# Self-Stabilizing Distributed Cooperative Reset 

Stéphane Devismes, Colette Johnen

## To cite this version:

Stéphane Devismes, Colette Johnen. Self-Stabilizing Distributed Cooperative Reset. [Research Report] Université Grenoble Alpes (France). 2019. hal-01976276v3

HAL Id: hal-01976276
https://hal.science/hal-01976276v3
Submitted on 19 Apr 2019

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# Self-Stabilizing Distributed Cooperative Reset* 

Stéphane Devismes ${ }^{\dagger}$ and Colette Johnen ${ }^{\ddagger}$<br>${ }^{\dagger}$ Université Grenoble Alpes, VERIMAG, UMR 5104, France<br>$\ddagger$ Université de Bordeaux, LaBRI, UMR 5800, France


#### Abstract

Self-stabilization is a versatile fault-tolerance approach that characterizes the ability of a system to eventually resume a correct behavior after any finite number of transient faults. In this paper, we propose a self-stabilizing reset algorithm working in anonymous networks. This algorithm resets the network in a distributed non-centralized manner, i.e., it is multi-initiator, as each process detecting an inconsistency may initiate a reset. It is also cooperative in the sense that it coordinates concurrent reset executions in order to gain efficiency. Our approach is general since our reset algorithm allows to build self-stabilizing solutions for various problems and settings. As a matter of fact, we show that it applies to both static and dynamic specifications since we propose efficient self-stabilizing reset-based algorithms for the (1minimal) $(f, g)$-alliance (a generalization of the dominating set problem) in identified networks and the unison problem in anonymous networks. Notice that these two latter instantiations enhance the state of the art. Indeed, in the former case, our solution is more general than the previous ones; while in the latter case, the time complexity of the proposed unison algorithm is better than that of previous solutions of the literature.


Keywords: Distributed algorithms, self-stabilization, reset, alliance, unison.

## 1 Introduction

In distributed systems, a self-stabilizing algorithm is able to recover a correct behavior in finite time, regardless of the arbitrary initial configuration of the system, and therefore also after a finite number of transient faults, provided that those faults do not alter the code of the processes.

For more than 40 years, a vast literature on self-stabilizing algorithms has been developed. Selfstabilizing solutions have been proposed for many kinds of classical distributed problems, e.g., token circulation [34], spanning tree construction [17], clustering [14], routing [27], propagation of information with feedback [12], clock synchronization [19], etc. Moreover, self-stabilizing algorithms have been designed to handle various environments, e.g., wired networks [34, 17, 14, 27, 12, 19], WSNs [8, 39], peer-to-peer systems [15, 13], etc. Drawing on this experience, general methodologies for making distributed algorithms self-stabilizing have been proposed. In particular, Katz and Perry [36] give a characterization of problems admitting a self-stabilizing solution. Precisely, they describe a general algorithm that transforms almost all algorithms (specifically, those algorithms that can be self-stabilized) into their corresponding stabilizing

[^0]version. However, this so-called transformer is, by essence, inefficient both in terms of space and time complexities: actually, its purpose is only to demonstrate the feasibility of the transformation.

Interestingly, many proposed general methods [36, 7, 4, 5] are based on reset algorithms. Such algorithms are initiated when an inconsistency is discovered in the network, and aim at reinitializing the system to a correct (pre-defined) configuration.

A reset algorithm may be centralized at a leader process (e.g., see [4]), or fully distributed, meaning multi-initiator (as our proposal here). In the former case, either the reset is coupled with a snapshot algorithm (which makes a global checking of the network), or processes detecting an incoherence (using local checking [6]) should request a reset to the leader. In the fully distributed case, resets are locally initiated by processes detecting inconsistencies. This latter approach is considered as more efficient when the concurrent resets are coordinated. In other words, concurrent resets have to be cooperative (in the sense of [37]) to ensure the fast convergence of the system to a consistent global state.

Self-stabilization makes no hypotheses on the nature (e.g., memory corruption or topological changes) or extent of transient faults that could hit the system, and a self-stabilizing system recovers from the effects of those faults in a unified manner. Now, such versatility comes at a price, e.g., after transient faults cease, there is a finite period of time, called the stabilization phase, during which the safety properties of the system are violated. Hence, self-stabilizing algorithms are mainly compared according to their stabilization time, the maximum duration of the stabilization phase.

General schemes and efficiency are usually understood as orthogonal issues. We tackle this problem by proposing an efficient self-stabilizing reset algorithm working in any anonymous connected network. Our algorithm is written in the locally shared memory model with composite atomicity, where executions proceed in atomic steps (in which a subset of enabled processes move, i.e., update their local states) and the asynchrony is captured by the notion of daemon. The most general daemon is the distributed unfair daemon. So, solutions stabilizing under such an assumption are highly desirable, because they work under any other daemon assumption.

The stabilization time is usually evaluated in terms of rounds, which capture the execution time according to the speed of the slowest processes. But, another crucial issue is the number of local state updates, i.e. the number of moves. Indeed, the stabilization time in moves captures the amount of computations an algorithm needs to recover a correct behavior.

The daemon assumption and time complexity are closely related. To obtain practical solutions, the designer usually tries to avoid strong assumptions on the daemon, like for example, assuming all executions are synchronous. Now, when the considered daemon does not enforce any bound on the execution time of processes, the stabilization time in moves can be bounded only if the algorithm works under an unfair daemon. For example, if the daemon is assumed to be distributed and weakly fair (a daemon stronger than the distributed unfair one) and the studied algorithm actually requires the weakly fairness assumption to stabilize, then it is possible to construct executions whose convergence is arbitrarily long in terms of atomic steps (and so in moves), meaning that, in such executions, there are processes whose moves do not make the system progress in the convergence. In other words, these latter processes waste computation power and so energy. Such a situation should be therefore prevented, making the unfair daemon more desirable than the weakly fair one.

There are many self-stabilizing algorithms proven under the distributed unfair daemon, e.g., [2, 21, 31]. However, analyzes of the stabilization time in moves is rather unusual and this may be an important issue. Indeed, recently, several self-stabilizing algorithms which work under a distributed unfair daemon have been shown to have an exponential stabilization time in moves in the worst case, e.g., the silent leader election algorithms from [21] (see [2]) and the Breadth-First Search (BFS) algorithm of Huang and Chen [33]
(see [22]).

### 1.1 Contribution

We propose an efficient self-stabilizing reset algorithm working in any anonymous connected network. Our algorithm is written in the locally shared memory model with composite atomicity, assuming a distributed unfair daemon, i.e., the most general scheduling assumption of the model. It is based on local checking and is fully distributed (i.e., multi-initiator). Concurrent resets are locally initiated by processes detecting inconsistencies, these latter being cooperative to gain efficiency.

As a matter of fact, our algorithm makes an input algorithm recovering a consistent global state within at most $3 n$ rounds, where $n$ is the number of processes. During a recovering, any process executes at most $3 n+3$ moves. Our reset algorithm allows to build efficient self-stabilizing solutions for various problems and settings. In particular, it applies to both static and dynamic specifications. In the static case, the self-stabilizing solution we obtain is also silent [28]: a silent algorithm converges within finite time to a configuration from which the values of the communication registers used by the algorithm remain fixed. Silence is a desirable property. Indeed, as noted in [28], the silent property usually implies more simplicity in the algorithm design. Moreover, a silent algorithm may utilize less communication operations and communication bandwidth.

To show the efficiency of our method, we propose two reset-based self-stabilizing algorithms, respectively solving the unison problem in anonymous networks and the 1-minimal $(f, g)$-alliance in identified networks.

Our unison algorithm has a stabilization time in $O(n)$ rounds and $O\left(\Delta . n^{2}\right)$ moves. Actually, its stabilization times in round matches the one of the previous best existing solution [11]. However, it achieves a better stabilization time in moves, since the algorithm in [11] stabilizes in $O\left(D \cdot n^{3}+\alpha \cdot n^{2}\right)$ moves (as shown in [23]), where $\alpha$ is greater than the length of the longest chordless cycle in the network.

As explained before, our 1-minimal $(f, g)$-alliance algorithm is also silent. Its stabilization time is $O(n)$ rounds and $O(\Delta . n . m)$ moves, where $D$ is the network diameter and $m$ is the number of edges in the network. To the best of our knowledge, until now there was no self-stabilizing algorithm solving that problem without any restriction on $f$ and $g$.

### 1.2 Related Work

Several reset algorithms have been proposed in the literature. In particular, several solutions, e.g., [7, 6], have been proposed in the I/O automata model. In this model, communications are implemented using messagepassing and assuming weakly fairness. Hence, move complexity cannot be evaluated in that model. In these papers, authors additionally assume links with known bounded capacity. In [7], authors introduce the notion of local checking, and propose a method that, given a self-stabilizing global reset algorithm, builds a selfstabilizing solution of any locally checkable problem (i.e., a problem where inconsistency can be locally detected) in an identified network. The stabilization time in rounds of obtained solutions depends on the input reset algorithm. In [6], authors focus on an restrictive class of locally checkable problems, those that are also locally correctable. A problem is locally correctable if the global configuration of the network can be corrected by applying independent corrections on pair neighboring processes. Now, for example, the 1 minimal $(f, g)$ alliance problem is not locally correctable since there are situations in which the correction of a single inconsistency may provoke a global correction in a domino effect reaction. Notice also that processes are not assumed to be identified in [6], however the considered networks are not fully anonymous
either. Indeed, each link has one of its incident processes designated as leader. Notice also that authors show a stabilization time in $O(H)$ when the network is a tree, where $H$ is the tree height.

Self-stabilization by power supply [1] also assumes message-passing with links of known bounded capacity and process identifiers. Using this technique, the stabilization time is in $O(n)$ rounds in general. Now, only static problems, e.g. leader election and spanning tree construction, are considered.

Fully anonymous networks are considered in [5] in message-passing systems with unit-capacity links and assuming weakly fairness. The proposed self-stabilizing reset has a memory requirement in $O\left(\log ^{\star}(n)\right)$ bits per process. But this small complexity comes at the price of a stabilization time in $O(n \log n)$ rounds.

Finally, Arora and Gouda have proposed a mono-initiator reset algorithm in the locally shared memory model with composite atomicity. Their self-stabilizing reset works in identified networks, assuming a distributed weakly fair daemon. The stabilization time of their solution is in $O(n+\Delta . D)$ rounds, where $\Delta$ is the degree of the network.

### 1.3 Roadmap

The remainder of the paper is organized as follows. In the next section, we present the computational model and basic definitions. In Section 3, we present, prove, and analyze the time complexity of our reset algorithm. In the two last sections, we propose two efficient instances of our reset-based method, respectively solving the unison problem in anonymous networks and the 1-minimal $(f, g)$-alliance in identified networks.

## 2 Preliminaries

### 2.1 Network

We consider a distributed system made of $n$ interconnected processes. Information exchanges are assumed to be bidirectional. Henceforth, the communication network is conveniently modeled by a simple undirected connected graph $G=(V, E)$, where $V$ is the set of processes and $E$ a set of $m$ edges $\{u, v\}$ representing the ability of processes $u$ and $v$ to directly exchange information together. We denote by $D$ the diameter of $G$, i.e., the maximum distance between any two pairs of processes. For every edge $\{u, v\}, u$ and $v$ are said to be neighbors. For every process $u$, we denote by $\delta_{u}$ the degree of $u$ in $G$, i.e., the number of its neighbors. Let $\Delta=\max _{u \in V} \delta_{u}$ be the (maximum) degree of $G$.

### 2.2 Computational Model

We use the composite atomicity model of computation [24] in which the processes communicate using a finite number of locally shared registers, simply called variables. Each process can read its own variables and that of its neighbors, but can write only to its own variables. The state of a process is defined by the values of its variables. A configuration of the system is a vector consisting of the states of each process.

Every process $u$ can access the states of its neighbors using a local labeling. Such labeling is called indirect naming in the literature [42]. All labels of $u$ 's neighbors are stored into the set $N(u)$. To simplify the design of our algorithms, we sometime consider the closed neighborhood of a process $u$, i.e., the set including $u$ itself and all its neighbors. Let $N[u]$ be the set of labels local to $u$ designating all members of its closed neighborhood, including $u$ itself. In particular, $N(u) \subsetneq N[u]$. We assume that each process $u$ can identify its local label $\alpha_{u}(v)$ in the sets $N(v)$ of each neighbor $v$ and $N[w]$ of each member $w$ of its closed neighborhood. When it is clear from the context, we use, by an abuse of notation, $u$ to designate both the process $u$ itself, and its local labels (i.e., we simply use $u$ instead of $\alpha_{u}(v)$ for $v \in N[u]$ ).

