Self-stabilization with r-operators

Technical Report n° 2001/358,
HEUDIASYC, UMR CNRS 6599,
Université de Technologie de Compiègne.

Bertrand Ducourthial
Laboratoire Heudiasyc, UMR CNRS 6599,
Université de Technologie de Compiègne,
BP 20529,
F60205 Compiègne Cedex, France.
Bertrand.Ducourthial@utc.fr

Sébastien Tixeuil
Laboratoire de Recherche en Informatique, UMR CNRS 8623,
Bâtiment 490, Université de Paris Sud,
F91405 Orsay Cedex, France.
tixeuil@lri.fr

February 5, 2001

Abstract

This paper 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, the solution presented in our paper can be useful for a large class of distributed systems.

Keywords: self-stabilization – r-operators – distributed systems – read-write atomicity – unfair scheduling

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* [10, 18]. Introduced by Dijkstra in [10], 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 [11] are tasks where the communications between the processors are fixed from some point of the execution. In addition to simplicity implied by the silence property, silent distributed algorithms may use fewer resources in terms of communication operations and communication bandwidth allocation. 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. When the algorithm checks that a register needs to be changed before performing a write operation, all write operations may be eliminated when the silent system has reached a legitimate configuration. Static problems, which consist to compute a global result in the system, lead to silent tasks: communications cease when the terminal configuration is reached. Examples of such tasks include leader election, spanning tree construction or single source shortest path algorithms. Note that many tasks fundamental to distributed systems are inherently non silent. Such tasks include mutual exclusion or token passing, where the contents of communication registers have to change infinitely often in every possible execution of the system.

Historically, research in self-stabilization over general networks has mostly covered undirected networks where bidirectional communication is feasible (the Update protocol of [12], or the algorithms presented in [13, 4]). Bidirectional communication is usually heavily used in bidirectional self-stabilizing systems to compare one node state with those of its neighbors and check for consistency. The self-stabilizing algorithms that are built upon the paradigm of local checking (see [6, 7]) use this scheme.

The lack of bidirectional communication was overcome in recent papers using several techniques. Strong connectivity (which is a weaker requirement than bidirectionality) was assumed to build a virtual well known topology on which the self-stabilizing algorithm may be run (a tree in [2], a ring in [3]). As many self-stabilizing algorithms exists for rings ([10]) or trees ([1]) in the literature, these constructions may be used to reuse existing algorithms in general networks.

The restriction of having either bidirectional communication media or strongly connected unidirectional networks 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 every node infinitely often. However, there exists several silent tasks for which global communication is not required. For example, the single source shortest path task only requires that a directed path exists from a node to any other node, but not the converse.

In [5], Attie *et al.* used the formalism of Iteration Systems to give sufficient conditions for convergence of systems solving related tasks. An important subset of silent tasks is computing routing metrics (*e.g.* to perform maximum flow routing), so that Gouda and Schneider in [16] provided a condition-based approach to determinate if a particular routing metric is maximizable. Silent tasks have been solved in a self-stabilizing way on directed graphs that are not strongly connected in [9], but the underlying network was assumed having no cycle (DAG). The absence of cycles permits to avoid cases where corrupted data moves forever in the system, preventing it from stabilizing.

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. As in [5], our solution is by giving a condition on the distributed algorithm. However, in [5], the condition is given in terms of global system property, while our condition is independent of the task to be solved, and is only determined by the algebraic properties of the function computed locally by the algorithm. While our objectives – deciding whether an operator-based algorithm is self-stabilizing – are different from those of [16] – deciding whether a metric is maximizable –, our two approaches do share common points. The *idempotency* of the local operator plays a crucial role in the theory of [16], while the *strict idempotency* (see Definition 5) of the local operator plays a fundamental role here. Unlike many approaches, our solution does not require any knowledge about the network: no size, diameter, maximum degree are needed.

To this purpose, we provide a parametrized algorithm that can be instantiated with a local function. 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.

In addition, since our approach is condition based, there is no additional layer used to make an algorithm that satisfies this condition tolerant to transient failures. In fact, when no transient faults appear in the system, the performance suffers no overhead. Our system performs in the fine-grained read/write atomicity model, where read and write actions occurring on different processors may be interleaved at will. Also worth of noting is the fact that our algorithm does not require any fairness property on the executions other than simple progression.

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 5. Application to fundamental problems in distributed computing area are presented in Section 6. We discuss the extension of our ideas and make some concluding remarks in Section 7.

2 Model

In this section, we define the concept of a distributed system, basically a graph whose vertices model processors and arcs model the communication media allowing them to exchange information. In this context, self-stabilizing distributed systems are introduced and defined as systems that eventually match a given specification in spite of transient failures that may corrupt all volatile variables. Then, r-operators are defined as functions on a set of values $\mathbb{S}$ that have several algebraic properties. Those r-operators are used as the core of our parametric algorithm to make it automatically self-stabilizing.

2.1 Distributed System

Graph modeling 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 pairs of vertices of V $(v_1, v_2 \in V)$. Each vertex u in V represents a processor P_u of 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 indexed from 1 to $\delta^-(v)$. We denote by $\mathsf{Ind}(u, v)$ the index of the incoming edge (u, v) in v.

A *directed path* $P_{v_0 \to v_k}$ 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. A *cycle* is a directed path where $v_0 = v_k$. A *loop* is a cycle of length 1. To make some wording shorter, we define an *empty path* as an empty list of edges. Its length is 0. An *elementary path* is a path where each vertex is encountered at most once. An empty path is elementary. A digraph without any cycle is called a *directed acyclic graph (DAG)*.

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 (assuming there exists at least one such path). The *diameter* of a digraph G, denoted by $\mathrm{Diam}(G)$, is the maximum of the distances between all couples of vertices in G between which a distance is defined. 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 (possibly empty) from v to w and a directed path from w to v. G is *strongly connected* if any v in G has the same 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 $\Gamma_G^{+1}(v)$. The *descendants* of v are all the vertices w such that there exists a path from v to w. Their set is denoted by $\Gamma_G^{+}(v)$. 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 $\Gamma_G^{-}(v)$. Finally, $\Gamma_G^{-k}(v)$ denotes the set of all ancestors u of v such that $\mathrm{d}_G(u, v) = k$, while $\Gamma_G^{+k}(v)$ denotes the set of all descendants w of v such that $\mathrm{d}_G(v, w) = k$. An *orphan* of a digraph G is a vertex that has no ancestor. A *patriarch* s of a digraph G is a vertex such that there exists a directed path from s to all other vertices.

Communications. A communication from processor P_u to processor P_v is only feasible if vertex u is a direct ancestor of vertex v in 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 to use it.

The processors maintain two types of variables: *field variables* and *local 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 by its direct descendants.

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

Processors. A processor is a deterministic sequential machine that runs a single 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. Internal actions of processors are not significant to their neighbors because the neighbors have no access to the variables that are manipulated by those actions. The `read` and `write` actions are the only way for two processors to communicate.

Protocol. A *distributed algorithm* $\mathcal{P}$ (or *protocol*) is a collection of local algorithms. A distributed system $\mathcal{S}$ executes $\mathcal{P}$ if every processor of $\mathcal{S}$ executes a local algorithm of $\mathcal{P}$.

Configuration and Executions. 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 system configuration. An *execution* e is a sequence of configurations $c_1, c_2, \ldots$ such that for $i = 1, 2, \ldots$, the configuration c_{i+1} is reached from c_i by a single step of one processor (a single step of one processor being internal computations followed by either a `read` or a `write` action, but not both). c_1 is called the *initial configuration* of e. This scheduling policy is known as the `read-write` atomicity model (see [13]). It should be noted that `read` and `write` actions are executed asynchronously,

All executions considered in this paper are assumed to be *maximal* meaning that the sequence is either infinite, or it is finite and no action is enabled in the final configuration. Note that no particular fairness assumptions are made.

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

2.2 Self-stabilization

A *specification* of a distributed system is a predicate on executions of that system. A system *matches its specification* if all its possible executions satisfy the specification. If we consider only *static* problems (*i.e.*, problems whose solutions consist of computing some global result), the specification can be given in terms of a set of configurations (such specifications are called *silent specifications*). Every execution satisfying the specification would be a sequence of such configurations. The set of configurations that satisfy 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 execution in $\mathcal{E}$. In addition, if the attractor is closed, then any subsequent execution of the algorithm remains in the same set of configurations.

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

Definition 2 ($\mathcal{L}$-stabilization) *Let $\mathcal{L}$ be a non-empty set of configurations ($\mathcal{L} \subset \mathcal{C}$) of a distributed system $\mathcal{S}$. $\mathcal{S}$ is* $\mathcal{L}$-stabilizing *if and only if $\mathcal{L}$ is a closed attractor for $\mathcal{C}$.*

In non-stabilizing distributed systems, possible executions 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.

Definition 3 (Self-stabilization) *Let $\mathcal{P}$ be a distributed algorithm (or protocol). Let $\mathcal{S}$ be a distributed system that executes $\mathcal{P}$. Let $\mathcal{L}$ be a set of configurations of $\mathcal{S}$ that defines a silent specification. Protocol $\mathcal{P}$ is self-stabilizing if $\mathcal{S}$ is $\mathcal{L}$-stabilizing.*

Complexity. The efficiency of a self-stabilizing protocol can be measured in terms of time and memory space needed to achieve the stabilization. Since we consider asynchronous systems, in order to find the time complexity of the protocol, we must assume a bound on the communication delay between any two neighbors. We refer to this bound as a *time unit*. Any finite sequence of internal actions is assumed to be instantaneous. The *time complexity* of a self-stabilizing protocol is the time needed to reach a configuration in $\mathcal{L}$ after the faults cease to occur. The *space complexity* of a self-stabilizing protocol is the memory space needed to hold the local and field variables (to maintain the information to communicate among the processors). This complexity can be given using the number of states for the automata used to describe the system, or the number of bits needed to hold the variables of the algorithm. We will use the later approach in this paper.

3 Parametric distributed algorithm

In this section, we describe a parametric distributed protocol composed, on each node, of a local algorithm denoted $\mathcal{PA}$ and parametrized by a local function. When applied on node v by processor P_v, $\mathcal{PA}$ simply gets input data from incoming neighbors of v, computes the local function and finally makes the result available to the outgoing neighbors of v.

3.1 Programming Notation

The local algorithm of each processor consists of a set of actions: $\langle \textit{action} \rangle \cdots \langle \textit{action} \rangle$. Each *action* has the form:

$$\langle \textit{guard} \rangle \longrightarrow \langle \textit{statement} \rangle$$

A *guard* is a boolean expression over the local variables of a processor and the field variables of its direct ancestors. A *statement* is allowed to 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*.

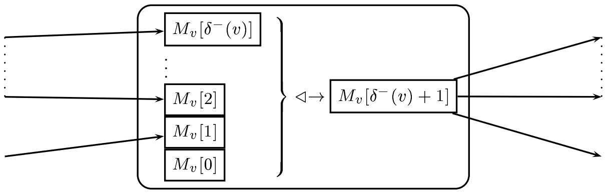


Figure 1: Node v Layout

3.2 Informal 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 $M_v[1], \ldots, M_v[\delta^-(v)]$, the *incoming variables*. To store the result of local execution, a single field variable is used at P_v: $M_v[\delta^-(v)+1]$, the *outgoing variable*, which is stored in the field register reg_v.

