4.4 *In any terminal configuration where exists $v \in \Psi_0$ such that v is non-legitimate, P_0^1 holds.*

Proof. We explicitly expand P_0^1 as follows:

$$\begin{array}{ll} P_0^1: & \exists v \in G, \exists n_v \geq 0 \text{s.t.} v \in \Psi_{n_v}, \\ & \exists u_1 \in \Gamma_G^{-1}(v), \\ & \exists n_1 \geq 0, u_1 \in \Psi_{n_1}, \\ & (u_1, v) \in E, \\ & M_{u_1}[\delta^-(u_1)+1] \prec_\star \lambda_0 \end{array}$$

Consider a node $v \in \Psi_0$. Following Lemma 4.3, we have

$$\mathcal{L}(M_v[\delta^-(v)+1]) = M_v[0] = \lambda_0$$

Suppose now that there exists a terminal configuration c_t such that the node v is non-legitimate:

$$M_v[\delta^-(v)+1] \neq \mathcal{L}\left(M_v[\delta^-(v)+1]\right)$$

Since the configuration c_t is terminal, no rule is enabled, which means that:

$$\begin{array}{rcl} M_v[\delta^-(v)+1] & = & \star\left\{r_v^i\left(M_v[i]\right), 0 \leq i \leq \delta^-(v)\right\} \\ & \prec_\star & M_v[0] \\ & \prec_\star & \lambda_0 \end{array}$$

Hence the node v has at least one incoming r-augmented variable smaller than λ_0:

$$\exists u_1 \in \Gamma_G^{-1}(v), r_v^{u_1}\left(M_v[\mathbf{ind}(\mathrm{u}_1)]\right) \prec_\star \lambda_0$$

Since all the r-mappings are strictly idempotent, we have:

$$\exists u_1 \in \Gamma_G^{-1}(v), M_v[\mathbf{ind}(\mathrm{u}_1)] \prec_\star r_v^{u_1}\left(M_v[\mathbf{ind}(\mathrm{u}_1)]\right) \prec_\star \lambda_0$$

Now, since the considered configuration is terminal, all the incoming variables of the node v have been well copied from the direct ancestors of v. Formally, we have:

$$\exists u_1 \in \Gamma_G^{-1}(v), M_{u_1}[\delta^-(u_1)+1] = M_v[\mathbf{ind}(\mathrm{u}_1)] \prec_\star r_v^{u_1}(\mathbf{ind}(\mathrm{u}_1)) \prec_\star \lambda_0$$

Note that (u_1, v) is a path of length one. Moreover, the node u_1 is non-legitimate (Lemma 4.3). We proved that, if there exists a terminal configuration c_t in which node v is non-legitimate, then Assertion P_0^1 holds, which means that there exists a path of length one of non-legitimate distinct nodes. □

Lemma 4.5 *In any terminal configuration where exists $v \in \Psi_0$ such that v is non-legitimate, P_0^k holds.*

Proof. We will prove by recurrence that, if there exists a terminal configuration c_t where node v is non-legitimate, then, for any $k \in \mathbb{N}$, P_0^k holds.

Base case. The base case is proved by Lemma 4.4.

Induction step. Suppose that there exists a terminal configuration c_t and an integer $k > 1$ such that P_0^{k-1} holds:

$$\begin{aligned} P_0^{k-1}: \quad & \exists v \in G, \exists n_v \geq 0 \text{ s.t. } v \in \Psi_{n_v}, \\ & \exists u_1 \in \Gamma_G^{-1}(v), \ldots, u_{k-1} \in \Gamma_G^{-(k-1)}(v), \\ & \forall i, 1 \leq i < k, \exists n_i \geq 0, u_i \in \Psi_{n_i}, \\ & (u_{k-1}, u_{k-2}) \in E, \ldots, (u_2, u_1) \in E, (u_1, v) \in E \\ & M_{u_{k-1}}[\delta^-(u_{k-1})+1] \prec_\star \ldots \prec_\star M_{u_1}[\delta^-(u_1)+1] \prec_\star \lambda_0 \end{aligned}$$