A distributed algorithm consists of one local program per process. The program of each process consists of a finite set of rules of the form

$$
\langle\text { label }\rangle:\langle\text { guard }\rangle \rightarrow\langle\text { action }\rangle
$$

Labels are only used to identify rules in the reasoning. A guard is a Boolean predicate involving the state of the process and that of its neighbors. The action part of a rule updates the state of the process. A rule can be executed only if its guard evaluates to true; in this case, the rule is said to be enabled. A process is said to be enabled if at least one of its rules is enabled. We denote by Enabled $(\gamma)$ the subset of processes that are enabled in configuration $\gamma$.

When the configuration is $\gamma$ and $\operatorname{Enabled}(\gamma) \neq \emptyset$, a non-empty set $\mathcal{X} \subseteq \operatorname{Enabled}(\gamma)$ is activated by a so-called daemon; then every process of $\mathcal{X}$ atomically executes one of its enabled rules, ${ }^{1}$ leading to a new configuration $\gamma^{\prime}$, and so on. The transition from $\gamma$ to $\gamma^{\prime}$ is called a step. The possible steps induce a binary relation over the set of configurations, denoted by $\mapsto$. An execution is a maximal sequence of configurations $e=\gamma_{0} \gamma_{1} \cdots \gamma_{i} \cdots$ such that $\gamma_{i-1} \mapsto \gamma_{i}$ for all $i>0$. The term "maximal" means that the execution is either infinite, or ends at a terminal configuration in which no rule is enabled at any process.

Each step from a configuration to another is driven by a daemon. We define a daemon as a predicate $\mathfrak{D}$ over executions. A daemon $\mathfrak{D}$ may restrain the set of possible executions (in particular, it may forbid some steps), i.e., only executions satisfying $\mathfrak{D}$ are possible. We assume here the daemon is distributed and unfair. "Distributed" means that while the configuration is not terminal, the daemon should select at least one enabled process, maybe more. "Unfair" means that there is no fairness constraint, i.e., the daemon might never select an enabled process unless it is the only enabled process. In other words, the distributed unfair daemon is defined by the predicate true (i.e. it is the most general daemon), and assuming that daemon, every execution is possible and $\mapsto$ is actually the set of all possible steps.

### 2.3 Self-Stabilization and Silence

Let A be a distributed algorithm. Let $P$ and $P^{\prime}$ be two predicates over configurations of A. Let $C$ and $C^{\prime}$ be two subsets of $\mathcal{C}_{\mathrm{A}}$, the set of A's configurations.

- $P$ (resp. $C$ ) is closed by A if for every step $\gamma \mapsto \gamma^{\prime}$ of A, $P(\gamma) \Rightarrow P\left(\gamma^{\prime}\right)$ (resp. $\gamma \in C \Rightarrow \gamma^{\prime} \in C$ ).
- A converges from $P^{\prime}$ (resp. $C^{\prime}$ ) to $P$ (resp. $C$ ) if each of its executions starting from a configuration satisfying $P^{\prime}$ (resp. in a configuration of $C^{\prime}$ ) contains a configuration satisfying $P$ (resp. a configuration of $C$ ).
- $P$ (resp. $C$ ) is an attractor for A if $P$ (resp. $C$ ) is closed by A and A converges from true (resp. from $\mathcal{C}_{\mathrm{A}}$ ) to $P$ (resp. to $C$ ).

Let $S P$ be a specification, i.e., a predicate over executions. Algorithm A is self-stabilizing for $S P$ (under the unfair daemon) if there exists a non-empty subset of its configurations $\mathcal{L}$, called the legitimate configurations, such that $\mathcal{L}$ is an attractor for A and every execution of a that starts in a configuration of $\mathcal{L}$ satisfies $S P$. Configurations of $\mathcal{C}_{\mathrm{A}} \backslash \mathcal{L}$ are called the illegitimate configurations.

In our model, an algorithm is silent [28] if and only if all its possible executions are finite. Let $S P^{\prime}$ be an predicate over configurations of A. Usually, silent self-stabilization is (equivalently) reformulated as follows. A is silent and self-stabilizing for the $S P^{\prime}$ if all its executions are finite and all its terminal configurations satisfy $S P^{\prime}$. Of course, in silent self-stabilization, the set of legitimate configurations is chosen as the set of terminal configurations.

[^1]
### 2.4 Time Complexity

We measure the time complexity of an algorithm using two notions: rounds [26,18] and moves [24]. We say that a process moves in $\gamma_{i} \mapsto \gamma_{i+1}$ when it executes a rule in $\gamma_{i} \mapsto \gamma_{i+1}$.

The definition of round uses the concept of neutralization: a process $v$ is neutralized during a step $\gamma_{i} \mapsto$ $\gamma_{i+1}$, if $v$ is enabled in $\gamma_{i}$ but not in configuration $\gamma_{i+1}$, and it is not activated in the step $\gamma_{i} \mapsto \gamma_{i+1}$.

Then, the rounds are inductively defined as follows. The first round of an execution $e=\gamma_{0} \gamma_{1} \ldots$ is the minimal prefix $e^{\prime}=\gamma_{0} \cdots \gamma_{j}$, such that every process that is enabled in $\gamma_{0}$ either executes a rule or is neutralized during a step of $e^{\prime}$. Let $e^{\prime \prime}$ be the suffix $\gamma_{j} \gamma_{j+1} \cdots$ of $e$. The second round of $e$ is the first round of $e^{\prime \prime}$, and so on.

The stabilization time of a self-stabilizing algorithm is the maximum time (in moves or rounds) over every possible execution (starting from any initial configuration) to reach a legitimate configuration.

### 2.5 Composition

We denote by $\mathrm{A} \circ \mathrm{B}$ the composition of the two algorithms A and B which is the distributed algorithm where the local program $(\mathrm{A} \circ \mathrm{B})(u)$, for every process $u$, consists of all variables and rules of both $\mathrm{A}(u)$ and $\mathrm{B}(u)$.

## 3 Self-Stabilizing Distributed Reset Algorithm

### 3.1 Overview of the Algorithm

In this section, we present our distributed cooperative reset algorithm, called SDR. The formal code of SDR, for each process $u$, is given in Algorithm 1. This algorithm aims at reinitializing an input algorithm $I$ when necessary. SDR is self-stabilizing in the sense that the composition $I \circ S D R$ is self-stabilizing for the specification of I. Algorithm SDR works in anonymous networks and is actually is multi-initiator: a process $u$ can initiate a reset whenever it locally detects an inconsistency in I, i.e., whenever the predicate $\neg \mathbf{P}$ _ICorrect $(u)$ holds (i.e., I is locally checkable). So, several resets may be executed concurrently. In this case, they are coordinated: a reset may be partial since we try to prevent resets from overlapping.

### 3.2 The Variables

Each process $u$ maintains two variables in Algorithm SDR: $s t_{u} \in\{C, R B, R C\}$, the status of $u$ with respect to the reset, and $d_{u} \in \mathbb{N}$, the distance of $u$ in a reset.

Variable $s t_{u}$. If $u$ is not currently involved into a reset, then it has status $C$, which stands for correct. Otherwise, $u$ has status either $R B$ or $R F$, which respectively mean reset broadcast and reset feedback. Indeed, a reset is based on a (maybe partial) Propagation of Information with Feedback (PIF) where processes reset their local state in I (using the macro reset) during the broadcast phase. When a reset locally terminates at process $u$ (i.e., when $u$ goes back to status $C$ by executing rule_C $(u)$ ), each member $v$ of its closed neighborhood satisfies $\mathbf{P}$ reset $(v)$, meaning that they are in a pre-defined initial state of $I$. At the global termination of a reset, every process $u$ involved into that reset has a state in I which is consistent w.r.t. that of its neighbors, i.e., P_ICorrect $(u)$ holds. Notice that, to ensure that $\mathbf{P}$ ICorrect $(u)$ holds at the end of a reset and for liveness issues, we enforce each process $u$ stops executing I whenever a member of its closed neighborhood (in particular, the process itself) is involved into a reset: whenever $\neg \mathbf{P}$ _Clean $(u)$ holds, $u$ is not allowed to execute I.

Variable $d_{u}$. This variable is meaningless when $u$ is not involved into a reset (i.e., when $u$ has status $C$ ). Otherwise, the distance values are used to arrange processes involved into resets as a Directed Acyclic Graph $(D A G)$. This distributed structure allows to prevent both livelock and deadlock. Any process $u$ initiating a reset (using rule rule_R $(u)$ ) takes distance 0 . Otherwise, when a reset is propagated to $u$ (i.e., when rule $\_\mathbf{R B}(u)$ is executed), $d_{u}$ is set to the minimum distance of a neighbor involved in a broadcast phase plus 1 ; see the macro compute $(u)$.

### 3.3 Typical Execution

Assume the system starts from a configuration where, for every process $u, s t_{u}=C$. A process $u$ detecting an inconsistency in $I$ (i.e., when $\neg \mathbf{P}_{\mathbf{\prime}} \mathbf{I C o r r e c t}(u)$ holds) stops executing $I$ and initiates a reset using rule_R $(u)$, unless one of its neighbors $v$ is already broadcasting a reset, in which case it joins the broadcast of some neighbor by rule_RB(u). To initiate a reset, $u$ sets $\left(s t_{u}, d_{u}\right)$ to $(R B, 0)$ meaning that $u$ is the root of a reset (see macro beRoot $(u)$ ), and resets its I's variables to an pre-defined state of I, which satisfies $\mathbf{P}_{\_}$reset $(u)$, by executing the macro $\operatorname{reset}(u)$. Whenever a process $v$ has a neighbor involved in a broadcast phase of a reset (status $R B$ ), it stops executing I and joins an existing reset using rule_RB(v), even if its state in $I$ is correct, (i.e., even if $\mathbf{P}$ _ICorrect $(v)$ holds). To join a reset, $v$ also switches its status to $R B$ and resets its I's variables $(\operatorname{reset}(v))$, yet it sets $d_{v}$ to the minimum distance of its neighbors involved in a broadcast phase plus 1 ; see the macro compute $(v)$. Hence, if the configuration of $I$ is not legitimate, then within at most $n$ rounds, each process receives the broadcast of some reset. Meanwhile, processes (temporarily) stop executing I until the reset terminates in their closed neighborhood thanks to the predicate P_Clean.

When a process $u$ involved in the broadcast phase of some reset realizes that all its neighbors are involved into a reset (i.e., have status $R B$ or $R F$ ), it initiates the feedback phase by switching to status $R F$, using rule $\_\mathbf{R F}(u)$. The feedback phase is then propagated up in the DAG described by the distance value: a broadcasting process $u$ switches to the feedback phase if each of its neighbors $v$ has not status $C$ and if $d_{v}>d_{u}$, then $v$ has status $R F$. This way the feedback phase is propagated up into the DAG within at most $n$ additional rounds. Once a root of some reset has status $R F$, it can initiate the last phase of the reset: all processes involves into the reset has to switch to status $C$, using rule_C, meaning that the reset is done. The values $C$ are propagated down into the reset DAG within at most $n$ additional rounds. A process $u$ can executing I again when all members of its closed neighborhood (that is, including $u$ itself) have status $C$, i.e., when it satisfies P_Clean $(u)$.

Hence, overall in this execution, the system reaches a configuration $\gamma$ where all resets are done within at most $3 n$ rounds. In $\gamma$, all processes have status $C$. However, process has not necessarily kept a state satisfying P_reset (i.e., the initial pre-defined state of I) in this configuration. Indeed, some process may have started executing I again before $\gamma$. However, the predicate $\mathbf{P}$ _Clean ensures that no resetting process has been involved in these latter (partial) executions of I. Hence, SDR rather ensures that all processes are in I's states that are coherent with each other from $\gamma$. That is, $\gamma$ is a so-called normal configuration, where P_Clean $(u) \wedge \mathbf{P}$ _ICorrect $(u)$ holds for every process $u$.

### 3.4 Stabilization of the Reset

If a process $u$ is in an incorrect state of Algorithm $\operatorname{SDR}$ (i.e., if $\mathbf{P} \_\mathbf{R 1}(u) \vee \mathbf{P} \_\mathbf{R 2}(u)$ holds), we proceed as for inconsistencies in Algorithm $I$. Either it joins an existing reset (using rule_RB $(u)$ ) because at least one of its neighbors is in a broadcast phase, or it initiates its own reset using rule_R(u). Notice also that
starting from an arbitrary configuration, the system may contain some reset in progress. However, similarly to the typical execution, the system stabilizes within at most $3 n$ rounds to a normal configuration.

Algorithm $\operatorname{SDR}$ is also efficient in moves. Indeed, in Sections 6 and 5 we will give two examples of composition I $\circ$ SDR that stabilize in a polynomial number of moves. Such complexities are mainly due to the coordination of the resets which, in particular, guarantees that if a process $u$ is enabled to initiate a reset $\left(\mathbf{P}_{-} \mathbf{U p}(u)\right)$ or the root of a reset with status $R B$, then it satisfies this disjunction since the initial configuration (cf., Theorem 3, page 12).