Notation 1 *We denote by $\mathbb{S}$ the set of inputs of the distributed system $\mathcal{S}$: all the values that can be stored in registers and local variables of processors of $\mathcal{S}$ belong to $\mathbb{S}$.*

On each processor P_v of $\mathcal{S}$, the local algorithm $\mathcal{PA}$ maintains a function denoted as $\lhd_v$ defined on $\mathbb{S}$ by:

$$\begin{array}{llcl} \lhd_v\colon & \mathbb{S}^{\delta^-(v)+1} & \to & \mathbb{S} \\ & M_v[0..\delta^-(v)] & \mapsto & \lhd_v\ (M_v[0..\delta^-(v)]) \end{array}$$

The input to $\lhd_v$ are the initial datum $M_v[0]$ of v and the values of the local variables copied from the direct ancestors into $M_v[1..\delta^-(v)]$. Function $\lhd_v$ outputs the current state of the processor computing the function. This result is stored in $M_v[\delta^-(v)+1]$.

The dynamic of each processor P_v is as follows:

(1) P_v reads data in the registers of its direct ancestors and copies them into its incoming variables

$$M_v[1], M_v[2], \ldots, M_v[\delta^-(v)]$$

(2) P_v performs a local execution using function $\lhd_v$, 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:

$$M_v[\delta^-(v)+1] \quad \leftarrow \quad \lhd_v\ \big(M_v[0], \ldots, M_v[\delta^-(v)]\big)$$

Figure 1 represents node v along with its M_v registers, while Figure 2 depicts a simplified vision of node v, without showing variables $M_v[1]$ to $M_v[\delta^-(v)]$. For the sake of simplicity, this simplified version is used in the forthcoming figures.

3.3 Formal Description

More formally, the local algorithm $\mathcal{PA}$ parametrized by a function $\lhd_v$ — and denoted by $\mathcal{PA}|_{\lhd_v}$ when necessary — consists of two guarded rules (see Algorithm 3.1).

The first rule, called *Copying rule* and denoted by R_1, 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

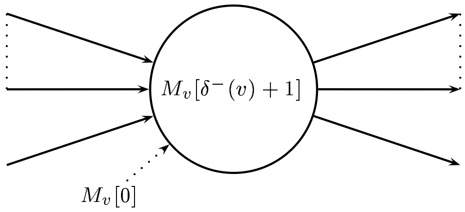


Figure 2: Node v Simplified Layout

Algorithm 3.1 The $\mathcal{PA}|_{\lhd_v}$ local parametric algorithm at node v (parameter is function $\lhd_v$)

Copying (R_1):

$$\left(P_u \in \Gamma_G^{-1}(v) : \begin{array}{ll} & (\alpha := \texttt{read}\,(M_u[\delta^-(u)+1])) \\ \wedge & M_v[\mathsf{ind}(u)] \neq \alpha \end{array}\right)$$

$$\longrightarrow M_v[\mathsf{ind}(u)] := \alpha \,;$$

Computing $(R_2|_{\lhd_v})$:

$$(\texttt{read}\,(M_v[\delta^-(v)+1]) \neq \lhd_v\, (M_v[0..\delta^-(v)]))$$

$$\longrightarrow \texttt{write}\,(M_v[\delta^-(v)+1], \lhd_v\, (M_v[0..\delta^-(v)]));$$

the field variable $M_u[\delta^-(v)+1]$ twice, and implements the `read`/`write` atomicity. In R_1, an expression of the form $\langle$*left*$\rangle$:= $\langle$*right*$\rangle$ 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 (or $R_2|_{\lhd_v}$ when necessary), computes the $\lhd_v$ function with the previously copied variables as input and stores the result into $M_v[\delta^-(v)+1]$.

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

4 r-operators

As explained in previous section, we concentrate on a single local algorithm that processes at each node the incoming data through a given operator that parameterizes the algorithm. Such an operator is sufficient to describe the behavior of the whole system. In this paper, we investigate sufficient conditions on the operator so that the system is self-stabilizing for a given specification. We begin to define the r-operators as an extension of Tel's infimum functions.

4.1 Infimum Functions

Thanks to Tel results (see [19]), the distributed protocol described above terminates when each $\mathcal{PA}$ local parametric algorithm is instantiated by an infimum over the set of inputs $\mathbb{S}$. An *infimum* (hereby called an *s-operator*) $\oplus$ over a set $\mathbb{S}$ is an associative, commutative and idempotent binary operator. Such an operator defines a partial order relation $\preceq_\oplus$ over the set $\mathbb{S}$ by: $x \preceq_\oplus y$ if and only if $x \oplus y = x$. Moreover, [19] assumes that there exists a greatest element on $\mathbb{S}$, denoted by $e_\oplus$, and verifying $x \preceq_\oplus e_\oplus$ for every $x \in \mathbb{S}$. If necessary, this element can be added to $\mathbb{S}$. In the following, we assume that an s-operator admits such an element in its definition of set $\mathbb{S}$.

Hence, the $(\mathbb{S}, \oplus)$ structure is an *Abelian idempotent semi-group*[1] (see [8]) with $e_\oplus$ as identity element. When parametrized by such an s-operator $\oplus$, the parametric local algorithm $\mathcal{PA}|_\oplus$ yields a silent distributed protocol. However, some counter examples show that such a protocol is not self-stabilizing with s-operators (see Section 5.3).

4.2 r-operators

Starting from Tel results, [14] proposed a distorted algebra — the r-algebra —, that generalizes the Abelian idempotent semi group. To ease comparison with this structure, we first recall definition and basic properties of *binary* r-operators (see [14]).

Definition 4 (Binary r-operator) *The operator $\triangleleft$ is a binary r-operator on $\mathbb{S}$ if there exists an s-operator $\oplus$ on $\mathbb{S}$ and an homomorphism of $(\mathbb{S}, \oplus)$ — called r-mapping and denoted by r — such that $\triangleleft$ satisfies, for any x and y in $\mathbb{S}$, $x \triangleleft y = x \oplus \mathsf{r}(y)$.*

The following proposition states that r-operators constitute a larger class of operators as s-operators (infimum). Hence some results proved for r-operators still apply for s-operators.

Proposition 1 *Let $\triangleleft$ be an r-operator on $\mathbb{S}$, based on the s-operator $\oplus$ and the r-mapping r. Then it satisfies the following properties : **(i)** r-associativity: $(x \triangleleft y) \triangleleft \mathsf{r}(z) = x \triangleleft (y \triangleleft z)$; **(ii)** r-commutativity: $\mathsf{r}(x) \triangleleft y = \mathsf{r}(y) \triangleleft x$; **(iii)** r-idempotency: $\mathsf{r}(x) \triangleleft x = \mathsf{r}(x)$ and **(iv)** identity: $x \triangleleft e_\oplus = x$. (where $e_\oplus$ is the identity element of $\oplus$).*

The following proposition allows us to introduce the *strict idempotency* that will be useful below.

Proposition 2 *Let $\triangleleft$ be an r-operator based on the s-operator $\oplus$. Then the following propositions are equivalent: (i) $\triangleleft$ is idempotent; (ii) for any $x \in \mathbb{S}$, $x \preceq_\oplus \mathsf{r}(x)$. Moreover, if $\triangleleft$ is idempotent, then $\mathsf{r}(e_\oplus) = e_\oplus$.*

Definition 5 (Strict idempotency) *An r-operator $\triangleleft$ based on the s-operator $\oplus$ is* strictly idempotent *if, for any $x \in \mathbb{S} \setminus \{e_\oplus\}$, $x \prec_\oplus \mathsf{r}(x)$ (i.e. $x \preceq_\oplus \mathsf{r}(x)$ and $\mathsf{r}(x) \neq x$).*

For example, the operator $\mathsf{minc}(x, y) = \mathsf{min}(x, y + 1)$ is a strictly idempotent r-operator on $\mathbb{Z} \cup \{+\infty\}$, with $+\infty$ as its identity element. It is based on the s-operator min and on the bijective r-mapping $\mathsf{r}(x) = x + 1$. Such an operator can also be defined on the finite set $\{0, 1, \ldots, 255\}$. In that case, the r-mapping is defined by $\mathsf{r}(x) = x + 1$ for $x \in \{0, \ldots, 254\}$ and $\mathsf{r}(255) = 255$.

We now define r-operators that accept an arbitrary number of arguments.

Definition 6 (r-operator) *A mapping $\lhd$ from $\mathbb{S}^n$ into $\mathbb{S}$ is an* r-operator *if there exists an s-operator $\oplus$ on $\mathbb{S}$ and $n-1$ homomorphisms (called r-mappings) $\mathsf{r}_1, \ldots, \mathsf{r}_{n-1}$ of $(\mathbb{S}, \oplus)$ such that $\lhd\ (x_0, \ldots, x_{n-1}) = x_0 \oplus \mathsf{r}_1(x_1) \oplus \cdots \oplus \mathsf{r}_{n-1}(x_{n-1})$ for any $x_0, \ldots, x_{n-1}$ in $\mathbb{S}$.*

Definition 7 *The r-operator $\lhd$ from $\mathbb{S}^n$ into $\mathbb{S}$ based on the s-operator $\oplus$ is idempotent (resp. strictly idempotent) if, for any $x \in \mathbb{S}$ (resp. $x \in \mathbb{S} \setminus \{e_\oplus\}$) and any r-mapping r_i ($1 \leq i < n$), $x \preceq_\oplus \mathsf{r}_i(x)$ (resp. $x \prec_\oplus \mathsf{r}_i(x)$, i.e. $x \preceq_\oplus \mathsf{r}_i(x)$ and $\mathsf{r}_i(x) \neq x$).*

4.3 Hypothesis

In the rest of the paper, we assume that every processor P_v of $\mathcal{S}$ uses the $\mathcal{PA}|_{\lhd_v}$ local algorithm instantiated with an r-operator $\lhd_v$. We do not require that all those r-operators $\lhd_v$ be the same for each node v. However we assume the following hypothesis.

Hypothesis 1 *The r-operators used by the processors of $\mathcal{S}$ are all based on the same s-operator $\oplus$, and the ordering relation defined by $\oplus$ is a total order relation on $\mathbb{S}$.*

[1]The prefix *semi* means that the structure cannot be completed to obtain a group, since the law $\oplus$ is idempotent.

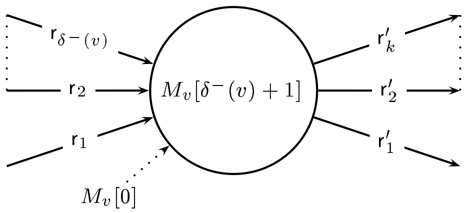


Figure 3: Node v Simplified Layout

Hypothesis 2 *All operators are defined on set $\mathbb{S}$ (see Notation 1). Moreover, if $\mathbb{S}$ is infinite, then the two following conditions are satisfied: (i) any sequence of strictly increasing elements is unbounded and (ii) no ROM initial datum contains the $e_\oplus$ element.*

Hypothesis 1 implies that for any set of data $\{x_0, \dots, x_k\}$, we have $\oplus\{x_0, \dots, x_k\} \in \{x_0, \dots, x_k\}$. Hypothesis 2 is satisfied by $\mathbb{N}$ or $\mathbb{Z}$, but not by $\mathbb{Q}$ or $\mathbb{R}$. In practice, since memory is only available in bounded flavor, all considered sets are finite.

When each processor performs local computations using an r-operator, each edge of the distributed system is associated with an r-mapping (see figure 3). The result of an r-mapping associated to an edge is called an *r-augmented variable*. Note that if all the r-mappings of a node v are equal, the operator used by processor P_v is a binary r-operator, while if they are all equal to the identity function Id, it is an s-operator.