Since c_t is terminal, the outgoing variables $M_{u_{k-1}}[\delta^-(u_{k-1})+1]$ of any node u_{k-1} of $\Gamma_G^{-(k-1)}(v)$ is equal to its initial datum $M_{u_{k-1}}[0]$ or to one of its incoming variable, augmented by its r-mapping, $r_v^i\left(M_{u_{k-1}}[i]\right)$, $1 \leq i \leq \delta^-(u_{k-1})$. However, by definition, there is no initial datum smaller than λ_0. Then, we have:

$$\begin{aligned} \exists u_k \in \Gamma_G^{-k}(v), \quad & \quad (u_k, u_{k-1}) \in E, \\ r_{u_{k-1}}^{u_k}\left(M_{u_{k-1}}[\mathbf{ind}(\mathrm{u_k})]\right) &= M_{u_{k-1}}[\delta^-(u_{k-1})+1] \\ &\prec_\star \ldots \\ &\prec_\star M_{u_1}[\delta^-(u_1)+1] \\ &\prec_\star \lambda_0 \end{aligned}$$

Since c_t is terminal, each incoming variable of the nodes in $\Gamma_G^{-(k-1)}(v)$ have been well copied from the outgoing variables of the nodes in $\Gamma_G^{-k}(v)$. Then, we have:

$$\begin{aligned} \exists u_k \in \Gamma_G^{-k}(v), \quad & \quad (u_k, u_{k-1}) \in E, \\ M_{u_k}[\delta^-(u_k)+1] &= M_{u_{k-1}}[\mathbf{ind}(\mathrm{u_k})] \\ r_{u_{k-1}}^{u_k}\left(M_{u_{k-1}}[\mathbf{ind}(\mathrm{u_k})]\right) &= M_{u_{k-1}}[\delta^-(u_{k-1})+1] \\ &\prec_\star \ldots \\ &\prec_\star M_{u_1}[\delta^-(u_1)+1] \\ &\prec_\star \lambda_0 \end{aligned}$$

The r-mappings are strictly idempotent, so :

$$\begin{array}{rcl} \exists u_k \in \Gamma_G^{-k}(v), & & (u_k, u_{k-1}) \in E, \\ M_{u_k}[\delta^-(u_k)+1] & \prec_\star & M_{u_{k-1}}[\delta^-(u_{k-1})+1] \\ & \prec_\star & \ldots \\ & \prec_\star & M_{u_1}[\delta^-(u_1)+1] \\ & \prec_\star & \lambda_0 \end{array}$$

Which in turn is a rewriting of assertion P_0^k.

Conclusion. Hence we proved by recurrence that assertion P_0^k holds for any $k \in \mathbb{N} \setminus \{0\}$. □

Lemma 4.6 *In any terminal configuration, no node in Ψ_0 is non-legitimate.*

Proof. From Lemma 4.5, in any terminal configuration, P_0^k holds for any $k \in \mathbb{N} \setminus \{0\}$. Since P_0^k cannot hold for k greater than the number of nodes in the system, no vertex of Ψ_0 can remain always illegitimate. □

Lemma 4.7 *If there exists a terminal configuration c_t such that all nodes $u \in \Psi_0 \cup \ldots \cup \Psi_{n-1}$ are legitimate and there exists a node v in Ψ_n which is non-legitimate, then P_n^1 holds for $n \in \mathbb{N}$.*

Proof. We explicitly expand P_n^1 as follows:

$$\begin{array}{rl} P_n^1 : & \exists v \in G, \exists n_v \geq n \text{s.t.} v \in \Psi_{n_v}, \\ & \exists u_1 \in \Gamma_G^{-1}(v), \\ & \exists n_1 \geq n, u_1 \in \Psi_{n_1}, \\ & (u_1, v) \in E, \\ & M_{u_1}[\delta^-(u_1)+1] \prec_\star \lambda_n \end{array}$$