### 3.5 Requirements on the Input Algorithm

According to the previous explanation, Algorithm I should satisfy the following prerequisites:

1. Algorithm I should not write into the variables of $\operatorname{SDR}$, i.e., variables $s t_{u}$ and $d_{u}$, for every process $u$.
2. For each process $u$, Algorithm I should provide the two input predicates $\mathbf{P} \_\mathbf{I C o r r e c t}(u)$ and $\mathbf{P}$ _reset $(u)$ to SDR, and the macro reset $(u)$. Those inputs should satisfy:
(a) $\mathbf{P}_{\text {_I }}$ ICorrect $(u)$ does not involve any variable of SDR and is closed by Algorithm I.
(b) P_reset $(u)$ involves neither a variable of SDR nor a variable of a neighbor of $u$.
(c) If $\neg \mathbf{P}$ ICorrect $(u) \vee \neg \mathbf{P} \_\mathbf{C l e a n}(u)$ holds ( $n . b$. $\mathbf{P}_{-} \mathbf{C l e a n}(u)$ is defined in SDR), then no rule of Algorithm I is enabled at $u$.
(d) If $\mathbf{P} \_\operatorname{reset}(v)$ holds, for every $v \in N[u]$, then $\mathbf{P} \_I \operatorname{Correct}(u)$ holds.
(e) If $u$ performs a move in $\gamma \mapsto \gamma^{\prime}$, where, in particular, it modifies its variables in Algorithm I by executing reset ( $u$ ) (only), then $\mathbf{P}$ _reset $(u)$ holds in $\gamma^{\prime}$.

## 4 Correctness and Complexity Analysis

### 4.1 Partial Correctness

Lemma 1 In any terminal configuration of $\operatorname{SDR}, \neg \mathbf{P} \_\mathbf{R 1}(u) \wedge \neg \mathbf{P} \_\mathbf{R 2}(u) \wedge \mathbf{P} \_\operatorname{Correct}(u)$ holds for every process $u$.

Proof. Let $u$ be any process and consider any terminal configuration of SDR. Since rule_RB(u) and rule_R $(u)$ are disabled, $\neg \mathbf{P} \_\mathbf{R B}(u)$ and $\neg \mathbf{P}_{-} \mathbf{U p}(u)$ hold. Since $\neg \mathbf{P}_{-} \mathbf{R B}(u) \wedge \neg \mathbf{P} \_\mathbf{U p}(u)$ implies $\neg \mathbf{P} \_\mathbf{R 1}(u) \wedge \neg \mathbf{P} \_\mathbf{R 2}(u) \wedge \mathbf{P} \_\operatorname{Correct}(u)$, we are done.

Since $\neg \mathbf{P} \_\mathbf{R 2}(u) \equiv s t_{u}=C \vee \mathbf{P}_{\text {_reset }}(u)$, we have the following corollary.
Corollary 1 In any terminal configuration of SDR, st ${ }_{u}=C \vee \mathbf{P}$ _reset $(u)$ holds for every process $u$.
Lemma 2 In any terminal configuration of SDR, st $t_{u} \neq R B$ for every process $u$.
Proof. Assume, by the contradiction, that some process $u$ satisfies $s t_{u}=R B$ in a terminal configuration of SDR. Without the loss of generality, assume $u$ is a process such that $s t_{u}=R B$ with $d_{u}$ maximum. First, $\mathbf{P}_{\_}$reset $(u)$ holds by Corollary 1. Then, every neighbor $v$ of $u$ satisfies $s t_{v} \neq C$, since otherwise rule_RB $(v)$ is enabled. So, every $v$ satisfies $s t_{v} \in\{R B, R F\}$, st $v_{v}=R B \Rightarrow d i s t_{v} \leq d_{u}$ (by definition of $u$ ), and $s t_{v}=R F \Rightarrow \mathbf{P}_{\_}$reset $(v)$ (by Corollary 1). Hence, rule_RF( $u$ ) is enabled, a contradiction.

## Algorithm 1 Algorithm SDR, code for every process $u$

## Inputs:

- P_ICorrect $(u)$ : predicate from the input algorithm I
- P_reset $(u)$ : predicate from the input algorithm I
- $\operatorname{reset}(u):$ macro from the input algorithm I


## Variables:

- $s t_{u} \in\{C, R B, R F\} \quad$ : the status of $u$
- $d_{u} \in \mathbb{N} \quad$ : the distance value associated to $u$


## Predicates:

- P_Correct $(u) \equiv s t_{u}=C \Rightarrow$ P_ICorrect $(u)$
- P_Clean $(u) \equiv \forall v \in N[u], s t_{u}=C$
- $\mathbf{P} \_\mathbf{R 1}(u) \equiv s t_{u}=C \wedge \neg \mathbf{P} \operatorname{reset}(u) \wedge\left(\exists v \in N(u) \mid s t_{v}=R F\right)$
- P_RB(u) $\equiv s t_{u}=C \wedge\left(\exists v \in N(u) \mid s t_{v}=R B\right)$
- $\quad \mathbf{P}-\mathbf{R F}(u) \equiv s t_{u}=R B \wedge \mathbf{P} \operatorname{reset}(u) \wedge$
$\left(\forall v \in N(u),\left(s t_{v}=R B \wedge d_{v} \leq d_{u}\right) \vee\left(s t_{v}=R F \wedge \mathbf{P}\right.\right.$ _reset $\left.\left.(v)\right)\right)$
- P_C $(u) \equiv s t_{u}=R F \wedge$
$\left(\forall v \in N[u]\right.$, P_reset $\left.(v) \wedge\left(\left(s t_{v}=R F \wedge d_{v} \geq d_{u}\right) \vee\left(s t_{v}=C\right)\right)\right)$
- $\operatorname{P\_ R2}(u) \equiv s t_{u} \neq C \wedge \neg \mathbf{P}$ reset $(u)$
- P_Up $(u) \equiv \neg \mathbf{P} \_\mathbf{R B}(u) \wedge\left(\mathbf{P} \_\right.$R1 $(u) \vee \mathbf{P} \_$R2 $(u) \vee \neg \mathbf{P}$ _Correct $\left.(u)\right)$


## Macros

- beRoot(u) : $s t_{u}:=R B ; d_{u}:=0$;
- compute $(u)$ : $s t_{u}:=R B ; d_{u}:=\operatorname{argmin}_{\left(v \in N(u) \wedge s t_{v}=R B\right)}\left(d_{v}\right)+1$;


## Rules:

rule_RB $(u) \quad: \quad \mathbf{P} \_\mathbf{R B}(u) \rightarrow$ compute $(u) ;$ reset $(u) ;$
rule_RF $(u) \quad: \quad \operatorname{P} \_\mathbf{R F}(u) \quad \rightarrow \quad s t_{u}:=R F ;$
rule_C $(u) \quad: \quad$ P_C $(u) \quad \rightarrow \quad s t_{u}:=C$;
rule_R $(u) \quad: \quad \mathbf{P}_{-} \mathbf{U p}(u) \rightarrow \quad \operatorname{beRoot}(u) ; \operatorname{reset}(u) ;$

Lemma 3 In any terminal configuration of $\mathrm{SDR}, s t_{u} \neq R F$ for every process $u$.
Proof. Assume, by the contradiction, that some process $u$ satisfies $s t_{u}=R F$ in a terminal configuration of SDR. Without the loss of generality, assume $u$ is a process such that $s t_{u}=R F$ with $d_{u}$ minimum. First, every neighbor $v$ of $u$ satisfies $s t_{v} \neq R B$, by Lemma 2. Then, every neighbor $v$ of $u$ such that $s t_{v}=C$ also satisfies P_reset $(v)$, since otherwise P_R1 $(v)$ holds, contradicting then Lemma 1. Finally, by definition of $u$ and by Corollary 1, every neighbor $v$ of $u$ such that $s t_{v}=R F$ both satisfies dist $v \geq d_{u}$ and P_reset $(v)$. Hence, rule_C $(u)$ is enabled, a contradiction.

Theorem 1 For every configuration $\gamma$ of SDR, $\gamma$ is terminal if and only if $\mathbf{P}$ _Clean $(u) \wedge \mathbf{P}$ ICorrect $(u)$ holds in $\gamma$, for every process $u$.

Proof. Let $u$ be any process and assume $\gamma$ is terminal. By Lemmas 2 and 3, st ${ }_{u}=C$ holds in $\gamma$. So, P_Clean $(u)$ holds in $\gamma$. Moreover, since P_Correct $(u)$ holds (Lemma 1), P_ICorrect ( $u$ ) also holds in $\gamma$, and we are done.

Assume now that for every process $u$, P_Clean $(u) \wedge$ P_ICorrect $(u)$ holds in $\gamma$. Then, $s t_{u}=C$ for every process $u$, and so rule_C $(u)$, rule_RB $(u)$, and $\operatorname{rule} \_\mathbf{R F}(u)$ are disabled for every $u$. Then, since every process has status $C, \neg \mathbf{P} \_\mathbf{R 1}(u) \wedge \neg \mathbf{P} \_\mathbf{R 2}(u)$ holds, moreover, P_ICorrect $(u)$ implies $\mathbf{P}_{-}$Correct $(u)$, so rule_ $\mathbf{R}(u)$ is also disabled in $\gamma$. Hence $\gamma$ is terminal, and we are done.

### 4.2 Termination

From Requirements 1 and 2a, we know that Algorithm I does not write into $s t_{u}$ and P_ICorrect $(u)$ is closed by I, for every process $u$. Hence follows.

Remark 1 For every process u, predicate P_Correct(u) (defined in SDR) is closed by I.
Requirements $1,2 \mathrm{~b}$, and 2 c ensures the following property.
Lemma 4 For every process $u$, predicates $\neg \mathbf{P} \_\mathbf{R 1}(u), \neg \mathbf{P} \_\mathbf{R 2}(u)$, and $\mathbf{P} \_\mathbf{R B}(u)$ are closed by I.
Proof. Let $\gamma \mapsto \gamma^{\prime}$ be any step of I.

- Assume that $\neg \mathbf{P} \_$R1 $(u)$ holds at some process $p$ in $\gamma$.

If $s t_{u} \neq C \vee\left(\forall v \in N(u) \mid s t_{v} \neq R F\right)$ in $\gamma$, then $s t_{u} \neq C \vee\left(\forall v \in N(u) \mid s t_{v} \neq R F\right)$ still holds in $\gamma^{\prime}$ by Requirement 1 , and we are done.

Otherwise, $s t_{u}=C \wedge \mathbf{P}$ reset $(u) \wedge\left(\exists v \in N(u) \mid s t_{v}=R F\right)$ holds in $\gamma$. In particular, $\neg \mathbf{P}$ _Clean $(u)$ holds in $\gamma$. Hence, no rule of I is enabled at $u$ in $\gamma$, by Requirement 2c. Consequently, P_reset $(u)$ still holds in $\gamma^{\prime}$. Since, $\mathbf{P}$ reset $(u)$ implies $\neg \mathbf{P} \_\mathbf{R 1}(u)$, we are done.

- Assume that $\neg \mathbf{P} \_$R2 $(u)$ holds at some process $u$ in $\gamma$. If $s t_{u}=C$ holds in $\gamma$, then $s t_{u}=C$ holds in $\gamma^{\prime}$ by Requirement 1 , and so $\neg \mathbf{P} \_\mathbf{R 2}(u)$ still holds in $\gamma^{\prime}$. Otherwise, $s t_{u} \neq C \wedge \mathbf{P}$ reset $(u)$ holds in $\gamma$. In particular, $\neg \mathbf{P} \_$Clean $(u)$ holds in $\gamma$. Hence, no rule of I is enabled at $u$, by Requirement $2 \mathbf{c}$, and by Requirement 2b, P_reset $(u)$, and so $\neg \mathbf{P} \_$R2 $(u)$, still holds in $\gamma^{\prime}$.
- By Requirement 1, P_RB $(u)$ is closed by I.

Recall that two rules are mutually exclusive if there is no configuration $\gamma$ and no process $u$ such that both rules at enabled $u$ in $\gamma$. Two algorithms are mutually exclusive if their respective rules are pairwise mutually exclusive. Now, whenever a process $u$ is enabled in SDR, $\neg \mathbf{P} \_$ICorrect $(u) \vee \neg \mathbf{P}$ _Clean $(u)$ holds and, by Requirement 2 c , no rule of I is enabled at $u$. Hence, follows.

Remark 2 Algorithms SDR and I are mutually exclusive.
Lemma 5 Rules of Algorithm SDR are pairwise mutually exclusive.
Proof. Since $\mathbf{P} \_\mathbf{R B}(u)$ implies $s t_{u}=C, \mathbf{P}_{\_} \mathbf{R F}(u)$ implies $s t_{u}=R B$, and $\mathbf{P}_{-} \mathbf{C}(u)$ implies $s t_{u}=R F$, we can conclude that rule $\_$RB $(u)$, rule $\_\mathbf{R F}(u)$, and rule_C $(u)$ are pairwise mutually exclusive.