Notation 2 *We call $\mathsf{r}_v^{\mathsf{Ind}(u,v)}$ the r-mapping corresponding to the incoming edge (u, v) of v. When no confusion is possible, we also denote this r-mapping by r_v^i, with $1 \leq i \leq \delta^-(v)$. To shorten our notations, we assume that, for any node v, there is also an r-mapping r_v^0 corresponding to the initial datum, and we state that $\mathsf{r}_v^0 \equiv \mathsf{Id}$.*

Notation 3 *Let $P_{u_0 \to u_n}$ be a path $(u_0, u_1) \dots (u_{n-1}, u_n)$ in G. To each of its edges (u_i, u_{i+1}) corresponds the r-mapping $\mathsf{r}_{u_{i+1}}^{\mathsf{Ind}(u_i,u_{i+1})}$. The composition of such r-mappings along the path is called r-path-mapping and is denoted as $\mathsf{r}_{P_{u_0 \to u_n}} = \mathsf{r}_{u_n}^{\mathsf{Ind}(u_{n-1},u_n)} \circ \dots \circ \mathsf{r}_{u_1}^{\mathsf{Ind}(u_0,u_1)}$ (French composition of the applications).*

5 Proof of Correctness

In this section, we prove that **(i)** rules of the $\mathcal{PA}$ local parametric algorithm can be considered as atomic although Rule R_2 contains one `read` and one `write` actions (see Section 5.1), **(ii)** if every local parametric algorithm is instantiated by an r-operator that is strictly idempotent, the resulting distributed protocol is self-stabilizing for the intended specification (see Sections 5.2 and 5.3), whereas **(iii)** if some r-operators are not strictly idempotent, the resulting distributed algorithm is not self-stabilizing (see Section 5.4), and finally **(iv)** we investigate complexity results related with the stabilization time of our algorithm (see Section 5.5).

5.1 Read/write Atomicity

In this section, we prove that our local parametric Algorithm $\mathcal{PA}$ conforms to the `read`/`write` atomicity specifications as stated in [13]. In the algorithm, the first rule can be safely assumed atomic, but the second rule has to be checked thoroughly.

Proposition 3 *Rules R_1 and R_2 of local parametric Algorithm $\mathcal{PA}$ can be considered as atomic.*

Proof. Rule R_1 contains only one `read` statement and some internal actions, and hence, can be considered atomic in the `read`/`write` atomicity model.

Rule R_2 contains one `read` statement and one `write` statement, both using the same register reg_u. In the `read` / `write` atomicity model, these two actions can be interleaved with other processor actions.

Now we prove that any such interleaving is equivalent to an execution where Rule R_2 is executed atomically.

Let u be the owner of reg_u and $\Gamma_G^{+1}(u)$ denote the set of u's direct descendants. Interleaving Rule R_2 at u and Rule R_2 at v ($v \neq u$) is equivalent to considering them as executing atomically since they apply to different registers (R_2 apply both of its two actions to the same register). For the same reason, interleaving R_2 at u and R_1 at $v \notin \Gamma_G^{+1}(u)$ is equivalent to considering them as executing atomically.

There remains the case when Rule R_2 at u (that includes $\mathtt{read}_u(\mathrm{reg}_u)$ action following by $\mathtt{write}_u(\mathrm{reg}_u)$ action) is interleaved with Rule R_1 at $v \in \Gamma_G^{+1}(u)$ (that includes $\mathtt{read}_v(\mathrm{reg}_u)$). The two actions involve the same register reg_u, and the resulting computation would be as follows:

$$\ldots, \mathtt{read}_u(\mathrm{reg}_u), \mathtt{read}_v(\mathrm{reg}_u), \mathtt{write}_u(\mathrm{reg}_u), \ldots$$

Now this computation is equivalent to the following, where Rule R_2 is executed atomically:

$$\ldots, \mathtt{read}_v(\mathrm{reg}_u), \mathtt{read}_u(\mathrm{reg}_u), \mathtt{write}_u(\mathrm{reg}_u), \ldots$$

Thus, any interleaved computation is equivalent to a computation where Rule R_2 is executed atomically. $\square$

5.2 Idempotency yields silent distributed computations

Proposition 4 (Silence) *Consider a distributed system $\mathcal{S}$ that satisfies Hypotheses 1 and 2, and where each processor P_v runs the local parametric Algorithms $\mathcal{PA}|_{\lhd_v}$ instantiated with a strictly idempotent r-operator $\lhd_v$. Then each execution is finite.*

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 (R_1 or R_2) 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-mappings 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 Rule R_2 is $\mathsf{r}_v^i(M_v[i])$, for one i in $\{0, \ldots, \delta^-(v)\}$, where [2]

$$\mathsf{r}_v^i\,(M_v[i]) = \begin{array}{ll} & \mathsf{r}_v^0\,(M_v[0]) \\ \oplus & \mathsf{r}_v^1\,(M_v[1]) \\ \oplus & \cdots \\ \oplus & \mathsf{r}_v^{\delta^-(v)}\,(M_v[\delta^-(v)]) \end{array}$$

instead of $y \in \mathbb{S}$, with $y = \mathsf{r}_v^i(M_v[i])$. We are now able to consider moves of the initial data during an execution of the system.

Any copying Rule R_1 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 $M_v[\mathsf{Ind}(u,v)]$ of v. This erases the previously stored expression in these incoming variables. The execution of a copying rule can be interpreted as a *horizontal move* between two neighbor nodes.

Any computing Rule R_2 moves the r-augmented datum stored into $M_v[i]$ for a single i in $\{0, \ldots, \delta^-(v)\}$, in a new expression $\mathsf{r}_v^i(M_v[i])$ which is stored into $M_v[\delta^-(v)+1]$. This erases the previously stored

[2]This is possible because we supposed that the s-operator $\oplus$ induces a total order relation (see Hypothesis 1). Note that we stated that r_v^0 is the identity function Id for any node v (see Notation 2).

expression in $M_v[\delta^-(v)+1]$. The execution of a computing rule can be interpreted as a *vertical move* inside one node.

From the construction of the algorithm, the two following properties hold in any execution at any node v: (i) between any two executions of Rule R_2 on v, at least one Rule R_1 is executed on v, (ii) between any two executions of Rule R_1 on v, at least one Rule R_2 is executed on some direct ancestor u of v. Since the network is finite, it is sufficient to prove that either a node v may not execute Rule R_1 forever or that a node v may not execute Rule R_2 forever. We choose the later approach.

Suppose that there exists some node v of $\mathcal{S}$ such that v executes R_2 infinitely often. Since initially there are a finite number of values, there exists one value ω that is moved vertically through Rule R_2 infinitely often at node v. By hypotheses, any r-mapping r verifies that, for any $y \in \mathbb{S}, y \prec_\oplus \mathsf{r}(y)$. From Hypothesis 2, there exists a configuration c such that either **(1)** $M_v[0] \prec_\oplus \mathsf{r}(\dots(\omega)\dots)$ or **(2)** $M_v[0] = \mathsf{r}(\dots(\omega)\dots) = e_\oplus$, the maximum element of $\mathbb{S}$ (in sense of $\preceq_\oplus$), holds.

(1) For the first case, consider that the result of the local computation is always lower or equal (in the sense of $\oplus$) than $M_v[0]$. It is then different from $\mathsf{r}(\dots(\omega)\dots)$. Hence, from this point, no vertical move of ω may occur at v, which contradicts the hypothesis.

(2) In the second case, where $\mathsf{r}(\dots(\omega)\dots) = M_v[0] = e_\oplus$, any subsequent r-mapping would leave the result unchanged. (idempotency yields $r(e_\oplus) = e_\oplus$, see Proposition 2). Then from this point, no additional vertical move of ω may occur at v, which also contradicts the hypothesis.

In any of the two possible cases, the hypothesis that there exists one node v that executes Rule R_2 infinitely often is contradicted, which proves that rule R_2 is executed only a finite number of times in system $\mathcal{S}$. From the strict interleaving relation between rules R_2 and R_1, we can conclude that no infinite execution (in the sense that some rules remain enabled) may occur in the distributed system we consider. □

5.3 Strict-idempotency yields self-stabilizing distributed computations

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. We state:

Definition 8 (Legitimate output)

$$\mathcal{L}\left(M_v[\delta^-(v)+1]\right) = \oplus \left\{ \begin{array}{l} \mathsf{r}_P(M_u[0]), \\ u \in \Gamma_G^-(v), \\ P_{u\to v} \textit{ an elementary path} \end{array} \right\}$$

For any node v, if $M_v[\delta^-(v)+1] = \mathcal{L}\left(M_v[\delta^-(v)+1]\right)$ the outgoing variable $M_v[\delta^-(v)+1]$ is *legitimate*. An incoming variable $M_v[\mathsf{Ind}(u,v)]$ is *legitimate* if $M_v[\mathsf{Ind}(u,v)] = \mathcal{L}\left(M_u[\delta^-(u)+1]\right)$. For any node v, the initial datum $M_v[0]$ is legitimate in any configuration, since this datum is stored in Read Only Memory. A node v is *legitimate* if its outgoing variable $M_v[\delta^-(v)+1]$ is legitimate. Given these definitions, a *legitimate configuration* is simply a configuration where all nodes are legitimate: the set $\mathcal{L}$ of legitimate configurations for $\mathcal{S}$ is defined as: $\forall l \in \mathcal{L}, \forall v$ in V, v is legitimate in configuration l.

We now wish to prove that the system is $\mathcal{L}$-stabilizing (see Definition 2). This is by proving that in any terminal configuration (which is eventually reached by Proposition 4), all nodes in the system are legitimate (and a configuration in $\mathcal{L}$ is reached).

Since we supposed that the s-operator $\oplus$ defines a total order, one can sort, for any given configuration, the legitimate incoming variables of all the nodes of the system.

Definition 9 (Λ^c) *Let c be a configuration, Λ^c denotes the sorted set of all legitimate incoming variables in c. Using the total ordering relation $\prec_\oplus$ induced by $\oplus$,*

$$\Lambda^c = \{\lambda_0^c, \lambda_1^c, \dots, \lambda_{k_c}^c\}$$

with $\lambda_i^c \prec_\oplus \lambda_j^c$ for $i < j$. For any i in $\{0,\dots,k_c\}$, λ_i^c denotes the i^{th} element of Λ^c.

Moreover, for any given configuration, one can sort out the nodes of the system into several distinct sets. Informally, the partition of V into the Ψ_n^c sets collects nodes sharing a legitimate variable. Then Ψ_0^c contains nodes having the smallest legitimate r-augmented variable λ_0^c, Ψ_1^c contains nodes having smallest remaining r-augmented legitimate variable $\lambda_1^c \in \Lambda^c \setminus \{\lambda_0^c\}$, and so on. Since all nodes have at least one legitimate variable ($M_v[0]$ is stored in ROM), the Ψ_i^c sets define a partition of V at configuration c.