Then consider $v \in \Psi_n$. There exists an integer q, $1 \leq q \leq \delta^-(v)$, such that $M_v[q]$ is legitimate and $r_v^q(M_v[q]) = \lambda_n$. Suppose now that there exists a terminal configuration c_t such that the node v is non-legitimate:

$$M_v[\delta^-(v)+1] \neq \mathcal{L}\left(M_v[\delta^-(v)+1]\right)$$

Since the configuration c_t is terminal, no rule is enabled, which means that:

$$\begin{array}{rcl} M_v[\delta^-(v)+1] & = & \star\left\{r_v^i\left(M_v[i]\right), 0 \leq i \leq \delta^-(v)\right\} \\ & \prec_\star & \lambda_n \end{array}$$

Suppose that $M_v[\delta^-(v)+1] = M_v[0]$. Then we have $M_v[0] \prec_\star \lambda_n$. But since $M_v[0] = \mathcal{L}(M_v[0])$, v would not be in Ψ_n. Hence we have

$$\begin{array}{rcl} \exists u_1 \in \Gamma_G^{-1}(v), r_v^{u_1}\left(M_v[\mathbf{ind}(\mathrm{u}_1)]\right) & = & M_v[\delta^-(v)+1] \\ & \prec_\star & \lambda_n \end{array}$$

Since all the r-mappings are strictly idempotent, we have:

$$\exists u_1 \in \Gamma_G^{-1}(v), M_v[\mathbf{ind}(\mathrm{u}_1)] \prec_\star r_v^{u_1}\left(M_v[\mathbf{ind}(\mathrm{u}_1)]\right) \prec_\star \lambda_n$$

Now, since the considered configuration is terminal, all the incoming variables of the node v have been well copied from the direct ancestors of v. Formally, We have:

$$\exists u_1 \in \Gamma_G^{-1}(v), M_{u_1}[\delta^-(u_1)+1] = M_v[\mathbf{ind}(\mathrm{u}_1)] \prec_\star r_v^{u_1}(\mathbf{ind}(\mathrm{u}_1)) \prec_\star \lambda_n$$

Since $M_v[\mathbf{ind}(\mathrm{u}_1)] \prec_\star \lambda_n$ and $v \in \Psi_n$, $M_v[Indu_1]$ cannot be legitimate. Thus $M_{u_1}[\delta^-(v)+1]$ is not legitimate and the node u_1 is not legitimate, which means that $u_1 \in \Psi_{n_1}$ with $n_1 \geq n$.

Note that (u_1, v) is a path of length one. We proved that, if there exists a terminal configuration c_t in which node v is non-legitimate, then Assertion P_n^1 holds, which means that there exists a path of length one of non-legitimate distinct nodes. $\square$

Lemma 4.8 *In any terminal configuration where exists $v \in \Psi_n$ such that v is non-legitimate, P_n^k holds.*

Proof. We will prove by recurrence that, if there exists a terminal configuration c_t where node $v \in \Psi_n$ is non-legitimate, then, for any $k \in \mathbb{N}$, P_n^k holds.

Base case. The base case is proved by Lemma 4.7.

Induction step. Suppose that there exists a terminal configuration c_t such that P_n^{k-1} holds:

$$\begin{array}{rl} P_n^{k-1}: & \exists v \in G, \exists n_v \geq n \text{s.t.} v \in \Psi_{n_v}, \\ & \exists u_1 \in \Gamma_G^{-1}(v), \ldots, u_{k-1} \in \Gamma_G^{-(k-1)}(v), \\ & \forall i, 1 \leq i < k, \exists n_i \geq n, u_i \in \Psi_{n_i}, \\ & (u_{k-1}, u_{k-2}) \in E, \ldots, (u_2, u_1) \in E, (u_1, v) \in E \\ & M_{u_{k-1}}[\delta^-(u_{k-1})+1] \prec_\star \ldots \prec_\star M_{u_1}[\delta^-(u_1)+1] \prec_\star \lambda_n \end{array}$$