Then, since $\mathbf{P} \_\mathbf{U p}(u)$ implies $\neg \mathbf{P} \_\mathbf{R B}(u)$, rules rule $\_\mathbf{R}(u)$ and rule_RB $(u)$ are mutually exclusive.
P_C $(u)$ implies P_Correct $(u) \wedge \mathbf{P}$ reset $(u)$ which, in turn, implies $\neg \mathbf{P} \_\mathbf{U p}(u)$. Hence, rule_R $(u)$ and rule_C $(u)$ are mutually exclusive.
$\mathbf{P}$ _RF $(u)$ implies $s t_{u}=R B \wedge \mathbf{P}$ reset $(u)$. Now, $\mathbf{P} \_\mathbf{U p}(u)$ implies $s t_{u}=C \vee \neg \mathbf{P}$ reset $(u)$. Hence, rule $\_\mathbf{R}(u)$ and rule $\mathbf{R F}(u)$ are mutually exclusive.

Lemma 6 For every process $u$, predicates $\neg \mathbf{P} \_\mathbf{R 1}(u)$ and $\neg \mathbf{P} \_\mathbf{R 2}(u)$ are closed by I o SDR.
Proof. By Remark 2 and Lemma 4, to prove this lemma it is sufficient to show that $\neg \mathbf{P} \_\mathbf{R 1}(u)$ and $\neg \mathbf{P} \_\mathbf{R 2}(u)$ are closed by SDR, for every process $u$.

Predicate $\neg \mathbf{P} \_\mathbf{R 2}(u)$ only depends on variables of $u$ by Requirement 2 b . So, if $u$ does not move, $\neg \mathbf{P} \_\mathbf{R 2}(u)$ still holds. Assume $\neg \mathbf{P} \_\mathbf{R 2}(u)$ holds in $\gamma$ and $u$ executes a rule of SDR in $\gamma \mapsto \gamma^{\prime}$. If $u$ executes rule_RB $(u)$ or rule_R $(u)$, then $u$ modifies its variables in I by executing reset $(u)$. Hence, in both cases, P_reset $(u)$ holds in $\gamma^{\prime}$ by Requirement 2 e and as $\mathbf{P}$ _reset $(u)$ implies $\neg \mathbf{P} \_\mathbf{R 2}(u)$, we are done. Otherwise, $u$ executes rule_RF $(u)$ or rule_C $(u)$. In both cases, P_reset $(u)$ holds in $\gamma$ and so in $\gamma^{\prime}$ by Requirement 2 b , and we are done.

Assume now that the predicate $\neg \mathbf{P} \_\mathbf{R 1}(u)$ holds in $\gamma$ and consider any step $\gamma \mapsto \gamma^{\prime}$. Assume first that $u$ moves in $\gamma \mapsto \gamma^{\prime}$. If $u$ executes rule_RB $(u)$, $\operatorname{rule} \_\mathbf{R F}(u)$, or $\operatorname{rule} \_\mathbf{R}(u)$, then $s t_{u} \neq C$ in $\gamma^{\prime}$, hence $\neg \mathbf{P} \_$R1 $(u)$ holds in $\gamma^{\prime}$. If $u$ executes rule_C $(u)$ in $\gamma \mapsto \gamma^{\prime}, u$ satisfies $\mathbf{P}$ _reset $(u)$ in $\gamma$, and so in $\gamma^{\prime}$ by Requirement 2b. Since $\mathbf{P}$ _reset $(u)$ implies $\neg \mathbf{P} \_$R1 $(u)$, we are done. Assume now that $u$ does not move in $\gamma \mapsto \gamma^{\prime}$. In this case, $\mathbf{P} \_\mathbf{R 1}(u)$ may become true only if at least a neighbor $v$ of $u$ switches to status $R F$, by executing rule_RF $(v)$. Now, in this case, $s t_{u} \neq C$ in $\gamma$, and so in $\gamma^{\prime}$. Consequently, $\neg \mathbf{P} \_\mathbf{R 1}(u)$ still holds in $\gamma^{\prime}$.

Theorem 2 For every process $u$, $\mathbf{P} \_\operatorname{Correct}(u) \vee \mathbf{P} \_\mathbf{R B}(u)$ is closed by I o SDR.
Proof. By Remarks 1 and 2, and Lemma 4, to prove this lemma it is sufficient to show that P_Correct $(u)$ $\vee \mathbf{P} \_\mathbf{R B}(u)$ is closed by SDR, for every process $u$.

Let $\gamma \mapsto \gamma^{\prime}$ be any step of SDR such that $\mathbf{P} \_$Correct $(u) \vee \mathbf{P} \_$RB $(u)$ holds in $\gamma$.

- Assume P_Correct $(u)$ holds in $\gamma$. By Requirement 2a, if P_ICorrect $(u)$ holds in $\gamma$, then P_ICorrect $(u)$ still holds in $\gamma^{\prime}$, and as P_ICorrect $(u)$ implies P_Correct $(u)$, we are done.

Assume now $\neg \mathbf{P}$ _ICorrect $(u)$ holds in $\gamma$. Then, P_Correct $(u) \wedge \neg \mathbf{P}$ _ICorrect $(u)$ implies $s t_{u} \neq$ $C$ in $\gamma$. Since $\mathbf{P}_{-} \mathbf{C}(u)$ implies $\mathbf{P}$ ICorrect $(u)$ by Requirement 2d, rule_C $(u)$ is disabled in $\gamma$, and Consequently, $s t_{u} \neq C$ in $\gamma^{\prime}$, which implies that $\mathbf{P} \_\operatorname{Correct}(u)$ still holds in $\gamma^{\prime}$.

- Assume $\mathbf{P} \_\mathbf{R B}(u)$ holds in $\gamma$. If $u$ moves in $\gamma \mapsto \gamma^{\prime}$, then $u$ necessarily executes rule $\_\mathbf{R B}(u)$; see Lemma 5. In this case, $s t_{u}=R B$ in $\gamma^{\prime}$, which implies $\mathbf{P}_{-} \operatorname{Correct}(u)$ in $\gamma^{\prime}$.

If $u$ does not move, then at least one neighbor of $u$ should switch its status from $R B$ to either $C$ or $R F$ so that $\neg \mathbf{P} \_\mathbf{R B}(u)$ holds in $\gamma^{\prime}$. Any neighbor $v$ of $u$ satisfying $s t_{v}=R B$ may only change its status by executing rule_RF $(v)$ in $\gamma \mapsto \gamma^{\prime}$. Now, $\mathbf{r u l e} \_\mathbf{R F}(v)$ is necessarily disabled in $\gamma$ since $s t_{u}=C$. Hence, $\mathbf{P} \_\mathbf{R B}(u)$ still holds in $\gamma^{\prime}$ in this case.

From Lemma 6 and Theorem 2, we can deduce the following corollary.
Corollary 2 For every process $u, \neg \mathbf{P}_{-} \mathbf{U p}(u)$ is closed by I o SDR.

### 4.2.1 Roots.

If the configuration is illegitimate w.r.t. the initial algorithm, then some processes locally detect the inconsistency by checking their state and that of their neighbors (using Predicate $\mathbf{P}_{\_}$ICorrect). Such processes, called here roots, should initiates a reset. Then, each root $u$ satisfies $s t_{u} \neq C$ all along the reset processing. According to its status, a root is either alive or dead, as defined below.

Definition 1 Let $\mathbf{P}$ root $(u) \equiv s t_{u}=R B \wedge\left(\forall v \in N(u)\right.$, st $\left.v_{v}=R B \Rightarrow d_{v} \geq d_{u}\right)$.

- A process $u$ is said to be an alive root if $\mathbf{P} \_\mathbf{U p}(u) \vee \mathbf{P} \_$root $(u)$.
- A process $u$ is said to be an dead root if $s t_{u}=R F \wedge\left(\forall v \in N(u)\right.$, status $\left._{v} \neq C \Rightarrow d_{v} \geq d_{u}\right)$.

By definition, follows.

Remark 3 For every process $u$, if $\mathbf{P}_{-} \mathbf{C}(u)$ holds, then $u$ is a dead root.
The next theorem states that no alive root is created during an execution.

Theorem 3 For every process $u$, $\neg \mathbf{P} \_\operatorname{root}(u) \wedge \neg \mathbf{P} \_\mathbf{U p}(u)$ is closed by I o SDR.
Proof. By Requirement 1 and Corollary 2, $\neg \mathbf{P}_{\_} \operatorname{root}(u) \wedge \neg \mathbf{P}_{-} \mathbf{U p}(u)$ is closed by I. Hence, by Remark 2, it is sufficient to show that $\neg \mathbf{P} \_\mathbf{r o o t}(u) \wedge \neg \mathbf{P} \_\mathbf{U p}(u)$ is closed by SDR.

Let $\gamma \mapsto \gamma^{\prime}$ be any step of $\operatorname{SDR}$ such that $\neg \mathbf{P}_{\_} \operatorname{root}(u) \wedge \neg \mathbf{P}_{-} \mathbf{U p}(u)$ holds in $\gamma$. By Corollary 2, $\neg \mathbf{P} \_\mathbf{U p}(u)$ holds in $\gamma^{\prime}$. To show that $\neg \mathbf{P} \_\operatorname{root}(u)$ holds in $\gamma^{\prime}$, we now consider the following cases:
$s t_{u}=R F$ in $\gamma$ : In this case, rule_RB $(u)$ and rule_R $(u)$ are respectively disabled in $\gamma$ since $s t_{u} \neq C$ and $\neg \mathbf{P}_{-} \mathbf{U p}(u)$ hold in $\gamma$. So, st $t_{u} \neq R B$ in $\gamma^{\prime}$, which implies that $\neg \mathbf{P} \_\operatorname{root}(u)$ still holds in $\gamma^{\prime}$.
$s t_{u}=R B$ in $\gamma$ : Then, $\neg \mathbf{P}_{\_} \operatorname{root}(u)$ in $\gamma$ implies that there is a neighbor $v$ of $u$ such that $s t_{v}=R B \wedge d_{v}<$ $d_{u}$ in $\gamma$. Let $\alpha$ be the value of $d_{v}$ in $\gamma$. Since $\neg \mathbf{P} \_\mathbf{U p}(u)$ holds in $\gamma, u$ may only execute rule_RF $(u)$ in $\gamma \mapsto \gamma^{\prime}$ and, consequently, $d_{u}>\alpha$ in $\gamma^{\prime}$. Due to the value of $d_{u}$, v may only execute rule_R(v) in $\gamma \mapsto \gamma^{\prime}$. Whether or not $v$ moves, st $=R B \wedge d_{v} \leq \alpha$ in $\gamma^{\prime}$. Hence, $s t_{v}=R B \wedge d_{v}<d_{u}$ in $\gamma^{\prime}$, which implies that $\neg \mathbf{P} \_$root $(u)$ still holds in $\gamma^{\prime}$.
$s t_{u}=C$ in $\gamma$ : If $u$ does not moves in $\gamma \mapsto \gamma^{\prime}$, then $\neg \mathbf{P} \operatorname{root}(u)$ still holds in $\gamma^{\prime}$. Otherwise, since $\neg \mathbf{P} \_\mathbf{U p}(u)$ holds in $\gamma, u$ can only execute rule_RB $(u)$ in $\gamma \mapsto \gamma^{\prime}$. In this case, P_RB $(u)$ implies that there is a neighbor $v$ of $u$ such that $s t_{v}=R B$ in $\gamma$. Without the loss of generality, assume $v$ is the neighbor of $u$ such that $s t_{v}=R B$ with the minimum distance value in $\gamma$. Let $\alpha$ be the value of $d_{v}$ in $\gamma$. Then, $s t_{u}=R B$ and $d_{u}=\alpha+1$ in $\gamma^{\prime}$. Moreover, since $s t_{u}=C$ and $s t_{v}=R B$ in $\gamma, v$ may only execute rule $\_\mathbf{R}(v)$ in $\gamma \mapsto \gamma^{\prime}$. Whether or not $v$ moves, $s t_{v}=R B \wedge d_{v} \leq \alpha$ in $\gamma^{\prime}$. Hence, $s t_{v}=R B \wedge d_{v}<d_{u}$ in $\gamma^{\prime}$, which implies that $\neg \mathbf{P} \_\operatorname{root}(u)$ still holds in $\gamma^{\prime}$.

### 4.2.2 Move Complexity.

Definition $2(A R)$ Let $\gamma$ be a configuration of I o SDR. We denote by $A R(\gamma)$ the set of alive roots in $\gamma$.
By Theorem 3, follows.
Remark 4 Let $\gamma_{0} \cdots \gamma_{i} \cdots$ be any execution of $\mathrm{I} \circ \mathrm{SDR}$. For every $i>0, A R\left(\gamma_{i}\right) \subseteq A R\left(\gamma_{i-1}\right)$.
Based on the aforementioned property, we define below the notion of segment.
Definition 3 (Segment) Let $e=\gamma_{0} \cdots \gamma_{i} \cdots$ be any execution of $I \circ$ SDR.