Definition 10 (Ψ_n^c) *Let c be a configuration, Ψ_n^c is the set of the nodes of the system satisfying the following criteria:*

$$\Psi_n^c = \left\{ v \in V \textit{ s.t. } \begin{array}{c} \exists i, 0 \leq i \leq \delta^-(v), \\ \mathcal{L}\left(M_v[i]\right) = M_v[i] \\ \textit{(i.e. } M_v[i] \textit{ is legitimate)}, \\ \mathsf{r}_v^i(M_v[i]) = \lambda_n^c, \\ \forall n' < n, v \notin \Psi_{n'}^c \end{array} \right\}$$

Lemma 1 *In any configuration c, there is no legitimate outgoing variable smaller than λ_0^c.*

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

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

Then we have

$$\oplus \left\{ \begin{array}{l} \mathsf{r}_{P_{u\to v}}\left(M_u[0]\right), u \in \Gamma_G^-(v), \\ P_{u\to v} \text{elementary path} \end{array} \right\} \prec_\oplus \lambda_0^c$$

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

$$M_u[0] \prec_\oplus \mathsf{r}_P\left(M_u[0]\right) \prec_\oplus \lambda_0^c$$

which is impossible, by definition of λ_0^c. □

Lemma 2 *Any node v of Ψ_0^c verifies $\mathcal{L}(M_v[\delta^-(v)+1]) = \mathcal{L}(M_v[0]) = M_v[0] = \lambda_0^c$.*

Proof. We have:

$$\mathcal{L}(M_v[\delta^-(v)+1]) = \mathsf{r}_v^i\left(\mathcal{L}\left(M_v[i]\right)\right) \text{ for one } i \text{ in } \{0, \ldots, \delta^-(v)\}$$

and since v is in Ψ_0^c,

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

But, following Lemma 1, no legitimate variable is smaller than λ_0^c, which means that

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

Now suppose that $i > 0$:

$$\mathcal{L}(M_v[\delta^-(v)+1]) = \mathsf{r}_v^i\left(\mathcal{L}(M_v[i])\right)$$

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

$$\begin{array}{rcl} \mathcal{L}(M_v[i]) & \prec_\oplus & \mathsf{r}_v^i\left(\mathcal{L}(M_v[i])\right) \\ & \prec_\oplus & \lambda_0^c \end{array}$$

which is impossible by definition of λ_0^c. Hence

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

□

For a given configuration c, we consider the following Assertion $P^c_{n,k}$, which is of technical use in the induction proofs, and informally described hereafter.

$$\begin{array}{lll} P^c_{n,k}: & \multicolumn{2}{l}{\exists v \in G, \exists n_v \geq n, v \in \Psi^c_{n_v},} \\ & \multicolumn{2}{l}{\exists u_1 \in \Gamma_G^{-1}(v), \ldots, u_k \in \Gamma_G^{-k}(v),} \\ & \multicolumn{2}{l}{\forall i, 1 \leq i \leq k, \exists n_i \geq n, u_i \in \Psi^c_{n_i},} \\ & \multicolumn{2}{l}{(u_k, u_{k-1}) \in E, \ldots, (u_2, u_1) \in E, (u_1, v) \in E} \\ & M_{u_k}[\delta^-(u_k)+1] & \prec_\oplus \ \ldots \\ & & \prec_\oplus \ M_{u_1}[\delta^-(u_1)+1] \\ & & \prec_\oplus \ \lambda^c_n \end{array}$$

Informally, $P^c_{n,k}$ states that there exists an elementary path of length k of illegitimate distinct vertices in configuration c (and such that the illegitimate values are smaller than λ_n). Indeed, $v \in \Psi^c_{n_v}$ with $n_v \geq n$ means that the smallest legitimate r-augmented incoming variable of v is $\lambda^c_{n_v}$, which is greater or equal to λ^c_n. If the outgoing variable of v is smaller than λ^c_n, then this value has been obtained by another r-augmented incoming variable, smaller than λ^c_n. Hence, this incoming variable cannot be legitimate, and therefore v cannot be legitimate too. Using the same reasoning, all nodes on the path are non-legitimate.

We now wish to prove that in a terminal configuration c_t, all nodes of V are legitimate. The proof is by recurrence on the $\Psi^{c_t}_n$ sets, which define a partition of V at configuration c_t.

This recurrence is segmented into base case (Lemma 5) and induction step (Lemma 8). The first part (base case) is proved by recurrence on k in $P^{c_t}_{0,k}$ where c_t is a terminal configuration (base case: Lemma 3, induction step: Lemma 4 and conclusion: Lemma 5). In the same way, the induction step is proved by recurrence on n in $P^{c_t}_{n,k}$ (base case: Lemma 6, induction step: Lemma 7 and conclusion: Lemma 8).

For each part, the proof is based on the fact that since the network is finite, there can not exist an infinite elementary path of non-legitimate nodes.

Lemma 3 *In any terminal configuration c_t where there exists $v \in \Psi^{c_t}_0$ such that v is non-legitimate, $P^{c_t}_{0,1}$ holds.*

Proof. We explicitly expand $P^{c_t}_{0,1}$ as follows:

$$\begin{array}{ll} P^{c_t}_{0,1}: & \exists v \in G, \exists n_v \geq 0, v \in \Psi^{c_t}_{n_v}, \\ & \exists u_1 \in \Gamma_G^{-1}(v), \\ & \exists n_1 \geq 0, u_1 \in \Psi^{c_t}_{n_1}, \\ & (u_1, v) \in E, \\ & M_{u_1}[\delta^-(u_1)+1] \prec_\oplus \lambda^{c_t}_0 \end{array}$$

Consider a node $v \in \Psi^{c_t}_0$. Following Lemma 2, we have

$$\mathcal{L}(M_v[\delta^-(v)+1]) = M_v[0] = \lambda^{c_t}_0$$

Suppose now that in configuration c_t, 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, Rule $\mathcal{R}_1$ is not enabled, which means that:

$$\begin{array}{rcl} M_v[\delta^-(v)+1] & = & \oplus\left\{\mathsf{r}^i_v\left(M_v[i]\right), 0 \leq i \leq \delta^-(v)\right\} \\ & \prec_\oplus & M_v[0] \\ & & \text{(can not be greater;} \\ & & \text{if equal then legitimate)} \\ & \prec_\oplus & \lambda^{c_t}_0 \end{array}$$

Hence the node v has at least one incoming r-augmented variable smaller than $\lambda_0^{c_t}$:

$$\exists u_1 \in \Gamma_G^{-1}(v), \mathsf{r}_v^{\mathsf{Ind}(u_1,v)}\left(M_v[\mathsf{Ind}(u_1,v)]\right) \prec_\oplus \lambda_0^{c_t}$$

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

$$\begin{array}{lll} \exists u_1 \in \Gamma_G^{-1}(v), & & \\ \quad M_v[\mathsf{Ind}(u_1,v)] & \prec_\oplus & \mathsf{r}_v^{\mathsf{Ind}(u_1,v)}\left(M_v[\mathsf{Ind}(u_1,v)]\right) \\ & \prec_\oplus & \lambda_0^{c_t} \end{array}$$

Again, since the considered configuration c_t is terminal, all the incoming variables of the node v have correctly been copied (Rule $\mathcal{R}_2$ is not enabled) from the direct ancestors of v. Formally, we have:

$$\begin{array}{lll} \exists u_1 \in \Gamma_G^{-1}(v), & & \\ \quad M_{u_1}[\delta^-(u_1)+1] & = & M_v[\mathsf{Ind}(u_1,v)] \\ & \prec_\oplus & \mathsf{r}_v^{\mathsf{Ind}(u_1,v)}(M_v[\mathsf{Ind}(u_1,v)]) \\ & \prec_\oplus & \lambda_0^{c_t} \end{array}$$

Note that (u_1, v) is a path of length one. Since

$$M_{u_1}[\delta^-(u_1)+1] \prec_\oplus \lambda_0^{c_t}$$

the node u_1 is non-legitimate (Lemma 2). We proved that, if there exists a terminal configuration c_t in which node v is non-legitimate, then Assertion $P_{0,1}^{c_t}$ holds, which means that there exists a path of length one of non-legitimate distinct nodes. □

Lemma 4 *In any terminal configuration c_t where there exists $v \in \Psi_0^{c_t}$ such that v is non-legitimate, $P_{0,k}^{c_t}$ holds, for any $k \in \mathbb{N} \setminus \{0\}$.*

Proof. We prove by recurrence that, if there exists a terminal configuration c_t where node $v \in \Psi_0^{c_t}$ is non-legitimate, then, for any $k \in \mathbb{N} \setminus \{0\}$, $P_{0,k}^{c_t}$ holds.

Base case. The base case ($k = 1$) is proved by Lemma 3.

Induction step. Suppose that there exists a terminal configuration c_t and an integer $k > 1$ such that $P_{0,k-1}^{c_t}$ holds:

$$\begin{array}{ll} P_{0,k-1}^{c_t}: & \exists v \in G, \exists n_v \geq 0, v \in \Psi_{n_v}^{c_t}, \\ & \exists u_1 \in \Gamma_G^{-1}(v), \ldots, u_{k-1} \in \Gamma_G^{-(k-1)}(v), \\ & \forall i, 1 \leq i < k, \exists n_i \geq 0, u_i \in \Psi_{n_i}^{c_t}, \\ & (u_{k-1}, u_{k-2}) \in E, \ldots, (u_2, u_1) \in E, (u_1, v) \in E \\ & \quad M_{u_{k-1}}[\delta^-(u_{k-1})+1] \quad \prec_\oplus \quad \ldots \\ & \qquad\qquad\qquad\qquad\qquad \prec_\oplus \quad M_{u_1}[\delta^-(u_1)+1] \\ & \qquad\qquad\qquad\qquad\qquad \prec_\oplus \quad \lambda_0^{c_t} \end{array}$$

Since c_t is terminal, Rule $\mathcal{R}_2$ is not enabled and 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,

$$\mathsf{r}_v^i\left(M_{u_{k-1}}[i]\right), 1 \leq i \leq \delta^-(u_{k-1})$$

However, by definition, there is no initial datum smaller than $\lambda_0^{c_t}$. Then, we have:

$$\begin{array}{rl}
\multicolumn{2}{l}{\exists u_k \in \Gamma_G^{-k}(v), (u_k, u_{k-1}) \in E,} \\
\multicolumn{2}{l}{\mathsf{r}_{u_{k-1}}^{\mathsf{Ind}(u_k,u_{k-1})}\left(M_{u_{k-1}}[\mathsf{Ind}(u_k,u_{k-1})]\right)} \\
= & M_{u_{k-1}}[\delta^-(u_{k-1})+1] \\
\prec_\oplus & \ldots \\
\prec_\oplus & M_{u_1}[\delta^-(u_1)+1] \\
\prec_\oplus & \lambda_0^{c_t}
\end{array}$$

Since c_t is terminal, Rule $\mathcal{R}_1$ is not enabled and each incoming variable of the nodes in $\Gamma_G^{-(k-1)}(v)$ have correctly been copied from the outgoing variables of the nodes in $\Gamma_G^{-k}(v)$. Then, we have:

$$\begin{array}{rcl}
\exists u_k \in \Gamma_G^{-k}(v), & & (u_k, u_{k-1}) \in E, \\
M_{u_k}[\delta^-(u_k)+1] & = & M_{u_{k-1}}[\mathsf{Ind}(u_k,u_{k-1})]
\end{array}$$

The r-mappings are strictly idempotent, so :

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

Which in turn is a rewriting of assertion $P_{0,k}^{c_t}$.

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

Lemma 5 *In any terminal configuration c_t, no node in $\Psi_0^{c_t}$ is non-legitimate.*

Proof. From Lemma 4, in any terminal configuration c_t, $P_{0,k}^{c_t}$ holds for any $k \in \mathbb{N} \setminus \{0\}$. Since $P_{0,k}^{c_t}$ cannot hold for k greater than the number of nodes in the system, no vertex of $\Psi_0^{c_t}$ is non legitimate. □

Lemma 5 established the base case for the recurrence on the $\Psi_n^{c_t}$ sets. We now prove the induction step with Lemmas 6 and 7.