Since c_t is terminal, the outgoing variables $M_{u_{k-1}}[\delta^-(u_{k-1})+1]$ of any node u_{k-1} of $\Gamma_G^{-(k-1)}(v)$ is equal to its initial datum $M_{u_{k-1}}[0]$ or to one of its incoming variable, augmented by its r-mapping, $r_v^i\left(M_{u_{k-1}}[i]\right)$, $\leq i \leq \delta^-(u_{k-1})$. No r-augmented legitimate variable is smaller than λ_n on u_{k-1}, else $u_{k-1} \in \Psi_{n'}$ with $n' < n$. In particular, $M_{u_{k-1}}[0] \not\prec_\star \lambda_n$. Thus we have:

$$\begin{array}{rcl} \exists u_k \in \Gamma_G^{-k}(v), & & (u_k, u_{k-1}) \in E, \\ r_{u_{k-1}}^{u_k}\left(M_{u_{k-1}}[\mathbf{ind}(\mathrm{u_k})]\right) & = & M_{u_{k-1}}[\delta^-(u_{k-1})+1] \\ & \prec_\star & \ldots \\ & \prec_\star & M_{u_1}[\delta^-(u_1)+1] \\ & \prec_\star & \lambda_n \end{array}$$

Since c_t is terminal, each incoming variable of the nodes in $\Gamma_G^{-(k-1)}(v)$ have been well copied from the outgoing variables of the nodes in $\Gamma_G^{-k}(v)$. Then, we have:

$$\begin{array}{rcl} \exists u_k \in \Gamma_G^{-k}(v), & & (u_k, u_{k-1}) \in E, \\ M_{u_k}[\delta^-(u_k)+1] & = & M_{u_{k-1}}[\mathbf{ind}(\mathrm{u_k})] \\ r_{u_{k-1}}^{u_k}\left(M_{u_{k-1}}[\mathbf{ind}(\mathrm{u_k})]\right) & = & M_{u_{k-1}}[\delta^-(u_{k-1})+1] \\ & \prec_\star & \ldots \\ & \prec_\star & M_{u_1}[\delta^-(u_1)+1] \\ & \prec_\star & \lambda_n \end{array}$$

The r-mappings are strictly idempotent, so :

$$\begin{array}{rcl} \exists u_k \in \Gamma_G^{-k}(v), & & (u_k, u_{k-1}) \in E, \\ M_{u_k}[\delta^-(u_k)+1] & \prec_\star & M_{u_{k-1}}[\delta^-(u_{k-1})+1] \\ & \prec_\star & \ldots \\ & \prec_\star & M_{u_1}[\delta^-(u_1)+1] \\ & \prec_\star & \lambda_n \end{array}$$

Finally, $M_{u_{k-1}}[\mathbf{ind}(\mathrm{u_k})]$ is not legitimate, else u_{k-1} would not be in Ψ_n ($M_{u_{k-1}}[\mathbf{ind}(\mathrm{u_k})] \prec_\star \lambda_n$). Thus $M_{u_k}[\delta^-(v)+1]$ is not legitimate and then the node u_k is not legitimate, which means that $u_k \in \Psi_{n'}$ with $n' \geq n$. That gives P_n^k.

Conclusion. Hence we proved by recurrence that Assertion P_n^k holds for any $k \in \mathbb{N} \setminus \{0\}$. □

Lemma 4.9 (correctness) *For any terminal configuration c_t, all nodes in $\mathcal{P}$ are legitimate.*

Proof. In Lemma 4.8, we prove that if there exists a terminal configuration c_t such that all nodes $u \in \Psi_0 \cup \ldots \cup \Psi_{n-1}$ are legitimate and there exists a node v in Ψ_n which is non-legitimate, then P_n^k holds for $k \in \mathbb{N} \setminus \{0\}$ and $n \in \mathbb{N}$. Since the network is finite, P_n^k may not hold for k greater than the number of nodes in the system. Thus, all nodes in Ψ_n are legitimate for all $n \in \mathbb{N}$, and then all nodes in $\mathcal{P}$ are legitimate. □

4.3 Read/write atomicity

In this section, we prove that although we assumed atomicity for rules of algorithm $\mathcal{PA}$ in our previous sections, it still conforms to `read`/`write` atomicity specifications as stated in [DIM93].