- If for every $i>0,\left|A R\left(\gamma_{i-1}\right)\right|=\left|A R\left(\gamma_{i}\right)\right|$, then the first segment of e is $e$ itself, and there is no other segment.
- Otherwise, let $\gamma_{i-1} \mapsto \gamma_{i}$ be the first step of e such that $\left|A R\left(\gamma_{i-1}\right)\right|>\left|A R\left(\gamma_{i}\right)\right|$. The first segment of $e$ is the prefix $\gamma_{0} \cdots \gamma_{i}$ and the second segment of e is the first segment of the suffix of e starting in $\gamma_{i}$, and so forth.

By Remark 4, follows.
Remark 5 Every execution of I O SDR contains at most $n+1$ segments where $n$ is the number of processes.
We now study how a reset propagates into the network. To that goal, we first define the notion of reset parent. Roughly speaking, the parents of $u$ in a reset are its neighbors (if any) that have caused its reset.

Definition 4 (Reset Parent and Children) RParent $(v, u)$ holds for any two processes $u$ and $v$ if $v \in$ $N(u), s t_{u} \neq C$, P_reset $(u), d_{u}>d_{v}$, and $\left(s t_{u}=s t_{v} \vee s t_{v}=R B\right)$.

Whenever RParent $(v, u)$ holds, $v$ (resp., $u$ ) is said to be a reset parent of $u$ in (resp., a reset child of $v$ ).
Remark that in a given configuration, a process may have several reset parents. Below, we define the reset branches, which are the trails of a reset in the network.

Definition 5 (Reset Branch) $A$ reset branch is a sequence of processes $u_{1}, \ldots, u_{k}$ for some integer $k \geq 1$, such that $u_{1}$ is an alive or dead root and, for every $1<i \leq k$, we have RParent $\left(u_{i-1}, u_{i}\right)$. The process $u_{i}$ is said to be at depth $i-1$ and $u_{i}, \cdots, u_{k}$ is called $a$ reset sub-branch. The process $u_{1}$ is the initial extremity of the reset branch $u_{1}, \ldots, u_{k}$.

Lemma 7 Let $u_{1}, \ldots, u_{k}$ be any reset branch.

1. $k \leq n$,
2. If $s t_{u_{1}}=C$, then $k=1$. Otherwise, $s t_{u_{1}} \cdots s t_{u_{k}} \in R B^{*} R F^{*}$.
3. $\forall i \in\{2, \ldots, k\}, u_{i}$ is neither an alive, not a dead root.

Proof. Let $i$ and $j$ such that $1 \leq i<j \leq k$. By definition, $d_{u_{i}}<d_{u_{j}}$ and so $u_{i} \neq u_{j}$. Hence, in a reset branch, each node appears at most once, and Lemma 7.1 holds.

Let $i \in\{2, \ldots, k\}$. Lemma 7.2 immediately follows from the following three facts, which directly derive from the definition of reset parent.

- $s t_{u_{i}} \neq C$.
- $s t_{u_{i}}=R B \Rightarrow s t_{u_{i-1}}=R B$.
- $s t_{u_{i}}=R F \Rightarrow s t_{u_{i-1}} \in\{R B, R F\}$.

Lemma 7.3 immediately follows from those two facts.

- $u_{i}$ is not a dead root, since $u_{i-1} \in N\left(u_{i}\right) \wedge s t_{u_{i-1}} \neq C \wedge d_{u_{i-1}}<d_{u_{i}}$.
- $u_{i}$ is not an alive root, indeed
- $\neg \mathbf{P} \_$root $(u)$ holds, since $u_{i-1} \in N\left(u_{i}\right) \wedge\left(s t_{u_{i}}=R B \Rightarrow s t_{u_{i-1}}=R B\right) \wedge d_{u_{i-1}}<d_{u_{i}}$.
- $\neg \mathbf{P} \_\mathbf{U p}\left(u_{i}\right)$ holds since $\neg \mathbf{P} \_\mathbf{R 1}(u) \wedge \neg \mathbf{P}$ _R2 $(u)$ holds because $\mathbf{P}$ reset $(u)$ holds, and $\mathbf{P}$ _Correct $(u)$ holds because $s t_{u} \neq C$.

Remark 6 In a configuration, a process u may belong to several branches. Precisely, u belongs to at least one reset branch, unless st ${ }_{u}=C \wedge \mathbf{P}$ _ICorrect $(u)$ holds.

Lemma 8 Let $\gamma_{x} \mapsto \gamma_{x+1}$ be a step of I $\circ$ SDR. Let $u_{1}, \ldots, u_{k}$ be a reset branch in $\gamma_{x}$. If $u_{1}$ is an alive root in $\gamma_{x+1}$, then $u_{1}, \ldots, u_{k}$ is a reset branch in $\gamma_{x+1}$.

Proof. We first show the following two claims:
Claim 1: If $u_{1}$ moves in $\gamma_{x} \mapsto \gamma_{x+1}$, then $u_{1}$ necessarily executes rule $\mathbf{R}$ in $\gamma_{x} \mapsto \gamma_{x+1}$.
Proof of the claim: Since $u_{1}$ is an alive root in $\gamma_{x+1}, u_{1}$ is an alive root in $\gamma_{x}$, by Theorem 3. Assume now, by the contradiction, that $u_{1}$ moves, but does not execute rule_R in $\gamma_{x} \mapsto \gamma_{x+1}$. Then, $\neg \mathbf{P} \_\mathbf{U p}\left(u_{k}\right)$ holds in $\gamma_{x}$, by Remark 2 and Lemma 5. So, by definition of alive root, $s t_{u}=R B$ in $\gamma_{x}$ and, from the code of SDR and Requirement 2c, $u_{1}$ executes rule_RF in $\gamma_{x} \mapsto \gamma_{x+1}$. Consequently, $s t_{u}=R F$. Now, $\neg \mathbf{P} \_\mathbf{U p}\left(u_{k}\right)$ still holds in $\gamma_{x+1}$, by Corollary 2. Hence, $u_{1}$ is not an alive root in $\gamma_{x+1}$, a contradiction.

Claim 2: For every $i \in\{2, \ldots, k\}$, if $u_{i}$ moves $\gamma_{x} \mapsto \gamma_{x+1}$, then $u_{i}$ executes rule_RF in $\gamma_{x} \mapsto \gamma_{x+1}$ and in $\gamma_{x}$ we have $s t_{u_{i}}=R B$ and $i<k \Rightarrow s t_{u_{i+1}}=R F$.
Proof of the claim: We first show that only rule_RF may be enabled at $u_{i}$ in $\gamma_{x}$.

- By definition, $s t_{u_{i}} \neq C$ and so $\neg \mathbf{P}$ _Clean $\left(u_{i}\right)$ holds in $\gamma_{x}$. Thus, by Requirement 2 c , all rules of I that are disabled at $u_{i}$ in $\gamma_{x}$.
- rule_RB $\left(u_{i}\right)$ is disabled in $\gamma_{x}$ since $s t_{u_{i}} \neq C$ (by definition).
- The fact that $u_{i}$ is not a dead root in $\gamma_{x}$ (Lemma 7) implies that rule_C $\left(u_{i}\right)$ is disabled in $\gamma_{x}$ (Remark 3).
- rule_R $\left(u_{i}\right)$ is disabled in $\gamma_{x}$ since $u_{i}$ is not an alive root (Lemma 7).

Hence, $u_{i}$ can only executes rule_RF in $\gamma_{x} \mapsto \gamma_{x+1}$. In this case, $s t_{u_{i}}=R B$ in $\gamma_{x}$; see the guard of rule_RF $\left(u_{i}\right)$. Moreover, if $i<k$, then $s t_{u_{i}}=R B \wedge R \operatorname{Parent}\left(u_{i}, u_{i+1}\right)$ implies that $u_{i+1} \in N\left(u_{i}\right)$, $s t_{u_{i+1}} \in\{R B, R F\}$, and $d_{u_{i}}<d_{u_{i+1}}$. Now, if $u_{i+1} \in N\left(u_{i}\right), s t_{u_{i+1}}=R B$, and $d_{u_{i}}<d_{u_{i+1}}$, then rule $\_\mathbf{R F}\left(u_{i}\right)$ is disabled. Hence, if $u_{i}$ moves $\gamma_{x} \mapsto \gamma_{x+1}$ and $i<k$, then $u_{i}$ executes rule_RF and so $s t_{u_{i+1}}=R F$ in $\gamma_{x}$.

Then, we proceed by induction on $k$. The base case $(k=1)$ is trivial. Assume now that $k>1$. Then, by induction hypothesis, $u_{1}, \ldots, u_{k-1}$ is a reset branch in $\gamma_{x+1}$. Hence, to show that $u_{1}, \ldots, u_{k}$ is a reset branch in $\gamma_{x+1}$, it is sufficient to show that $\operatorname{RParent}\left(u_{k-1}, u_{k}\right)$ holds in $\gamma_{x+1}$. Since, $\operatorname{RParent}\left(u_{k-1}, u_{k}\right)$ holds in $\gamma_{x}$, we have $s t_{u_{k}} \neq C$ in $\gamma_{x}$. So, we now study the following two cases:

- $s t_{u_{k}}=R B$ in $\gamma_{x}$. By Claim 2, $u_{k}$ may only execute rule_RF in $\gamma_{x} \mapsto \gamma_{x+1}$. Consequently, $s t_{u_{k}} \in\{R B, R F\} \wedge \mathbf{P} \operatorname{reset}\left(u_{k}\right) \wedge d_{u_{k}}=d$ holds in $\gamma_{x+1}$, where $d>0$ is the value of $d_{u_{k}}$ in $\gamma_{x}$. Consider now process $u_{k-1}$. Since $s t_{u_{k}}=R B$ in $\gamma_{x}, s t_{u_{k-1}}=R B$ too in $\gamma_{x}$ (Lemma 7).
If $u_{k-1}$ does not move in $\gamma_{x} \mapsto \gamma_{x+1}$, then RParent $\left(u_{k-1}, u_{k}\right)$ still holds in $\gamma_{x+1}$, and we are done.
Assume now that $u_{k-1}$ moves in $\gamma_{x} \mapsto \gamma_{x+1}$. Then, we necessarily have $k=2$ since otherwise, Claim 2 applies for $i=k-1$ : $u_{k-1}$ moves in $\gamma_{x} \mapsto \gamma_{x+1}$ only if $s t_{u_{k}}=R F$ in $\gamma_{x}$. Now, $k=2$ implies that $u_{k-1}$ executes rule_R in $\gamma_{x} \mapsto \gamma_{x+1}$ (by Claim 1), so $s t_{u_{k-1}}=R B$ and $d_{u_{k-1}}=0<d$ in $\gamma_{x+1}$. Consequently, RParent $\left(u_{k-1}, u_{k}\right)$ still holds in $\gamma_{x+1}$, and we are done.
- $s t_{u_{k}}=R F$ in $\gamma_{x}$.

By Claim 2, $u_{k}$ does not move in $\gamma_{x} \mapsto \gamma_{x+1}$. So, st $t_{u_{k}}=R F \wedge \mathbf{P}$ reset $\left(u_{k}\right) \wedge d_{u_{k}}=d$ holds in $\gamma_{x+1}$, where $d>0$ is the value of $d_{u_{k}}$ in $\gamma_{x}$.
If $u_{k-1}$ does not move in $\gamma_{x} \mapsto \gamma_{x+1}$, we are done. Assume, otherwise, that $u_{k-1}$ moves in $\gamma_{x} \mapsto$ $\gamma_{x+1}$.
Then, if $k=2$, then $u_{k-1}$ executes rule_R in $\gamma_{x} \mapsto \gamma_{x+1}$ (by Claim 1), so $s t_{u_{k-1}}=R B$ and $d_{u_{k-1}}=0<d$ in $\gamma_{x+1}$, and so RParent $\left(u_{k-1}, u_{k}\right)$ still holds in $\gamma_{x+1}$.
Otherwise ( $k>2$ ), $u_{k-1}$ necessarily executes rule_RF in $\gamma_{x} \mapsto \gamma_{x+1}$ (by Claim 2): st $t_{u_{k-1}}=$ $R F$ and $d_{u_{k-1}}<d_{u_{k}}$ in $\gamma_{x+1}$ (n.b., neither $d_{u_{k-1}}$ nor $d_{u_{k}}$ is modified in $\gamma_{x} \mapsto \gamma_{x+1}$ ). Hence, RParent $\left(u_{k-1}, u_{k}\right)$ still holds in $\gamma_{x+1}$, and we are done.

Lemma 9 Let u be any process. During a segment $S=\gamma_{i} \cdots \gamma_{j}$ of execution of I o SDR, if u executes the rule rule_RF, then $u$ does not execute any other rule of SDR in the remaining of $S$.