Lemma 6 *If there exists a terminal configuration c_t such that all nodes $u \in \Psi_0^{c_t} \cup \ldots \cup \Psi_{n-1}^{c_t}$ are legitimate and there exists a node v in $\Psi_n^{c_t}$ which is non-legitimate, then $P_{n,1}^{c_t}$ holds for $n \in \mathbb{N}$.*

Proof. We explicitly expand $P_{n,1}^{c_t}$ as follows:

$$\begin{array}{ll}
P_{n,1}^{c_t}: & \exists v \in G, \exists n_v \geq n, v \in \Psi_{n_v}^{c_t}, \\
& \exists u_1 \in \Gamma_G^{-1}(v), \\
& \exists n_1 \geq n, u_1 \in \Psi_{n_1}^{c_t}, \\
& (u_1, v) \in E, \\
& M_{u_1}[\delta^-(u_1)+1] \prec_\oplus \lambda_n^{c_t}
\end{array}$$

Consider a node $v \in \Psi_n^{c_t}$. By definition of $\Psi_n^{c_t}$, there exists an integer q, $0 \leq q \leq \delta^-(v)$, such that $M_v[q]$ is legitimate and $\mathsf{r}_v^q(M_v[q]) = \lambda_n^{c_t}$. 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, Rule $\mathcal{R}_2$ is not enabled, which means that:

$$\begin{array}{rcl} M_v[\delta^-(v)+1] & = & \oplus\left\{\mathsf{r}_v^i\left(M_v[i]\right), 0 \le i \le \delta^-(v)\right\} \\ & \prec_\oplus & \lambda_n^{c_t} \end{array}$$

Suppose that $M_v[\delta^-(v)+1] = M_v[0]$. Then we have

$$M_v[0] \prec_\oplus \lambda_n^{c_t}$$

But since $M_v[0] = \mathcal{L}(M_v[0])$, v would not be in $\Psi_n^{c_t}$. Hence we have

$$\begin{array}{lcl} \exists u_1 \in \Gamma_G^{-1}(v), & & \\ \mathsf{r}_v^{\mathsf{Ind}(u_1,v)}\left(M_v[\mathsf{Ind}(u_1,v)]\right) & = & M_v[\delta^-(v)+1] \\ & \prec_\oplus & \lambda_n^{c_t} \end{array}$$

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

$$\begin{array}{lcl} \exists u_1 \in \Gamma_G^{-1}(v), & & \\ M_v[\mathsf{Ind}(u_1,v)] & \prec_\oplus & \mathsf{r}_v^{\mathsf{Ind}(u_1,v)}\left(M_v[\mathsf{Ind}(u_1,v)]\right) \\ & \prec_\oplus & \lambda_n^{c_t} \end{array}$$

Again, since the considered configuration c_t is terminal, all the incoming variables of the node v have correctly been copied from the direct ancestors of v (Rule $\mathcal{R}_1$ is not enabled). Formally, we have:

$$\begin{array}{lcl} \exists u_1 \in \Gamma_G^{-1}(v), & & \\ M_{u_1}[\delta^-(u_1)+1] & = & M_v[\mathsf{Ind}(u_1,v)] \\ & \prec_\oplus & \mathsf{r}_v^{\mathsf{Ind}(u_1,v)}(M_v[\mathsf{Ind}(u_1,v)]) \\ & \prec_\oplus & \lambda_n^{c_t} \end{array}$$

Since $M_v[\mathsf{Ind}(u_1,v)] \prec_\oplus \lambda_n^{c_t}$ and $v \in \Psi_n^{c_t}$, $M_v[\mathsf{Ind}(u_1,v)]$ cannot be legitimate. Thus $M_{u_1}[\delta^-(v)+1]$ is not legitimate and the node u_1 is not legitimate, which means that there exists some $n_1 \ge n$ such that $u_1 \in \Psi_{n_1}^{c_t}$.

We proved that, if there exists a terminal configuration c_t in which node $v \in \Psi_n^{c_t}$ is non-legitimate, then Assertion $P_{n,1}^{c_t}$ holds, which means that there exists a path of length one of non-legitimate distinct nodes. □

Lemma 7 *If there exists a terminal configuration c_t such that all nodes $u \in \Psi_0^{c_t} \cup \ldots \cup \Psi_{n-1}^{c_t}$ are legitimate and there exists a node v in $\Psi_n^{c_t}$ which is non-legitimate, then $P_{n,k}^{c_t}$ holds for $n \in \mathbb{N}$ and $k \in \mathbb{N} \setminus \{0\}$.*

Proof. We will prove by recurrence that, if there exists a terminal configuration c_t where node $v \in \Psi_n^{c_t}$ is non-legitimate, then, for any $k \in \mathbb{N} \setminus \{0\}$, $P_{n,k}^{c_t}$ holds.

Base case. The base case ($k = 1$) is proved by Lemma 6: $P_{n,1}^{c_t}$ holds.

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

$$\begin{array}{ll} P_{n,k-1}^{c_t}: & \exists v \in G, \exists n_v \ge n, v \in \Psi_{n_v}^{c_t}, \\ & \exists u_1 \in \Gamma_G^{-1}(v), \ldots, u_{k-1} \in \Gamma_G^{-(k-1)}(v), \\ & \forall i, 1 \le i < k, \exists n_i \ge n, u_i \in \Psi_{n_i}^{c_t}, \\ & (u_{k-1}, u_{k-2}) \in E, \ldots, (u_2, u_1) \in E, (u_1, v) \in E \\ & \quad M_{u_{k-1}}[\delta^-(u_{k-1})+1] \quad \prec_\oplus \quad \ldots \\ & \qquad\qquad\qquad\qquad\quad \prec_\oplus \quad M_{u_1}[\delta^-(u_1)+1] \\ & \qquad\qquad\qquad\qquad\quad \prec_\oplus \quad \lambda_n^{c_t} \end{array}$$

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 r-augmented incoming variable

$$\mathsf{r}_v^i\left(M_{u_{k-1}}[i]\right), 1 \leq i \leq \delta^-(u_{k-1})$$

No r-augmented legitimate variable is smaller than $\lambda_n^{c_t}$ on u_{k-1}, else u_{k-1} would be in $\Psi_{n'}^{c_t}$ with $n' < n$. In particular, $M_{u_{k-1}}[0] \not\prec_\oplus \lambda_n^{c_t}$. Thus we have:

$$\begin{array}{l} \exists u_k \in \Gamma_G^{-k}(v), (u_k, u_{k-1}) \in E, \\ \mathsf{r}_{u_{k-1}}^{\mathsf{Ind}(u_k,u_{k-1})}\left(M_{u_{k-1}}[\mathsf{Ind}(u_k,u_{k-1})]\right) \\ \qquad\qquad \begin{array}{cl} = & M_{u_{k-1}}[\delta^-(u_{k-1})+1] \\ \prec_\oplus & \ldots \\ \prec_\oplus & M_{u_1}[\delta^-(u_1)+1] \\ \prec_\oplus & \lambda_n^{c_t} \end{array} \end{array}$$

Since c_t is terminal, Rule $\mathcal{R}_1$ is not enabled and each incoming variable of the nodes in $\Gamma_G^{-(k-1)}(v)$ have correctly been copied from the outgoing variables of the nodes in $\Gamma_G^{-k}(v)$. Then, we have:

$$\begin{array}{lcl} \exists u_k \in \Gamma_G^{-k}(v), & & (u_k, u_{k-1}) \in E, \\ M_{u_k}[\delta^-(u_k)+1] & = & M_{u_{k-1}}[\mathsf{Ind}(u_k,u_{k-1})] \end{array}$$

The r-mappings are strictly idempotent, so :

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

Finally, $M_{u_{k-1}}[\mathsf{Ind}(u_k,u_{k-1})]$ is not legitimate, else u_{k-1} would not be in $\Psi_n^{c_t}$.Thus $M_{u_k}[\delta^-(v)+1]$ is not legitimate and node u_k is not legitimate, which means that $u_k \in \Psi_{n'}^{c_t}$ with $n' \geq n$. That gives $P_{n,k}^{c_t}$.

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

Lemma 8 *If sets $\Psi_0^{c_t}, \ldots \Psi_{n-1}^{c_t}$ are legitimate, then $\Psi_n^{c_t}$ is legitimate.*

Proof. In Lemma 7, we proved that if there exists a terminal configuration c_t such that all nodes $u \in \Psi_0^{c_t} \cup \ldots \cup \Psi_{n-1}^{c_t}$ are legitimate and there exists a node v in $\Psi_n^{c_t}$ which is non-legitimate, then $P_{n,k}^{c_t}$ holds for $k \in \mathbb{N} \setminus \{0\}$ and $n \in \mathbb{N}$. Since the network is finite, $P_{n,k}^{c_t}$ may not hold for k greater than the number of nodes in the system, and then all nodes of $\Psi_n^{c_t}$ are legitimate. □

Proposition 5 (Correctness) *Consider a distributed system $\mathcal{S}$ that satisfies Hypotheses 1 and 2, and where each processor P_v runs the local parametric Algorithms $\mathcal{PA}|_{\lhd_v}$ instantiated with r-operator $\lhd_v$. If all r-operators are strictly idempotent, then in any terminal configuration c_t of any execution, all nodes in $\mathcal{S}$ are legitimate.*

Proof. We show that any node in $\mathcal{S}$ is legitimate in configuration c_t by recurrence on the $\Psi_n^{c_t}$ sets, which define a partition of the vertices of $\mathcal{S}$.

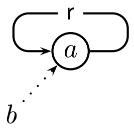


Figure 4: Necessary Condition

Base Case. The fact that $\Psi_0^{c_t}$ is legitimate is proven by Lemma 5.

Induction Step. If sets $\Psi_0^{c_t}, \ldots, \Psi_{n-1}^{c_t}$ are legitimate, then $\Psi_n^{c_t}$ is legitimate, was proven in Lemma 8.

Conclusion. All nodes in $\mathcal{S}$ are legitimate in configuration c_t. □

Then from Proposition 4 and 5, we can state the following sufficient condition:

Proposition 6 (Sufficient condition) *Consider a distributed system $\mathcal{S}$ that satisfies Hypotheses 1 and 2, and where each processor P_v runs the local parametric Algorithms $\mathcal{PA}|_{\lhd_v}$ instantiated with an r-operator $\lhd_v$. If the r-operators $\lhd_v$ are all strictly idempotent, then the distributed protocol is self-stabilizing.*

Proof. We proved that each execution e of our distributed protocol terminates in configuration c_t, where no rule is applicable (Proposition 4, Silent). Moreover, we proved that each such c_t configuration is legitimate (Proposition 5, Correctness). Hence, the set of terminal configurations is legitimate and a closed attractor for our distributed protocol, which means that Algorithm $\mathcal{PA}$ instantiated with r-operators that are strictly idempotent leads to self-stabilizing distributed protocols. □

5.4 Strict Idempotency is Necessary for Self-stabilization

In this section, we show that strict idempotency of the r-operators is a necessary condition to insure self-stabilization of the distributed algorithm based on local parametric Algorithm $\mathcal{PA}$ instantiated by r-operators.

Proposition 7 (Necessary condition) *Consider a distributed system $\mathcal{S}$ that satisfies Hypotheses 1 and 2, and where each processor P_v runs the local Algorithm $\mathcal{PA}|_{\lhd_v}$ instantiated with an r-operator $\lhd_v$. If some r-operators $\lhd_v$ are not strictly idempotent, the distributed algorithm is not self-stabilizing.*

Proof. To prove that the distributed algorithm is not self-stabilizing, it is sufficient to exhibit a particular system and a particular initialization from which the distributed system does not stabilize.