Lemma 4.10 (`read`/`write` atomicity) *Algorithm 3.2 supports* `read`/`write` *atomicity.*

Proof. Rule $R_1|\mathcal{F}$ contains only one `read` statement and some internal actions, and hence, works in the same manner using the read/write atomicity. Rule $R_2|\mathcal{F}$ contains one `read` statement and one `write` statement, both using the same register r. This rule should be written as two rules to follow the rules of the read/write atomicity. We now show that in any computation in $\mathcal{S}_\mathcal{F}$, whether Rule $R_2|\mathcal{F}$ is executed in one or two steps, is indistinguishable to any processor. Let P_i be the owner of r and $P_{j_1}, P_{j_2}, \ldots$ be the direct ancestors (which have read access to r) of P_i. Then, for any P_i and P_{j_k}, the following computations are indistinguishable to any processor:

$\ldots, \mathtt{read}_i(r), \mathtt{read}_{j_k}(r), \mathtt{write}_i(r), \ldots$

$\ldots, \mathtt{read}_{j_k}(r), \mathtt{read}_i(r), \mathtt{write}_i(r), \ldots$

Therefore, Rule $R_2|\mathcal{F}$ as written in Algorithm 3.1 works in the same manner using the `read`/`write` atomicity if it is written as two separate rules. □

4.4 Necessary condition for self-stabilization

In [Duc98b], the idempotency property of order $k \in \mathbb{N} \setminus \{0\}$ was shown a necessary condition to insure termination of a distributed (non-stabilizing) algorithm based on an r-operator. In this section, we show that strict idempotency is a necessary condition to insure self-stabilization of a distributed algorithm based on an r-operator.

proposition 4.1 *If algorithm $\mathcal{PA}$ is instantiated with n-ary r-operators that are not strictly idempotent as $\mathcal{F}_v$, then $\mathcal{PA}$ is not self-stabilizing.*

Proof. The proof is by exhibiting a particular network and initialization for which the $\mathcal{PA}$ algorithm does not stabilize.

Consider the distributed system of figure 1 and suppose processor P_v uses for its local computations the binary r-operator $\triangleleft$ defined by: for all x and y in A, $x \triangleleft y = x \star r(y)$ where $\star$ is an s-operator on A and r is an isomorphism of $(A, \star)$ (see Section 2.3). Let a be an element of A verifying $a = r(a)$ and consider the case where the processor P_v is initialized with a datum b greater (in sense of $\preceq_\star$, the total order relation induced by $\star$) than a: $a \preceq_\star M_v[0] = b$. We suppose moreover that $M_v[1] = e$ at the beginning.

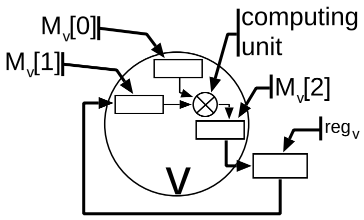

Figure 1: A simple network with three processors

Without any subsequent transient fault, the register reg_v and the local variables of P_v stabilize on the value b. Now suppose that, after a transient failure, the register reg_v contains an erroneous datum a. This datum will be copied into $M_v[1]$ and, after a computation, in $M_v[2]$. The computation will then terminate on the erroneous result a (because $b \star a = a$ and $r(a) = a$). Hence, the $\mathcal{PA}$ algorithm instantiated with $\triangleleft$ is not self-stabilizing. □

5 Applications

In this section, we briefly give some examples of r-operators designed to solve particular problems.

In the first two applications given in this section, the initial data are only used to differentiate a particular set of nodes from the others, leading to a semi-uniform algorithm: all nodes execute the same overall code, but based on some hardwired values, some internal functions may not have the same behaviour. On current networks, it is common to have unique identifiers for all nodes (through the hardware network device) so those assumptions do not clash with self-stabilization assumptions.