Proof. Let $\gamma_{x} \mapsto \gamma_{x+1}$ be a step of $S$ in which $u$ executes rule_RF. Let $\gamma_{y} \mapsto \gamma_{y+1}$ (with $y>x$ ) be the next step in which $u$ executes its next rule of SDR. (If $\gamma_{x} \mapsto \gamma_{x+1}$ or $\gamma_{y} \mapsto \gamma_{y+1}$ does not exist, then the lemma trivially holds.) Then, since $\operatorname{rule} \mathbf{R F}(u)$ is enabled in $\gamma_{x}, \neg \mathbf{P} \_\mathbf{U p}(u)$ holds in $\gamma_{x}$, by Lemma 5. Consequently, $\neg \mathbf{P} \_\mathbf{U p}(u)$ holds forever from $\gamma_{x}$, by Corollary 2 . Hence, from the code of SDR and Requirement $2 \mathbf{c}$, $u$ necessarily executes rule_C in $\gamma_{y} \mapsto \gamma_{y+1}$ since $s t_{u}=R F \wedge \neg \mathbf{P} \_\mathbf{U p}(u)$ holds in $\gamma_{y}$. In $\gamma_{x}$, since $s t_{u}=R B, u$ belongs to some reset branches (Remark 6) and all reset branches containing $u$ have an alive root (maybe $u$ ) of status $R B$ (Lemma 7). Let $v$ be any alive root belonging to a reset branch containing $u$ in $\gamma_{x}$. In $\gamma_{y}, u$ is the dead root, since $\mathbf{P}_{-} \mathbf{C}(u)$ holds (Remark 3). By Lemma 7, either $u=v$ or $u$ no more belong to a reset branch whose initial extremity is $v$. By Lemma 8 and Theorem $3, v$ is no more an alive root in $\gamma_{y}$. Still by Theorem 3, the number of alive roots necessarily decreased between $\gamma_{x}$ and $\gamma_{y}$ : $\gamma_{x} \mapsto \gamma_{x+1}$ and $\gamma_{y} \mapsto \gamma_{y+1}$ belong to two distinct segments of the execution.

Theorem 4 The sequence of rules of SDR executed by a process $u$ in a segment of execution of I o SDR belongs to the following language:

$$
(\text { rule_C }+\varepsilon)(\text { rule_RB }+ \text { rule_R }+\varepsilon)(\text { rule_RF }+\varepsilon)
$$

Proof. From the code of SDR and Requirement 2c, we know that after any execution of rule_C ( $u$ ), the next rule of SDR $u$ will execute (if any), is either rule_RB or rule_R. Similarly, immediately after an execution of rule_RB $(u)$ (resp., rule_R $(u)$ ), $s t_{u}=R B \wedge \mathbf{P}$ _reset $(u)$ holds (see Requirement 2 e ) and $\mathbf{P}$ _reset $(u)$ holds while $u$ does not switch to status $C$ (Requirements 2 b and 2c). So the next rule of SDR $u$ will execute (if any) is rule_RF. Finally, immediately after any execution of rule_RF $(u), s t_{u}=R F \wedge \mathbf{P}$ _reset $(u)$ holds until (at least) the next execution of a rule of SDR since P_reset ( $u$ ) holds while $u$ does not switch to status $C$ (Requirements 2 b and 2c). Then, the next rule of SDR $u$ will execute (if any) is rule_C. However, if this latter case happens, $\operatorname{rule} \mathbf{R F}(u)$ and rule_C $(u)$ are executed in different segments, by Lemma 9.

Since a process can execute rules of I only if its status is $C$, we have the following corollary.
Corollary 3 The sequence of rules executed by a process $u$ in a segment of execution of I o SDR belongs to the following language:

$$
(\text { rule_C }+\varepsilon) \text { words_I }(\text { rule_RB }+ \text { rule_R }+\varepsilon)\left(\text { rule } \_\mathbf{R F}+\varepsilon\right)
$$

where words_I is any sequence of rules of I .
From Remark 5 and Theorem 4, follows.
Corollary 4 Any process $u$ executes at most $3 n+3$ rules of SDR in any execution of $\mathrm{I} \circ \mathrm{SDR}$.
Let $S=\gamma_{0} \cdots \gamma_{j} \cdots$ be a segment of execution of I $\circ$ SDR. Let $c_{\mathrm{I}}^{S}$ be the configuration of I in which every process $u$ has the local state $c_{\mathrm{I}}^{S}(u)$ defined below.

1. $c_{\mathrm{I}}^{S}(u)=\gamma_{0 \mid \mathrm{I}}(u)$, if $u$ never satisfies $s t_{u}=C$ in $S$,
2. $c_{\mathrm{I}}^{S}(u)=\gamma_{i \mid \mathrm{I}}(u)$ where $\gamma_{i}$ is the first configuration such that $s t_{u}=C$ in $S$, otherwise.

The following lemma is a useful tool to show the convergence of $I \circ S D R$.

Lemma 10 Let $S=\gamma_{0} \cdots \gamma_{j} \cdots$ be a segment of execution of $I \circ$ SDR. For every process $u$, let words_I $(u)$ be the (maybe empty) sequence of rules of I executed by $u$ in $S$. There is a prefix of execution of I starting in $c_{\mathrm{I}}^{S}$ that consists of the executions of words_I(u), for every process $u$.

Proof. For every process $u$, for every $i \geq 0$, let prewords_I $\mathbf{I}\left(\gamma_{i}, u\right)$ be the prefix of words_I $(u)$ executed by process $u$ in the prefix $\gamma_{0} \cdots \gamma_{i}$ of $S$. For every $i \geq 0$, let $\mathbf{c w}_{-} \mathbf{I}\left(\gamma_{i}\right)$ be the configuration of $I$ in which every process $u$ has the local state $\mathbf{c} \mathbf{w}_{-} \mathbf{I}\left(\gamma_{i}\right)(u)$ defined below.

- $\mathbf{c w} \_\mathbf{I}\left(\gamma_{i}\right)(u)=c_{\mathrm{I}}^{S}(u)$, if prewords $\_\mathbf{I}\left(\gamma_{i}, u\right)=\emptyset$,
- $\mathbf{c w} \_\mathbf{I}\left(\gamma_{i}\right)(u)=s$ where $s$ is the state assigned to $u$ by the execution of its last rule of prewords $\_\mathbf{I}\left(\gamma_{i}, u\right)$ in the prefix $\gamma_{0} \cdots \gamma_{i}$ of $S$, otherwise.

The lemma is immediate from the following induction: for every $i \geq 0$, there is a possible prefix of execution of I that starts from $c_{\mathrm{I}}^{S}$, ends in $\mathbf{c w}_{-} \mathbf{I}\left(\gamma_{i}\right)$, and consists of the executions of prewords_I $\left(\gamma_{i}, u\right)$, for every process $u$. The base case $(i=0)$ is trivial. Assume the induction holds for some $i \geq 0$ and consider the case $i+1$. For every process $u$, either $u$ does not execute any rule of I in $\gamma_{i} \mapsto \gamma_{i+1}$ and prewords_I $\left(\gamma_{i+1}, u\right)=$ prewords_I $\left(\gamma_{i}, u\right)$, or $u$ executes some rule $R_{u}$ of I in $\gamma_{i} \mapsto \gamma_{i+1}$ and prewords_I $\mathbf{I}\left(\gamma_{i+1}, u\right)=$ prewords_I $\left(\gamma_{i}, u\right) \cdot R_{u}$. If all processes satisfy the former case, then, by induction hypothesis, we are done. Otherwise, by induction hypothesis, it is sufficient to show the transition from $\mathbf{c w} \mathbf{I}_{\mathbf{I}}\left(\gamma_{i}\right)(u)$ to $\mathbf{c w}_{-} \mathbf{I}\left(\gamma_{i+1}\right)(u)$ consisting of the execution of $R_{u}$ by every process $u$ such that prewords_ $\mathbf{I}\left(\gamma_{i+1}, u\right)=$ prewords_I $\left(\gamma_{i}, u\right) \cdot R_{u}$ is a possible step of $I$. To see this, consider any process $u$ such that prewords_I $\left(\gamma_{i+1}, u\right)=$ prewords_I $\left(\gamma_{i}, u\right) \cdot R_{u}$. Since $u$ executes $R_{u}$ in $\gamma_{i} \mapsto \gamma_{i+1}$, P_Clean $(u)$ holds in $\gamma_{i}$ meaning that every member $v$ of $N[u]$ (in particular, $u$ ) satisfies $s t_{v}=C$ in $\gamma_{i}$. Then, by Corollary 3, every $v$ (in particular, $u$ ) is in the state $\mathbf{c} \mathbf{w}_{\_} \mathbf{I}\left(\gamma_{i}\right)(u)$. Hence, as $R_{u}$ is enabled in $\gamma_{i}$, $R_{u}$ is enabled in $\mathbf{c w} \mathbf{I}\left(\gamma_{i}\right)$ too, and we are done.

### 4.2.3 Round Complexity.

Below, we use the notion of attractor, defined at the beginning of the section.

## Definition 6 (Attractors)

- Let $\mathcal{P}_{1}$ a predicate over configurations of $\operatorname{I} \circ$ SDR which is true if and only if $\neg \mathbf{P}_{-} \mathbf{U} \mathbf{p}(u)$ holds, for every process $u$.
- Let $\mathcal{P}_{2}$ a predicate over configurations of $\mathrm{I} \circ \mathrm{SDR}$ which is true if and only if (1) $\mathcal{P}_{1}$ holds and (2) $\neg \mathbf{P} \_\mathbf{R B}(u)$ holds, for every process $u$.
- Let $\mathcal{P}_{3}$ a predicate over configurations of $\mathrm{I} \circ \mathrm{SDR}$ which is true if and only if (1) $\mathcal{P}_{2}$ holds and (2) $s t_{u} \neq R B$, for every process $u$.
- $\mathcal{P}_{4}$ a predicate over configurations of $\mathrm{I} \circ \mathrm{SDR}$ which is true if and only if (1) $\mathcal{P}_{3}$ holds and (2) st ${ }_{u} \neq R F$, for every process $u$.
In the following, we call normal configuration any configuration satisfying $\mathcal{P}_{4}$.
Lemma $11 \mathcal{P}_{1}$ is an attractor for I o SDR. Moreover, I o SDR converges from true to $\mathcal{P}_{1}$ within at most one round.

Proof. First, for every process $u, \neg \mathbf{P}_{\_} \mathbf{U p}(u)$ is closed by I o SDR (Corollary 2). Consequently, $\mathcal{P}_{1}$ is closed by I $\circ$ SDR. Moreover, to show that I o SDR converges from true to $\mathcal{P}_{1}$ within at most one round, it is sufficient to show that any process $p$ satisfies $\neg \mathbf{P} \_\mathbf{U p}(u)$ during the first round of any execution of $I \circ$ SDR. This property is immediate from the following two claims.

Claim 1: If $\mathbf{P}_{-} \mathbf{U p}(u)$, then $u$ is enabled.
Proof of the claim: By definition of rule_R(u).
Claim 2: If $\mathbf{P} \_\mathbf{U p}(u)$ holds in $\gamma$ and $u$ moves in the next step $\gamma \mapsto \gamma^{\prime}$, then $\neg \mathbf{P} \_\mathbf{U p}(u)$ holds in $\gamma^{\prime}$.
Proof of the claim: First, by Remark 2, Lemma 5, and the guard of rule_R(u), rule_R(u) is executed in $\gamma \mapsto \gamma^{\prime}$. Then, immediately after rule_R $(u)$, st $t_{u}=R B$ and $\mathbf{P}_{\text {reset }}(u)$ holds (see Requirement 2e), now $s t_{u}=R B$ and P_reset $(u)$ implies $\neg \mathbf{P} \_\mathbf{U p}(u)$.

Lemma $12 \mathcal{P}_{2}$ is closed by I o SDR.
Proof. By Requirement $1, \neg \mathbf{P} \_\mathbf{R B}(u)$ is closed by $I$, for every process $u$. So, by Lemma 11 and Remark 2, it is sufficient to show that for every step $\gamma \mapsto \gamma^{\prime}$ of I ○ SDR such that $\mathcal{P}_{2}$ holds in $\gamma$, for every process $u$, if $u$ executes a rule of SDR in $\gamma \mapsto \gamma^{\prime}$, then $\neg \mathbf{P} \_\mathbf{R B}(u)$ still holds in $\gamma^{\prime}$.

So, assume any such step $\gamma \mapsto \gamma^{\prime}$ and any process $u$.