Consider the distributed system composed of a single processor P_v and a single link, which is a loop from P_v to P_v (see Figure 4). Assume that P_v uses the non-strictly idempotent binary r-operator $\lhd$ defined by $x \lhd y = x \oplus \mathsf{r}(y)$ for all x and y in $\mathbb{S}$, where $\oplus$ is an s-operator on $\mathbb{S}$ and r is an homomorphism of $(\mathbb{S}, \oplus)$ (see Section 4). Let a be an element of $\mathbb{S} \setminus \{e_\oplus\}$ satisfying $a = \mathsf{r}(a)$, that contradicts the strict idempotency hypothesis. Now consider the case where the processor P_v is initialized with a datum b strictly greater (in sense of $\preceq_\oplus$, the total order relation induced by $\oplus$) than a: $a \preceq_\oplus M_v[0] = b$. We suppose moreover that $M_v[1] = e_\oplus$ at the beginning. Without any subsequent transient fault, $M_v[1]$ and $M_v[2]$ stabilize on the value b.

Now suppose that, after a transient failure, $M_v[2]$ contains an erroneous datum a. This datum will be copied into $M_v[1]$. The execution will then terminate on the erroneous result a (because $\mathsf{r}(a) = a$ and $b \oplus a = a$). Hence, the $\mathcal{PA}|_{\lhd}$ local algorithm instantiated with the non strictly idempotent r-operator $\lhd$ does not lead to a self-stabilizing distributed algorithm. □

From Propositions 6 and 7, we deduce the following result.

Theorem 1 (Necessary and sufficient condition) *Consider a distributed system $\mathcal{S}$ that satisfies Hypotheses 1 and 2, and where each processor P_v runs the local parametric Algorithms $\mathcal{PA}|_{\triangleleft_v}$ instantiated with an r-operator $\triangleleft_v$. Then the distributed algorithm is self-stabilizing if and only if the r-operators $\triangleleft_v$ are all strictly idempotent.*

As a result, in Sections 5.5 and 6, we only consider strictly idempotent r-operators.

5.5 Complexity

In this section, we investigate the memory space and time needed for the system to stabilize into a legitimate configuration.

In order to give an upper bound on the space and time requirements, we assume that the set $\mathbb{S}$ is finite. (Note that this assumption is used for complexity results only, since our algorithm was proved to be correct even in the case where $\mathbb{S}$ is infinite and Hypothesis 2 holds.)

The space complexity result is immediately given by the assumptions made when writing the local parametric Algorithm $\mathcal{PA}$.

Proposition 8 (Space complexity) *For executing local parametric Algorithm $\mathcal{PA}$, each processor $v \in \mathcal{S}$ needs*

$$\big(\delta^-(v) + 1\big) \times \log_2(|\mathbb{S}|)$$

bits, where $|\mathbb{S}|$ denotes the number of elements in the input data set.

Proof. Each processor v has $\delta^-(v)$ local variables that hold the value of the register of the corresponding direct ancestor, and one register used to communicate with its direct descendants. Each of these local variables or register may hold a value in a finite set $\mathbb{S}$, then need $\log_2(|\mathbb{S}|)$ bits. Note that the constant stored in ROM is not taken into account in this result. □

In the convergence part of the proof, we only assumed that the executions were maximal. In order to provide an upper bound on the stabilization time for our algorithm, we assume strong synchrony between nodes. (Note that this assumption is used for complexity results only, since our algorithm was proved to be correct even in the case of asynchronous and unfair executions.)

Proposition 9 (Time complexity) *Assuming a synchronous system $\mathcal{S}$, the stabilization time of the distributed protocol based on local parametric Algorithm $\mathcal{PA}$ is $O(D + |\mathbb{S}|)$, where D is the diameter of the network.*

Proof. We define ϕ as the function that returns the index of a given element of $\mathbb{S}$. This index always exists since $\mathbb{S}$ is ordered by a total order relation. The signature of ϕ is as follows:

$$\begin{array}{rccc} \phi: & \mathbb{S} & \to & \mathbb{N} \\ & s & \mapsto & \phi(s) \end{array}$$

and we have:

$$s_1 \prec_{\oplus} s_2 \Rightarrow \phi(s_1) < \phi(s_2)$$

After $O(D)$ steps, every node in the network has received values from all of their ancestors. If those values were badly initialized, then the received values are also possibly badly valued.

For each node u, we consider the difference between the index of its final value (since the algorithm terminates) and the index of the smallest received value which is badly initialized. The biggest possible difference is $M - m$, where M is the maximum index value of $\mathbb{S}$ and m the minimum index value of $\mathbb{S}$. This difference is called d and is $O(|\mathbb{S}|)$.

Let l be the length of the smallest r-path-mapping. Any r-path mapping increases a value index by at least l.

In the worst case, there exists a node that has an incorrect input value indexed with m, a correct input value indexed with M, so it has to wait that the incorrect value index is increased by $M - m$ before the incorrect value effect is canceled. Each l times units at least, this incorrect value index is increased by l. Again, in the worst case, if $\lfloor \frac{d}{l} \rfloor < \frac{d}{l}$, another incorrect value may still be lower than the correct value, and the greatest cycle may be followed, inducing an extra d time delay. Overall, after the first $O(D)$ times units, $\left(\lfloor \frac{d}{l} \rfloor \times l\right) + d = O(d)$ time units are needed. □

6 Applications

In this section, we briefly give some examples of r-operators designed to solve particular problems. Those examples use different techniques to illustrate the expressiveness of r-operators in distributed systems.

In the first three applications given in this section, ROM are 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 behavior. On current networks, it is common to have unique non variable identifiers for all nodes (through the hardware network device) so those assumptions do not clash with self-stabilization assumptions.

6.1 Distance Computation, Shortest Dipath Spanning Tree and Forest

Computing the distance from some patriarch u of the network requires u-sourced paths length computations. Such a computation can be done using incrementation ($x \mapsto x + 1$) on the edges. Moreover, if two different paths from u arrive at node v by two different incoming edges, then v should choose the paths with the smallest length. Hence, v performs a minimum computation. Intuitively, an operator such as

$$(x, y) \mapsto \mathsf{min}(x, y + 1)$$

should solve the problem. We now prove that this operator is correct.

Suppose that registers in the distributed system are k bits wide, and that 2^k is larger than the distance from u to any other node in the network (if not, the problem can not be solved). Let $\mathbb{S}$ be the *finite* set $\{0, \ldots, 2^k - 1\}$ (Hypothesis 2 is satisfied). The operator min is an s-operator on $\mathbb{S}$ which defines a *total* order relation usually denoted by $<$ (Hypothesis 1 is satisfied). Its identity element is $2^k - 1$. Let r be a mapping from $\mathbb{S}$ to $\mathbb{S}$ defined by $\mathsf{r}(x) = x + 1$ for $0 \leq x < 2^k - 1$ and $\mathsf{r}(2^k - 1) = 2^k - 1$. This mapping is an homomorphism of $(\mathbb{S}, \mathsf{min})$. Let $\mathsf{minc}(x, y)$ be the binary r-operator defined on $\mathbb{S}$ by $\mathsf{minc}(x, y) = \mathsf{min}(x, \mathsf{r}(y))$ (Definition 4). Since $x < \mathsf{r}(x) = x + 1$ for any $x \in \mathbb{S} \setminus \{2^k - 1\}$, minc is a strictly idempotent r-operator on $\mathbb{S}$ (Definition 5).

Let $\mathcal{S}$ be a distributed system such that there exists a patriarch u with its constant $M_u[0] = 0$ and all others nodes $v \neq u$ with their constants $M_v[0] = 2^k - 1$. Moreover, we assume that each processor P_v of $\mathcal{S}$ runs $\mathcal{PA}|_{\mathsf{minc}}$.

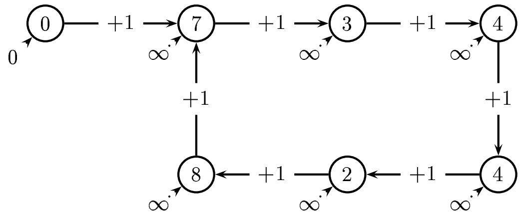
According to Theorem 6, this distributed system $\mathcal{S}$ is self-stabilizing. When the system is stabilized, each node w has the following legitimate value (see Definition 8):

$$M[\delta^-(w) + 1] = \mathsf{min} \left\{ \begin{array}{l} \mathsf{r}_{P_{v \to w}}(M_v[0]), v \in \Gamma_G^-(w), \\ P_{v \to w} \text{ elementary path} \end{array} \right\}$$

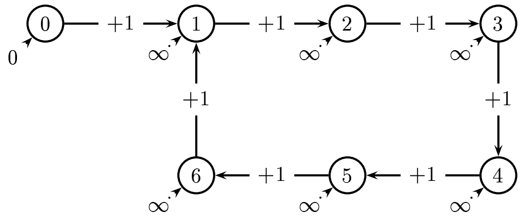
This is equivalent to $M[\delta^-(w) + 1] = \mathsf{r}^{d_G(u,w)}(M_u[0]) = d_G(u, w)$. Hence, the r-operator minc solves the distance computation in any network from a patriarch u in a self-stabilizing way.

After stabilization of system $\mathcal{S}$, a shortest path spanning tree rooted at the patriarch is maintained by the knowledge, on each node, of one incoming variable containing the smallest datum of all those in the incoming variables. Figure 5 shows two possible configurations for $\mathcal{S}$: the first is an incorrect configuration due to a transient failure, while the second is a legitimate configuration obtained after stabilization of the algorithm.

If there are several patriarchs in the system, each node will be in the tree that has the closest patriarch from itself. This completes our solution of the shortest path spanning forest problem.



(a) Faulty Configuration



(b) Terminal Configuration

Figure 5: Distance Computation

6.2 Single and Multiple Source Shortest Path

The single source shortest path problem is similar to the distance computation, except that the edge weights are not necessarily equal to 1. Assume that ω_v^i is the weight of the i^{th} incoming edge of node v. As in Section 6.1, we consider the same *finite* set $\mathbb{S}$, assuming that 2^k is larger than the maximal weighted distance from the patriarch u to any other node in the network (if not, the problem can not be solved). The operator $\oplus$ defined by $x \oplus y = \mathsf{min}(x, y)$ is an s-operator on $\mathbb{S}$ that defines a *total order* relation on $\mathbb{S}$. Hence Hypotheses 1 and 2 are satisfied. For each weight ω_v^i, the mapping r_v^i defined by $x \mapsto \mathsf{min}(2^k - 1, x + \omega_v^i)$ is an homomorphism of $(\mathbb{S}, \oplus)$: $\mathsf{r}(x \oplus y) = \mathsf{min}(2^k - 1, \mathsf{min}(x, y) + \omega_v^i) = \mathsf{min}\left(\mathsf{min}(2^k - 1, x + \omega_v^i), \mathsf{min}(2^k - 1, y + \omega_v^i)\right) = \mathsf{r}_v^i(x) \oplus \mathsf{r}_v^i(y)$.