5.0.1 Distance computation, shortest dipath spanning tree and forest

The distance computation from a main source u in a network is solved with the binary r-operator **minc** defined as $\mathbf{minc}(x, y) = \mathbf{min}(x, y+1)$ on[4] $A = \overline{\mathbb{Z}} = \mathbb{Z} \cup \{\pm\infty\}$, with $M_u[0] = 0$ and $M_v[0] = +\infty$ for all processor P_v such that $v \neq u$.

The operator **minc** is based on the s-operator **min** and on the r-mapping $r(x) = x + 1$. **min** defines a total ordering on $\overline{\mathbb{Z}}$. Moreover $x \prec_{\mathbf{min}} r(x)$ for any $x \in \overline{\mathbb{Z}}$, which means that the r-operator **minc** is strictly idempotent.

After stabilization of Algorithm 3.1 instantiated with **minc**, a shortest path spanning tree rooted at the main source is maintained by the knowledge, on each node, of one incoming variable containing the smallest datum of all those in the incoming variables.

If there are several main sources, each node will be in the tree which has the closest root from itself. That solves the shortest path spanning forest.

5.0.2 Single and multiple source shortest path problem

This problem is similar to the distance computation, except that the weight of each edge is not always equal to 1. We define $\mathbf{minc}_w$ as a n-ary r-operator on $\overline{\mathbb{Z}}$ (on each node v, $n = \delta^-(v)$):

$$\mathbf{minc}_w\,(x_0, \ldots, x_{n-1}) = \mathbf{min}\,(x_0, x_0 + w_1, \ldots, x_{n-1} + w_{n-1})$$

If each w_i is strictly larger than 0, then each r-mapping verifies $x \prec_\star r_i(x) = x + w_i$. In this case, the $\mathbf{minc}_w$ n-ary r-operator is strictly idempotent.

As for the distance computation, the initial data are initialized with 0 on the main source and $+\infty$ on the other vertices. After stabilization of Algorithm 3.1 instantiated with $\mathbf{minc}_w$, and assuming each w_i is the weight of the i^{th} incoming edge of the processor, a solution to the single source shortest path problem is obtained.

As for the shortest path spanning forest, the same operator solves the problem defined with multiple main sources.

5.0.3 Depth-first-search tree

For this algorithm, we assume each processor is affected a unique identifier, and that a total order can be defined on those identifiers. We consider the set A of the ordered lists of identifiers and the s-operators $\star$ which gives the smallest list using the lexicographical order on the lists of identifiers.

For instance, if the identifiers are letters $a, b, c, \ldots$ ordered by the alphabetical order, the operator $\star$ gives the following results:

$$\begin{array}{lcl} (a, b, d, e,) \star (a, b, c, d, e, f) & = & (a, b, c, d, e, f) \\ (a, a) \star (a) & = & (a) \\ (f, g, f) \star (f, f) & = & (f, f) \end{array}$$

[4]For implementation purpose, due to the impossibility of coding an infinite set, one might consider the set $A = \{0, \ldots, \mathrm{Diam}(G)\}$ and r a bijective mapping from $\{0, \ldots, \mathrm{Diam}(G) - 1\}$ into $\{1, \ldots, \mathrm{Diam}(G)\}$ defined by $r(x) = x + 1$. The results of this paper are still true with these assumptions.

Moreover, we consider on each node v the r-mapping When implementing this application, it is sufficient to consider the lists of at most $|V|$ elements, and a bijective mapping r from $A \backslash \{$ lists of size $|V|\}$ into $A \setminus \{$ empty list $\}$. The results of this paper remain valid with these assumptions.

$$\begin{array}{rcl} A & \to & A \\ l & \mapsto & r(l) = l \cup (v) \end{array}$$

where $\cup$ is the concatenation of two lists. We define the binary r-operator **lexicat** by: $\textbf{lexicat}(l_1, l_2) = l_1 \star r(l_2)$.

We have, for any list l, $l \prec_\star r(l)$ and then the **lexicat** r-operator is strictly idempotent. Moreover, the lexicographical order on the lists of names of vertices is a total order relation, because the names of vertices are supposed to be totally ordered.