- If $s t_{u}=C$ in $\gamma$, then $\forall v \in N(u), s t_{v} \neq R B$ in $\gamma$, since $\gamma$ satisfies $\mathcal{P}_{2}$. Now, no rule rule_RB or rule $\_\mathbf{R}$ can be executed in $\gamma \mapsto \gamma^{\prime}$ since $\gamma$ satisfies $\mathcal{P}_{2}$. So, $\forall v \in N(u)$, st $t_{v} \neq R B$ in $\gamma^{\prime}$, and consequently $\neg \mathbf{P} \_\mathbf{R B}(u)$ still holds in $\gamma^{\prime}$.
- If $s t_{u} \neq C$ in $\gamma^{\prime}$, then $\neg \mathbf{P} \_\mathbf{R B}(u)$ holds in $\gamma^{\prime}$.
- Assume now that $s t_{u} \neq C$ in $\gamma$ and $s t_{u}=C$ in $\gamma^{\prime}$. Then, $u$ necessarily executes rule_C in $\gamma \mapsto \gamma^{\prime}$. In this case, $\forall v \in N(u), s t_{v} \neq R B$ in $\gamma$. Now, no rule rule_RB or rule_R can be executed in $\gamma \mapsto \gamma^{\prime}$ since $\gamma$ satisfies $\mathcal{P}_{2}$. So, $\forall v \in N(u), s t_{v} \neq R B$ in $\gamma^{\prime}$, and consequently $\neg \mathbf{P} \_\mathbf{R B}(u)$ still holds in $\gamma^{\prime}$.

Hence, in all cases, $\neg \mathbf{P} \_\mathbf{R B}(u)$ still holds in $\gamma^{\prime}$, and we are done.

Lemma 13 I o SDR converges from $\mathcal{P}_{1}$ to $\mathcal{P}_{2}$ within at most $n-1$ rounds.
Proof. Let $u$ be any process of status $R B$. Then, $u$ belongs to at least one reset branch (Remark 6). Let $m d(u)$ be the maximum depth of $u$ in a reset branch it belongs to. Then, $m d(u)<n$, by Lemma 7 .

Consider now any execution $e=\gamma_{0} \cdots \gamma_{i} \cdots$ of I o SDR such that $\gamma_{0}$ satisfies $\mathcal{P}_{1}$. Remark first that from $\gamma_{0}$, rule_R $\mathbf{R}(v)$ is disabled forever, for every process $v$, since $\mathcal{P}_{1}$ is closed by I o SDR (Lemma 11).

Claim 1: If some process $u$ satisfies st $t_{u}=R B$ in some configuration $\gamma_{i}(i \geq 0)$, then from $\gamma_{i}$, while $s t_{u}=R B, m d(u)$ cannot decrease.
Proof of the claim: Consider any $\gamma_{i} \mapsto \gamma_{i+1}$ where $s t_{u}=R B$ both in $\gamma_{i}$ and $\gamma_{i+1}$. This in particular means that $u$ does not move in $\gamma_{i} \mapsto \gamma_{i+1}$. Let $u_{1}, \ldots, u_{k}=u$ be a reset branch in $\gamma_{i}$ such that $k=m d(u) . \forall x \in\{1, \ldots, k-1\}, u_{x}$ has a neighbor ( $u_{x+1}$ ) such that $s t_{x+1}=R B \wedge d_{x+1}>d_{u}$ in $\gamma_{i}$, by Lemma 7 and the definition of a reset branch. Hence, every $u_{x}$ is disabled in $\gamma_{i}$. Consequently, $u$ is still at depth at least $k$ in a reset branch defined in $\gamma_{i+1}$, and we are done.

Claim 2: For every process u that executes rule_RB $(u)$ in some step $\gamma_{i} \mapsto \gamma_{i+1}$ of the $j$ th round of $e$, we have st ${ }_{u}=R B \wedge m d(u) \geq j$ in $\gamma_{i+1}$.
Proof of the claim: We proceed by induction. Assume a process $u$ executes rule_RB $(u)$ in some step $\gamma_{i} \mapsto \gamma_{i+1}$ of the first round of $e$. In $\gamma_{i}$, there is some neighbor $v$ of $u$ such that $s t_{v}=R B$. Since $s t_{u}=C$ in $\gamma_{i}, v$ is disabled in $\gamma_{i}$. Consequently, RParent $(v, u)$ holds in $\gamma_{i+1}$, and so $s t_{u}=$ $R B \wedge m d(u) \geq 1$ holds in $\gamma_{i+1}$. Hence, the claim holds for $j=1$.
Assume now that the claim holds in all of the $j$ first rounds of $e$, with $j \geq 1$.
Assume, by the contradiction, that some process $u$ executes rule_RB in a step $\gamma_{i} \mapsto \gamma_{i+1}$ of the $(j+1)^{t h}$ round of $e$, and does not satisfy $s t_{u}=R B \wedge m d(u) \geq j+1$ in $\gamma_{i+1}$. Then, by definition of rule_RB $(u), s t_{u}=C$ in $\gamma_{i}$ and $s t_{u}=R B \wedge m d(u)<j+1$ in $\gamma_{i+1}$. Let $x$ be the value of $m d(u)$ in $\gamma_{i+1}$. We have $x<j+1$. Without the loss of generality, assume that no process satisfies this condition before $u$ in the $(j+1)^{\text {th }}$ round of $e$ and any process $v$ that fulfills this condition in the same step as $u$ satisfies $s t_{v}=R B \wedge x \leq m d(v)<j+1$ in $\gamma_{i+1}$. Then, by definition $m d(u)$ and Lemma 7, there is a neighbor $v$ of $u$ such $s t_{v}=R B$ and $m d(v)=x-1<j$ in $\gamma_{i+1}$. Moreover, by definition of $u$ and Claim 1, st $t_{v}=R B$ and $m d(v) \leq x-1<j$ since (at least) the first configuration of the $(j+1)$ th round of $e$, which is also the last configuration of the $j$ th round of $e$. So, by induction hypothesis, st $v_{v}=R B$ and $m d(v) \leq x-1<j$ since (at least) the end of the $(x-1)$ th round of $e$. If $s t_{u} \neq C$ in the last configuration of the $(x-1)$ th round of $e$, then $s t_{u} \neq C$ continuously until $\gamma_{i}$ (included) since meanwhile rule_C(u) is disabled because $s t_{v}=R B$. Hence, $u$ cannot execute rule $\quad \mathbf{R B}(u)$ in $\gamma_{i} \mapsto \gamma_{i+1}$, a contradiction. Assume otherwise that $s t_{u}=C$ in the last configuration of the $(x-1)$ th round of $e$. Then, $u$ necessarily executes rule_RB during the $x$ th round of $e$, but not in the $(j+1)^{t h}$ round of $e$ since $s t_{v}=R B$ continuously until $\gamma_{i}$ (included), indeed, after the execution of rule $\mathbf{R B}(u)$ in the $x$ th round, the two next rules executed by $u$ (if any) are necessarily rule_RF followed by rule_C, but rule_C $(u)$ is disabled while $s t_{v}=R B$. Hence, rule_RB $(u)$ is not executed in $\gamma_{i} \mapsto \gamma_{i+1}$, a contradiction.

By Claim 2, no process executes rule_RB during the $n$th round of $e$. Now, along $e$, we have:

- If $\mathbf{P} \_\mathbf{R B}(u)$ holds, then $u$ is enabled (see rule_RB $(u)$ ), and
- If P_RB $(u)$ holds in $\gamma_{i}($ with $i \geq 0)$ and $u$ moves in the next step $\gamma_{i} \mapsto \gamma_{i+1}$, then $\neg \mathbf{P} \_\mathbf{R B}(u)$ holds in $\gamma_{i+1}$.

Indeed, $u$ necessarily executes rule_RB $(u)$ in $\gamma_{i} \mapsto \gamma_{i+1}$ (Remark 2 and Lemma 5) and, consequently, $s t_{u}=R B$ in $\gamma_{i+1}$, which implies that $\neg \mathbf{P} \_\mathbf{R B}(u)$ holds in $\gamma_{i+1}$.

Hence, we can conclude that the last configuration of the $(n-1)$ th round of $e$ satisfies $\mathcal{P}_{2}$, and we are done.

Lemma $14 \mathcal{P}_{3}$ is closed by I o SDR. Moreover, I o SDR converges from $\mathcal{P}_{2}$ to $\mathcal{P}_{3}$ within at most $n$ rounds.
Proof. By Requirement 1, no rule of I can set the status of a process to $R B$. Then, let $\gamma$ be a configuration of I $\circ$ SDR such that $\mathcal{P}_{3}(\gamma)$ holds. Since $\mathcal{P}_{1}(\gamma)$ and $\mathcal{P}_{2}(\gamma)$ also holds, no rule rule_R or rule_RB is enabled in $\gamma$. Hence, after any step from $\gamma$, there is still no process of status $R B$ and we can conclude that $\mathcal{P}_{3}$ is closed by $\mathrm{I} \circ \mathrm{SDR}$ since we already know that $\mathcal{P}_{2}$ is closed by $\mathrm{I} \circ \mathrm{SDR}$ (Lemma 12).

Let $\gamma$ be any configuration satisfying $\mathcal{P}_{2}$ but not $\mathcal{P}_{3}$. To show the convergence from from $\mathcal{P}_{2}$ to $\mathcal{P}_{3}$ within at most $n$ rounds, it is sufficient to show that at least one process $u$ switches from $s t_{u}=R B$ to $s t_{u} \neq R B$
within the next round from $\gamma$, since we already know that once $s t_{u} \neq R B$ after $\gamma, s t_{u} \neq R B$ holds forever (recall that all configurations reached from $\gamma$ satisfies $\mathcal{P}_{2}$; see Lemma 12).

Let $m u$ be a process of status $R B$ with a maximum distance value in $\gamma$. Since $\neg \mathbf{P} \mathbf{U} \mathbf{U p}(m u) \wedge$ $\neg \mathbf{P} \_\mathbf{R B}(m u) \wedge s t_{m u} \neq C$ holds, $\mathbf{P} \_$reset $(m u)$ holds in $\gamma$. Let $v$ be any neighbor of $m u$. Since $\neg \mathbf{P} \_\mathbf{R B}(v) \wedge s t_{m u}=R B$ holds, $s t_{v} \neq C$ in $\gamma$. Again, since $\neg \mathbf{P} \_\mathbf{U p}(v) \wedge \neg \mathbf{P} \_\mathbf{R B}(v) \wedge s t_{v} \neq C$ holds, Preset $(v)$ holds in $\gamma$. According to the definition of $m u$, we have $\left(s t_{v}=R B \wedge d_{v} \leq d_{m u}\right) \vee s t_{v}=R F$. We can conclude that along any execution from $\gamma$, rule $\mathbf{R F}(m u)$ is enabled until $m u$ executes this rule. In this case, rule_RF $(m u)$ will be executed in the next move of $m u$ by Remark 2 and Lemma 5. So, during the next round from $\gamma, \operatorname{rule} \_\mathbf{R F}(m u)$ is executed, i.e., $s t_{m u}$ is set to $R F$, and we are done.

Let $\gamma$ by any configuration of I $\circ$ SDR. We denote by $\gamma_{\mid S D R}$ the projection of $\gamma$ over variables of SDR. By definition, $\gamma_{\mid S D R}$ is a configuration of SDR.

Lemma 15 For every configuration $\gamma$ of $\mathrm{I} \circ \mathrm{SDR}, \gamma \in \mathcal{P}_{4}$ (i.e., $\gamma$ is a normal configuration) if and only if $\gamma_{\mid S D R}$ is a terminal configuration of SDR.
Proof. Let $\gamma$ be a configuration of I o SDR. By definition of $\mathcal{P}_{4}$, if $\gamma \in \mathcal{P}_{4}$, then $\gamma_{\mid S D R}$ is a terminal configuration of SDR. Then, if $\gamma_{\mid S D R}$ is a terminal configuration of $\operatorname{SDR}$, then $\gamma \in \mathcal{P}_{4}$ by Lemmas 1,2 , and 3.

Lemma $16 \mathcal{P}_{4}$ is closed by I o SDR. Moreover, I o SDR converges from $\mathcal{P}_{3}$ to $\mathcal{P}_{4}$ within at most $n$ rounds.
Proof. By Requirement 1, no rule of I can set the status of a process to $R F$. So, by Lemma 15, we can conclude that $\mathcal{P}_{4}$ is closed by $I \circ S D R$.

Let $\gamma$ be any configuration satisfying $\mathcal{P}_{3}$ but not $\mathcal{P}_{4}$. To show the convergence from from $\mathcal{P}_{3}$ to $\mathcal{P}_{4}$ within at most $n$ rounds, it is sufficient to show that at least one process $u$ switches from $s t_{u}=R F$ to $s t_{u} \neq R F$ within the next round from $\gamma$, since we already know that once $s t_{u} \neq R F$ after $\gamma, s t_{u} \neq R F$ holds forever (recall that all configurations reached from $\gamma$ satisfies $\mathcal{P}_{3}$ by Lemma 14, and only process of status $R B$ may switch to status $R F$ ).