We then define the r-operator minc_v as follows:

$$\begin{aligned}&\mathsf{minc}_v\left(x_0, \ldots, x_{\delta^-(v)}\right)\\ &\qquad = \mathsf{min}\left(\begin{array}{c} x_0, \mathsf{min}(2^{k-1}, x_1 + \omega_v^1), \\ \ldots, \mathsf{min}(2^{k-1}, x_{n-1} + \omega_v^{\delta^-(v)}) \end{array}\right)\end{aligned}$$

Then we have:

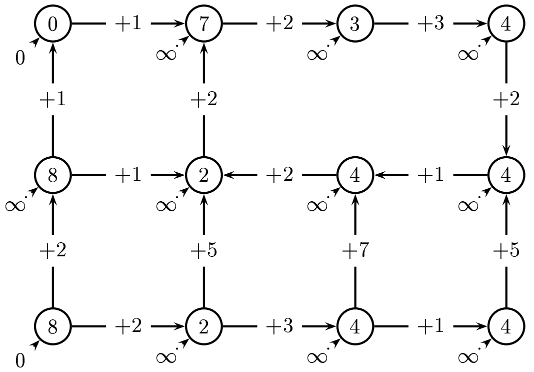
$$\begin{aligned}&\mathsf{minc}_v\left(x_0, \ldots, x_{\delta^-(v)}\right)\\ &\qquad = \mathsf{min}\left(\begin{array}{c} 2^{k-1}, x_0, x_1 + \omega_v^1, \\ \ldots, x_{n-1} + \omega_v^{\delta^-(v)} \end{array}\right)\end{aligned}$$

If each ω_v^i is strictly larger than 0, then each r-mapping $\mathsf{r}_v^i(x) = \mathsf{min}(2^k - 1, x + \omega_v^i)$ verifies $x \prec_\oplus \mathsf{r}_v^i(x)$ for all $x \in \mathbb{S} \setminus \{2^k - 1\}$. The minc_v r-operator is thus strictly idempotent (Definition 7).

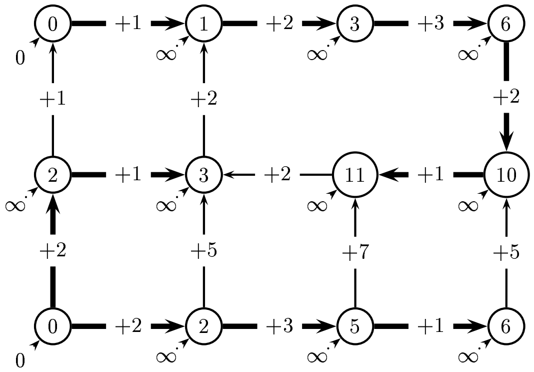
As in previous application, suppose that the patriarch u has its constant $M_u[0] = 0$ and others nodes $v \neq u$ have their constant $M_v[0] = 2^k - 1$. Depending on the implementations, such a setting should not require more than one bit per node. Moreover, assume that each processor P_v runs the $\mathcal{PA}|_{\mathsf{minc}_v}$ local algorithm.

According to Theorem 6, this distributed system $\mathcal{S}$ is self-stabilizing. When $\mathcal{S}$ is stabilized, each node w has the following legitimate value (see Definition 8):

$$M[\delta^-(w) + 1] = \mathsf{min}\left\{\begin{array}{l} \mathsf{r}_{P_{v \to w}}(M_v[0]), v \in \Gamma_G^-(w), \\ P_{v \to w} \text{ elementary path} \end{array}\right\}$$



(a) Faulty Configuration



(b) Terminal Configuration

Figure 6: Multiple Source Shortest Path Computation

This is equivalent to $M[\delta^{-}(w)+1] = \mathsf{r}_{P_{u\to w}}(M_u[0])$, where $\mathsf{r}_{P_{u\to w}}(M_u[0])$ is the smallest weighted distance from u to w. Hence, the r-operator minc_v solves the weighted distance computation in any network from a patriarch u in a self-stabilizing way.

Figure 6 shows two possible configurations of such a distributed system $\mathcal{S}$.

After stabilization of $\mathcal{S}$, a "lightest" path spanning tree rooted at u is maintained provided that each node chooses one of the incoming variables that lead to the smallest path weight. That solves the single source shortest path problem. As in Section 6.1, the same operator solves the problem with multiple patriarchs.

6.3 Best Reliable Path from some Transmitters

In hazardous telecommunication networks where nodes must choose their "best" transmitter, distance is not always the relevant criterium. When the failure rate of neighbor connections is computable and keeps more or less constant, it is interesting to know the transmitter from which the failure rate path is the lowest, and to know the path itself.

We suppose that all registers are k-bits wide and that reliability rates of the links are sampled from 0 (link out of order) to 2^k-1 (no failure at all). The best reliable path is computed from node to node: for all its ancestors u, each node v computes a reception rate τ by multiplying the reception rate τ_u of u by the edge rate $\tau_{(u,v)}$ of the edge (u,v): $\tau = \lfloor \tau_u \times \tau_{(u,v)}/255 \rfloor$. Node v can then compute its best reception rate τ_v by a maximum computation, and determine its best incoming edge. Hence, to know the best reliable path from a transmitter, each node has to perform multiplications and maximum computations. We now build an r-operator that performs these computations, and solves the problem in a self-stabilizing way.

Let $\mathbb{S}$ be the *finite* set $\{0, \ldots, 2^k - 1\}$. The maximum operator max is an s-operator on $\mathbb{S}$ that defines a *total* order relation $\preceq_{\mathsf{max}}$ which is, in fact, the usual order $\geq$ on the integers. We denote by τ_v^i the reliability rate of the i^{th} incoming edge of the node v. We suppose that there is no edge with a reliability rate equal to $2^k - 1$ (on each link, there is sometimes some failures). Hence $\tau_v^i \in \mathbb{S} \setminus \{2^k - 1\}$. Hypotheses 1 and 2 are then satisfied. For each rate τ_v^i, the mapping r_v^i defined by $\mathsf{r}_v^i(x) \mapsto \lfloor x \times \tau_v^i/(2^k - 1) \rfloor$ is an homomorphism of $(\mathbb{S}, \mathsf{max})$. Indeed,

$$\begin{array}{rcl} \mathsf{r}_v^i(\mathsf{max}(x,y)) & = & \lfloor \mathsf{max}(x,y) \times \frac{\tau_v^i}{(2^k-1)} \rfloor \\ & = & \lfloor \mathsf{max}(x \times \frac{\tau_v^i}{(2^k-1)}, y \times \frac{\tau_v^i}{(2^k-1)}) \rfloor \\ & = & \mathsf{max}(\lfloor x \times \frac{\tau_v^i}{(2^k-1)} \rfloor, \lfloor y \times \frac{\tau_v^i}{(2^k-1)} \rfloor) \\ & = & \mathsf{max}(\mathsf{r}_v^i(x), \mathsf{r}_v^i(y)) \end{array}$$

We then define the r-operator maxmul_v on $\mathbb{S}$ by:

$$\begin{array}{l} \mathsf{maxmul}_v(x_0, \ldots, x_{n-1}) \\ \quad = \mathsf{max} \left(\begin{array}{l} x_0, \lfloor x_1 \times \frac{\tau_v^1}{(2^k-1)} \rfloor, \\ \ldots, \lfloor x_{n-1} \times \frac{\tau_v^{n-1}}{(2^k-1)} \rfloor \end{array} \right) \end{array}$$

Since we assume that $0 \leq \tau_v^i < 2^k - 1$, we have $x \prec_{\mathsf{max}} \mathsf{r}_v^i(x)$ for all r-mappings r_v^i (which means that $x > \mathsf{r}_v^i(x)$). Thus the r-operator maxmul_v is strictly idempotent (Definition 7).

Suppose that for any transmitter u, $M_u[0] = 2^k - 1$ and that for all other nodes v, $M_v[0] = 0$. Moreover, assume that each processor P_v runs the $\mathcal{PA}|_{\mathsf{maxmul}_v}$ local algorithm. According to Theorem 6, such a distributed system $\mathcal{S}$ is self-stabilizing. When $\mathcal{S}$ is stabilized, each node w owns the following legitimate value (see Definition 8):

$$M[\delta^-(w) + 1] = \mathsf{max} \left\{ \begin{array}{l} \mathsf{r}_{P_{v \to w}}(M_v[0]), v \in \Gamma_G^-(w), \\ P_{v \to w} \text{ elementary path} \end{array} \right\}$$

which is equivalent to $M[\delta^-(w) + 1] = \mathsf{r}_{P_{u \to w}}(M_u[0])$, where $\mathsf{r}_{P_{u \to w}}(M_u[0])$ is the best reliability rate of a path P from one ancestor transmitter u of w to the node w. Hence, the r-operator maxmul_v allows to solve the best reliability path from a transmitter in any network in a self-stabilizing way. Each node should keep the information corresponding to the greatest reliability rate.

Figure 7 shows two possible configurations of such a system $\mathcal{S}$ where the reliability rates of the links are sampled from 0 (link out of order) to 255 (no failure at all), *i.e.* where registers are 8-bits wide.

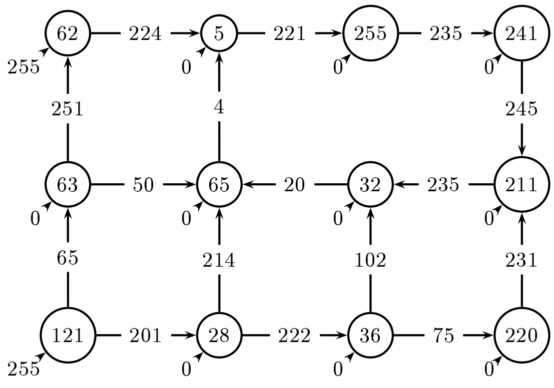
Note that our approach makes the algorithm input adaptive: if reliability rates are adjusted online during the distributed application, then the forest of the best reliable paths is updated without the need of human intervention.

6.4 Depth-first Search Tree

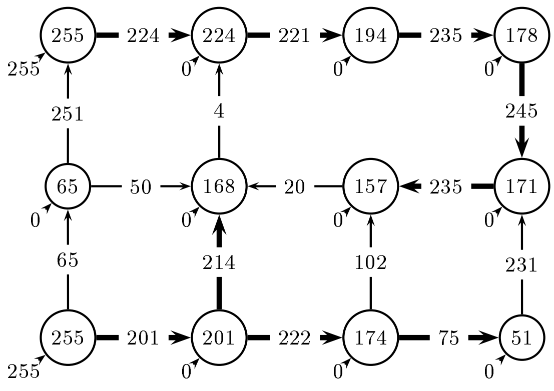
In this section, we give an r-operator that allows to find a depth-first-search tree rooted on a patriarch u in the network, despite transient failures.

To solve this fundamental problem, we assume that each processor holds a constant unique identifier, and that a total order can be defined on those identifiers. Consider the ordered lists of node's identifiers; $\emptyset$ denotes the empty list. We denote by min the binary operator that returns the smallest list of its two operands using the lexicographical order. For instance, if the identifiers are letters $a, b, c, \ldots$ ordered by the alphabetical order, the operator min gives the following results: $\mathsf{min}\left((a, b, d, e,), (a, b, c, d, e, f)\right) = (a, b, c, d, e, f)$, and $\mathsf{min}\left((a, a), (a)\right) = (a)$.

Let L be the greatest list (in sense of min) that can be coded in the k-bits wide registers of the distributed system $\mathcal{S}$. We then consider the set $\mathbb{S}$ composed of the lists of identifiers l verifying $\mathsf{min}(l, L) = l$. Hence $\mathbb{S}$ is the *finite* set $\{\emptyset, \ldots, L\}$ (Hypothesis 2). The operator $\oplus$ defined by $l_1 \oplus l_2 = \mathsf{min}(L, l_1, l_2)$ is an s-operator on $\mathbb{S}$ that defines a *total* order relation (Hypothesis 1). Its identity element is L. We assume that there exists no



(a) Faulty Configuration



(b) Terminal Configuration

Figure 7: Best Reliable Path from Some Transmitters Construction

elementary path P in the network such that the list composed of the identifiers of the nodes of P is greater than L (which means that the registers are wide enough to store all necessary lists).