We suppose that a main source has the smallest name of the set of all the names of the vertices. If, on each node v of the network, the initial datum $M_v[0]$ is initialized by the list (v), then after stabilization of Algorithm 3.1 instantiated with **lexicat**, a depth-first-search tree is maintained by the knowledge, on each node, of one incoming variable containing the smallest list of all those in the incoming variables.

6 Conclusion

We characterized a set of silent tasks that can be solved in a self-stabilizing way using a single algorithm parameterized by a r-operator. This operator must conform to some restrictions, namely being strictly idempotent and that the s-operator it is based on induces a total order on the elements of the computing set A. Unlike previous self-stabilizing algorithms, our parametric algorithm can be used in any directed graph even if its underlying topology does not allow the building of well known topologies (*i.e.* trees or rings).

Although being simple in its formulation, the parameterized algorithm can be applied to a broad range of distributed tasks such as Distance computations, Shortest path calculus and Depth-first search tree construction.

To conclude, we would like to point out two current research interests of the authors.

1. Extensions to r-operators have been proposed in [Duc98a], such as *generalized* r-operators (where the r-mappings need several variables). We conjecture that, under similar hypotheses as those described in this paper, these operators lead to self-stabilizing systems at no additional cost.
2. In this paper, we assume the s-operator $\star$ (upon which the r-operators are based) defines a total ordering. The authors are investigating the possibility that the system remains self-stabilizing when the order relation induced by $\star$ is only partial.

References

[ABDT98] L. O. Alima, J. Beauquier, A. K. Datta and S. Tixeuil. *Self-stabilization with Global Rooted Synchronizers.* ICDCS98 Proceedings of the 18th International Conference on Distributed Computing Systems, 1998.

[AHOPV94] D. Alstein, J.-H. Hoepman, B. E. Olivier and P. I. A. Van Der Put. *Self-stabilizing mutual exclusion on directed graphs.* In Computer Science in the Netherlands (Utrech, The Netherlands, 1994), E. Backer (Ed.), Stichting MathematischCentrum, Amsterdam, pp. 42-53.

[AKY90] Y. Afek, S. Kutten and M. Yung. *Memory-efficient self-stabilization on general networks.* In Workshop on Distributed Algorithms, pp. 15-28, 1990.

[BDDN95] B. Bourgon, S. K. Das, A. K. Datta, and V. Natarajan. *A Self-Stabilizing Shortest Path Algorithm in a DAG.* Fourteenth IEEE International Phoenix Conference on Computers and Communications, Scottsdale, Arizona, pp. 341-345, March 29-30, 1995.

[D74] E. W. Dijkstra. *Self-Stabilizing Systems in Spite of Distributed Control.* Communications of the ACM 17,11 pp. 643-644, 1974.

[DGS96] S. Dolev, G. Gouda, and M. Schneider. *Memory Requirements for Silent Stabilization.* Proc. of the 15th Annual ACM Symp. on Principles of Distributed Computing, pp. 27-34, 1996.

[DIM93] S. Dolev, A. Israeli and S. Moran *Self Stabilization of Dynamic Systems Assuming Only Read/Write Atomicity.* Distributed Computing, Vol. 7, pp. 3-16, 1993.

[Duc97] B. Ducourthial. *Properties of the r-Operators to Insure Termination of Distributed Computations.* Technical Report 1140, Laboratoire de Recherche en Informatique, Bât. 490. Université Paris-Sud, 91405 Orsay, France, December 1997.

[Duc98a] B. Ducourthial. *Generalization of the r-Operators.* Technical report, Laboratoire de Recherche en Informatique, Bât. 490. Université Paris-Sud, 91405 Orsay, France, 1998. To appear.

[Duc98b] B Ducourthial. *New Operators for Computing with Associative Nets.* Proceedings of the Fivth International Colloquium on Structural Information and Communication Complexity (SIROCCO'98), Amalfi, Italia, to appear.

[S93c] M. Schneider. *Self-stabilization.* ACM Computing Surveys, vol. 25, pp 45-67, 1993.

[Tel94] G. Tel. *Introduction to Distributed Algorithms.* Cambridge University Press, Cambridge, UK, 1994.