For each node v, consider the r-mappings r_v defined on $\mathbb{S}$ by $\mathsf{r}_v(l) = \min(L, l \cup (v))$, where v is the identifier of the node and $\cup$ denotes the concatenation operator of two lists. Such mappings are homomorphisms of $(\mathbb{S}, \oplus)$:

$$\begin{aligned}\mathsf{r}_v(l_1 \oplus l_2) &= \min(L, \min(L, l_1, l_2) \cup (v)) \\ &= \min(L, \min(L, l_1 \cup (v)), \min(L, l_2 \cup (v))) \\ &= \mathsf{r}_v(l_1) \oplus \mathsf{r}_v(l_2)\end{aligned}$$

We then define the binary r-operator $\mathsf{lexicat}_v$ on $\mathbb{S}$ by $\mathsf{lexicat}_v(l_1, l_2) = l_1 \oplus \mathsf{r}_v(l_2)$ (Definition 4). For any list $l \in \mathbb{S} \setminus \{L\}$, $l \prec_\oplus \mathsf{r}_v(l)$. The r-operator $\mathsf{lexicat}_v$ is then strictly-idempotent (Definition 5).

Suppose that the patriarch u (root of the depth-first-search tree) has $M_u[0] = \emptyset$ while other nodes $v \neq u$ are such that $M_v[0] = L$. Moreover, we assume that each processor P_v runs the $\mathcal{PA}|_{\mathsf{lexicat}_v}$ local parametric algorithm.

According to Theorem 6, any induced distributed system is self-stabilizing. When the system is stabilized, each node w owns the following legitimate value (r_P concatenates the list of identifiers of the nodes of P to any list sent along P):

$$M[\delta^-(w) + 1] = \min \left\{ \begin{array}{l} \mathsf{r}_{P_{v \to w}}(M_v[0]), v \in \Gamma_G^-(w), \\ P_{v \to w} \text{ elementary path} \end{array} \right\}$$

which is equivalent to $M[\delta^-(w) + 1] = \mathsf{r}_{P_{u \to w}}(M_u[0])$, where $\mathsf{r}_{P_{u \to w}}(M_u[0])$ is the smallest (in sense of $\min$) list of identifiers of all the paths from the patriarch u to the node w. We now prove that this result induces a depth-first-search tree rooted at u.

First, all legitimate values are lists beginning by u, the identifier of the root. Moreover, all nodes $v \neq u$ have in their incoming data $M_v[1]..M_v[\delta^-(v)]$ a smaller list than their own legitimate value $M_v[\delta^-(v) + 1]$. This smallest incoming list is the legitimate value of one ancestor. Hence, if each node chooses one of its incoming edge, corresponding to the smallest received value, we obtain a tree, rooted at u, that we will denotes T_u.

Now, to prove that this tree is a depth-first-search tree, it is sufficient to verify that the numbering of each node given by the legitimate values is a depth-first-search numbering of the network $G(V, E)$:

$$\forall v, w \in V, \quad w \in \Gamma_G^+(v)$$
$$\Rightarrow \left\{ \begin{array}{l} \left(\begin{array}{ll} & w \in \Gamma_{T_u}^+(v) \\ \text{and} & M_v[\delta^-(v) + 1] \prec_\oplus M_w[\delta^-(w) + 1] \end{array} \right) \\ \quad \textit{or} \\ \left(\begin{array}{ll} & w \notin \Gamma_{T_u}^+(v) \\ \text{and} & M_w[\delta^-(w) + 1] \prec_\oplus M_v[\delta^-(v) + 1] \end{array} \right) \end{array} \right.$$

The legitimate values are all unique since they denote a unique path. Moreover the numbering increases along the edges of T_u. Hence, if this numbering is not a depth-first-search numbering, there exists an edge (v, w) in G such that $M_v[\delta^-(v) + 1] \prec_\oplus M_w[\delta^-(w) + 1]$ and $w \notin T_v$ where T_v denotes the subtree of T_u rooted at v. In this case, we have $w \notin T_v$ and $v \notin T_w$ (see Figure 8).

Then there exists a node t which is an ancestor of both v and w in T_u and yet different from both v and w. Let v' be the first node in the path from t to v in T_u and let w' (resp. w'') be the first (resp. the last) one in the path from t to w. We have $M_v[\delta^-(v) + 1] = M_t[\delta^-(t) + 1] \cup (v') \cup \cdots \cup (v)$ and $M_w[\delta^-(w) + 1] = M_t[\delta^-(t)+1] \cup (w') \cup \cdots \cup (w'') \cup (w)$. Since $M_v[\delta^-(v)+1] \prec_\oplus M_w[\delta^-(w)+1]$, we have $(v') \prec_\oplus (w')$. Thus the incoming list received by w from its direct ancestor v ($M_t[\delta^-(t)+1] \cup (v') \cup \cdots \cup (v)$) is smallest (in sense of the lexicographical order $\oplus$) than the one received by its direct ancestor w'' ($M_t[\delta^-(t)+1] \cup (w') \cup \cdots \cup (w'')$), which contradicts the fact that w has chosen one of its smallest ancestor to construct the depth-first-search tree T_u.

Hence T_u is a depth-first-search tree rooted at the patriarch u; the legitimate values give a depth-first-search numbering, and indicates the list of the nodes from the root to themselves in the tree. They allow also to compute the distance from the root to the node in this depth-first-search tree (length of the list).

The $\mathsf{lexicat}_v$ r-operator allows to solve the depth-first-search problem in any network, in spite of transient failure.

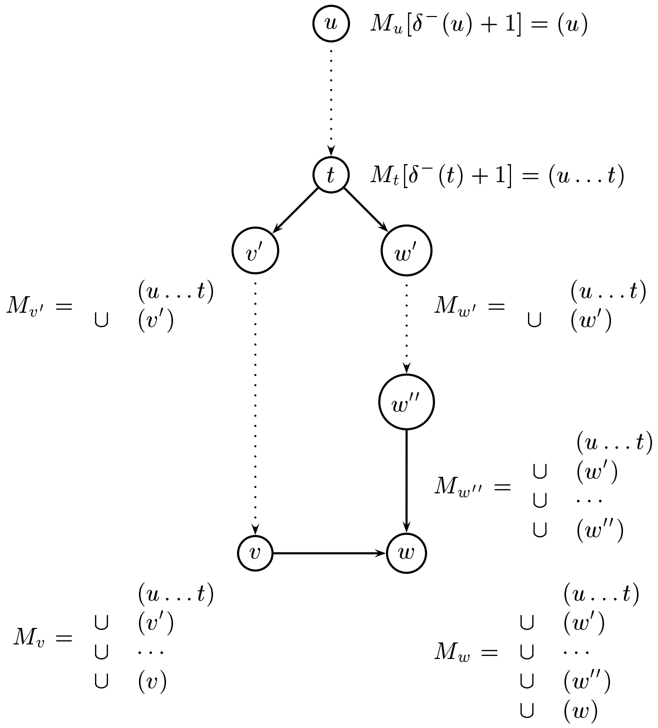


Figure 8: Depth-first-search Tree Construction

7 Conclusion

We characterized a set of silent tasks that can be solved in a self-stabilizing way using a single algorithm parameterized by an 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 $\mathbb{S}$. Moreover, such a condition is local to each node and thus easily checkable. Note that 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 Computation, Shortest Path Calculus, Depth-first Search Tree Construction and Best Reliability Path. A nice property of our approach is that no knowledge on the communication graph is needed: the resulting algorithm does not need any information about the network topology, size, degree or diameter to stabilize. Of course, if such information is available, the stabilization time will be reduced appropriately.

It should be interesting to further develop our approach using the layered scheme of [17]. In [17], the lower layer, that ensures stabilization, is basically the r-operator minc (see Section 6.1) along with some bound on the network diameter, while the higher layer computes a maximum metric tree with an operator that is not strictly idempotent. By considering appropriate domains for local algorithms, this layering of operators would lead to solve problems (*e.g.* maximum flow routing) for which strictly idempotent r-operators are not easily found. Still any such solution would benefit from our weak hypothesis (read-write atomicity, unfair scheduling, no global knowledge about topology required).

References

[1] L. O. Alima, J. Beauquier, A. K. Datta and S. Tixeuil. *Self-stabilization with Global Rooted Synchronizers*. In Proceedings of the Eighteenth International Conference on Distributed Computing Systems (ICDCS'98), pp. 102–109, 1998.

[2] Y. Afek and A. Bremler. *Self-stabilizing Unidirectionnal Network Algorithms by Power Supply*. Chicago Journal of Theoretical Computer Science, Vol. 1998, Art. 3, The MIT Press, December 7, 1998.

[3] 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.

[4] Y. Afek, S. Kutten and M. Yung. *Memory-efficient self-stabilization on general networks*. In Workshop on Distributed Algorithms (WDAG'90), pp. 15-28, 1990.

[5] A. Arora, P. Attie, M. Evangelist and M. G. Gouda. *Convergence of Iteration Systems*. Distributed Computing, Vol. 7, pp. 43–53, 1993.

[6] B. Awerbuch, B. Patt-Shamir and G. Varghese. *Self-stabilization by Local Checking and Correction*. In IEEE Annual Symposium on Foundations of Computer Science, pp 268–277, Los Alamitos, CA, October 1991.

[7] B. Awerbuch, B. Patt-Shamir, G. Varghese and S. Dolev. *Self-stabilization by Local Checking and Global Reset*. In International Workshop on Distributed Algorithms, pp 326–339, Berlin, 1994.

[8] F. Baccelli, G. Cohen, G. Olsder and J.-P. Quadrat. *Synchronization and Linearity, an algebra for discrete event systems*. Series in Probability and Mathematical Statistics. Wiley, Chichester, UK, 1992.

[9] S. K. Das, A. K. Datta, and S. Tixeuil. *Self-StaBiLiZing Algorithms in DAG Structured Networks*. Parallel Processing Letters, Vol. 9(4), pp. 563–574, December 1999.

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

[11] S. Dolev, G. Gouda, and M. Schneider. *Memory Requirements for Silent Stabilization*. Proc. of the Fifteenth Annual ACM Symposium on Principles of Distributed Computing (PODC'96), pp. 27-34, 1996.

[12] S. Dolev and T. Herman. *SuperStaBiLiZing Protocols for Dynamic Distributed Systems*. Chicago Journal of Theoretical Computer Science, Vol 1997, Art. 4, The MIT Press, December 19, 1997.

[13] 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.

[14] B. Ducourthial. *New Operators for Computing with Associative Nets*. Proceedings of the Fifth International Colloquium on Structural Information and Communication Complexity (SIROCCO'98), Amalfi, Italia, 1998.

[15] B. Ducourthial and S. Tixeuil. *Self-stabilizing Global Computations with r-operators*. Proc. of the Second International Conference On Principles Of DIStributed Computing (OPODIS'98), pp. 99–113, Hermes, France, 1998.

[16] M. G. Gouda and M. Schneider. *Maximizable Routing Metrics*. In Proceedings of the Sixth International Conference on Network Protocols (ICNP'98), pp. 71–78, October 1998.

[17] M. G. Gouda and M. Schneider. *Stabilization of Maximal Metric Trees*. In Proceedings of the Fourth Workshop on Self-stabilizing Systems (WSS'99), Austin, Texas, pp. 10–17, 1999.

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

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