Let $m u$ be a process of status $R F$ with a minimum distance value in $\gamma$. Since $\neg \mathbf{P} \mathbf{U} \mathbf{U p}(m u) \wedge \neg \mathbf{P} \_\mathbf{R B}(m u) \wedge$ $s t_{m u} \neq C$ holds, P_reset $(m u)$ holds in $\gamma$. Let $v$ be any neighbor of $m u$. By definition of $\mathcal{P}_{3}, s t_{v} \neq R B$ in $\gamma$. Moreover, since $\neg \mathbf{P} \_\mathbf{U p}(v) \wedge \neg \mathbf{P} \_\mathbf{R B}(v)$ and $v$ has a neighbor of status $R F(m u)$, P_reset $(v)$ holds in $\gamma$. According to the definition of $m u$, we have $\left(s t_{v}=R F \wedge d_{v} \geq d_{m u}\right) \vee s t_{v}=C$. We can conclude that along any execution from $\gamma$, rule_C $(m u)$ is enabled until $m u$ executes this rule. In this case, rule_C $(m u)$ will be executed in the next move of $m u$ by Remark 2 and Lemma 5. So, during the next round from $\gamma$, rule_C $(m u)$ is executed, i.e., $s t_{m u}$ is set to $C$, and we are done.

By Lemmas 11-16 and Theorem 1, follows.
Corollary $5 \mathcal{A}_{4}$ is an attractor for I o SDR. Moreover, I o SDR converges from true to $\mathcal{P}_{4}$ within at most $3 n$ rounds. For every configuration $\gamma$ of $\mathrm{I} \circ \mathrm{SDR}, \gamma$ satisfies $\mathcal{A}_{4}$ (i.e. $\gamma$ is a normal configuration) if and only if P_Clean $(u) \wedge$ P_ICorrect $(u)$ holds in $\gamma$, for every process $u$.

## 5 Asynchronous Unison

### 5.1 The Problem

We now consider the problem of asynchronous unison (introduced in [20]), simply referred to as unison in the following. This problem is a clock synchronization problem: each process $u$ holds a variable (usually
an integer variable) called clock, here noted $c_{u}$. Then, the problem is specified as follows:

- Each process should increment its clock infinitely often. (liveness)
- The difference between clocks of every two neighbors should be at most one increment at each instant. (safety)

Notice that we consider here periodic clocks, i.e., the clock incrementation is modulo a so-called period, here noted $K$.

### 5.2 Related Work

The first self-stabilizing asynchronous unison for general connected graphs has been proposed by Couvreur et al. [20]. It is written in the locally shared memory model with composite atomicity assuming a central unfair daemon and a period $K>n^{2}$. No complexity analysis was given. Another solution which stabilizes in $O(n)$ rounds has been proposed by Boulinier et al. in [11]. This solution is also written in the locally shared memory model with composite atomicity, however it assumes a distributed unfair daemon. In this solution, the period $K$ should satisfy $K>C_{G}$ and another parameter $\alpha$ should satisfy $\alpha \geq T_{G}-2 . C_{G}$ is the cyclomatic characteristic of the network and $T_{G}$ is the length of the longest chordless cycle. Boulinier also proposed in his PhD thesis [10] a parametric solution which generalizes both the solutions of [20] and [11]. In particular, the study of this parametric algorithm reveals that the solution of Couvreur et al. [20] still works assuming a distributed unfair daemon and has a stabilization time in $O(D . n)$ rounds, where $D$ is the network diameter.

### 5.3 Contribution

We first propose a distributed algorithm, called U . Starting from a pre-defined configuration, U implements the unison problem in anonymous networks, providing that the period $K$ satisfies $K>n$. U is not selfstabilizing, however we show that the composite algorithm $U \circ S D R$ is actually an efficient self-stabilizing unison algorithm. Indeed, its stabilization times in round matches the one of the best existing solution [11]. Moreover, it achieves a better stabilization time in moves, since it stabilizes in $O\left(D . n^{2}\right)$ moves, while the algorithm in [11] stabilizes in $O\left(D \cdot n^{3}+\alpha . n^{2}\right)$ moves; as shown in [23].

### 5.4 Algorithm $U$

Overview. We consider here anonymous (bidirectional) networks of arbitrary connected topology. Moreover, every process has the period $K$ as input. $K$ is required to be (strictly) greater than $n$, the number of processes. The formal code of Algorithm $U$ is given in Algorithm 2. Informally, each process maintains a single variable, its clock $c_{u}$, using a single rule rule_ $\mathbf{U}(u)$.

In the following, we assume that the system is initially in the configuration $\gamma_{\text {init }}$ where every process $u$ satisfies $c_{u}=0 \wedge s t_{u}=C$. Basically, starting from $\gamma_{\text {init }}$, a process $u$ can increment its clock $c_{u}$ modulo $K$ (using rule rule_U $\mathbf{U}(u)$ ) if it is on time or one increment late with each of its neighbors; see predicate P_Up(u).

Correctness. Below, we focus on configurations of U satisfying P_ICorrect $(u) \wedge$ P_Clean $(u)$, for every process $u$. Indeed, $\gamma_{\text {init }}$ belongs to this class of configurations. Moreover, a configuration of $U \circ S D R$

```
Algorithm 2 Algorithm U, code for every process \(u\)
Inputs:
    - \(\quad s t_{u} \in\{C, R B, R F\} \quad: \quad\) variable of SDR
    - P_Clean \((u)\) : predicate of SDR
    - \(K \quad:\) a constant from the system satisfying \(K>n\)
```


## Variables:

- $\quad c_{u} \in \mathbb{N} \quad: \quad$ the clock of $u$


## Predicates:

- P_Ok $(u, v) \equiv c_{v} \in\left\{\left(c_{u}-1\right) \% K, c_{u},\left(c_{u}+1\right) \% K\right\}$
- P_ICorrect $(u) \equiv\left(\forall v \in N(u), \mathbf{P} \_\mathbf{O k}(u, v)\right)$
- P_reset $(u) \equiv c_{u}=0$
- $\mathbf{P}_{-} \mathbf{U p}(u) \quad \equiv\left(\forall v \in N(u), c_{v} \in\left\{c_{u},\left(c_{u}+1\right) \% K\right\}\right)$


## Macros:

- $\operatorname{reset}(u) \quad: \quad c_{u}:=0$;


## Rules:

```
rule_U(u) : P_Clean (u)^P_Upp}(u)\quad->\quad\mp@subsup{c}{u}{}:=(\mp@subsup{c}{u}{}+1)%K
```

is normal if and only if $\mathbf{P}_{\_} \operatorname{Clean}(u) \wedge \mathbf{P}$ ICorrect $(u)$ holds for every process $u$. Hence, the properties we exhibit now will be, in particular, satisfied at the completion of SDR.

Consider any two neighboring processes $u$ and $v$ such that $\mathbf{P}_{-} \mathbf{O k}(u, v)$ in some configuration $\gamma$, i.e., $c_{v} \in\left\{\left(c_{u}-1\right) \% K, c_{u},\left(c_{u}+1\right) \% K\right\}$. Without the loss of generality, assume that $c_{v} \in\left\{c_{u},\left(c_{u}+1\right) \% K\right\}$ (otherwise switch the role of $u$ and $v$ ). Let $\gamma \mapsto \gamma^{\prime}$ be the next step. If $c_{v}=c_{u}$ in $\gamma$, then $c_{v} \in\left\{\left(c_{u}-\right.\right.$ 1) $\left.\% K, c_{u},\left(c_{u}+1\right) \% K\right\}$ in $\gamma^{\prime}$ since each clock increments at most once per step, i.e., $\mathbf{P} \_\mathbf{O k}(u, v)$ still holds in $\gamma^{\prime}$. Otherwise, $\left.c_{v}=\left(c_{u}+1\right) \% K\right\}$ in $\gamma$, and so $v$ is disabled and only $u$ may move. If $u$ does not move, then $\left.c_{v}=\left(c_{u}+1\right) \% K\right\}$ still holds in $\gamma^{\prime}$, otherwise $c_{v}=c_{u}$ in $\gamma^{\prime}$. Hence, in both cases, $\mathbf{P} \_\mathbf{O k}(u, v)$ still holds in $\gamma^{\prime}$. Hence, follows.

Lemma 17 P_ICorrect $(u)$ is closed by U , for every process $u$.
Since Algorithm U does not modify any variable from Algorithm SDR, we have
Remark 7 P_Clean $(u)$ is closed by U, for every process $u$.
Corollary 6 P_ICorrect $(u) \wedge \mathbf{P}_{-}$Clean $(u)$ is closed by U, for every process $u$.
Corollary 7 Any execution of U , that starts from a configuration where $\mathbf{P}$ _ICorrect $(u) \wedge \mathbf{P}$ _Clean $(u)$ holds for every process $u$, satisfies the safety of the unison problem.

Lemma 18 Any configuration where $\mathbf{P} \_\mathbf{I C o r r e c t}(u) \wedge \mathbf{P} \_$Clean $(u)$ holds for every process $u$ is not terminal.

Proof. Assume, by the contradiction, a terminal configuration $\gamma$ where $\mathbf{P}_{\_} \operatorname{ICorrect}(u) \wedge \mathbf{P}_{-} \operatorname{Clean}(u)$ for every process $u$. Then, every process $u$ has at least one neighbor $v$ such that $c_{v}=\left(c_{u}-1\right) \% K$. Since the number of processes is finite, in $\gamma$ there exist elementary cycles $u_{1}, \ldots, u_{x}$ such that

1. for every $i \in\{1, \ldots, x-1\}, u_{i}$ and $u_{i+1}$ are neighbors, and $c_{u_{i}}=\left(c_{u_{i+1}}-1\right) \% K$; and
2. $u_{1}$ and $u_{x}$ are neighbors and $c_{u_{x}}=\left(c_{u_{1}}-1\right) \% K$.

By transitivity, Case 1 implies that $c_{u_{1}}=\left(c_{u_{x}}-(x-1)\right) \% K$. So, from Case 2, we obtain $c_{u_{x}}=\left(c_{u_{x}}-\right.$ $x) \% K$. Now, by definition $x \leq n$ and $K>n$ so $c_{u_{x}} \neq\left(c_{u_{x}}-x\right) \% K$, a contradiction. Hence, $\gamma$ is not terminal.

Lemma 19 Any execution of U , that starts from a configuration where $\mathbf{P}$ _ICorrect $(u) \wedge$ P_Clean $(u)$ holds for every process $u$, satisfies the liveness of the unison problem.

Proof. Let $e$ be any execution of $U$ that starts from a configuration where P_ICorrect $(u) \wedge$ P_Clean $(u)$ holds for every process $u$. Assume, by the contradiction, that $e$ does not satisfy the liveness of unison. Then, $e$ contains a configuration $\gamma$ from which some processes (at least one) never more executes rule_U. Let $F$ be the non-empty subset of processes that no more move from $\gamma$. Let $I=V \backslash F$. By Lemma $18, I$ is not empty too. Now, since the network is connected, there are two processes $u$ and $v$ such that $u \in I$ and $v \in F$. Now, after at most 3 increments of $u$ from $\gamma, \mathbf{P} \_\mathbf{O k}(u, v)$ no more holds, contradicting Lemma 17.

Consider now any execution $e$ of $U$ starting from $\gamma_{\text {init }}$. In $\gamma_{\text {init }}$, we have $\mathbf{P}$ _Clean $(u) \wedge \mathbf{P}$ _ICorrect $(u)$ for every process $u$. Hence, by Corollary 7 and Lemma 19, follows.

Theorem 5 U is distributed (non self-stabilizing) unison.

Properties of U . Consider any execution $e$ of U starting from a configuration $\gamma$ which does not satisfy P_Clean $(u) \wedge$ P_ICorrect $(u)$ for every process $u$. Then, there exists at least one process $u$ satisfying $\neg \mathbf{P}$ _Clean $(u) \vee \neg \mathbf{P}$ _ICorrect $(u)$ in $\gamma$, and $u$ is disabled forever in $e$. Indeed, if $\neg \mathbf{P}$ _Clean $(u)$, then $\neg \mathbf{P}$ _Clean $(u)$ holds forever since $U$ does not write into SDR's variables. If $\neg \mathbf{P}$ _ICorrect $(u)$ holds, then there is a neighbor $v$ such that $\neg \mathbf{P} \_\mathbf{O k}(u, v)$ holds, both $u$ and $v$ are disabled, hence so $\neg \mathbf{P} \_\mathbf{O k}(u, v)$ holds forever, which implies that $\neg \mathbf{P} \_\mathbf{U p}(u)$ forever. Now, since $u$ is disabled forever, each neighbor of $u$ moves at most three times in $e$. Inductively, every node at distance $d$ from $u$ moves at most $3 d$ times. Overall, we obtain the following lemma.

Lemma 20 In any execution of U starting from a configuration which does not satisfy P_Clean $(u) \wedge$ P_ICorrect ( $u$ ) for every process $u$, each process moves at most $3 D$ times, where $D$ is the network diameter.

### 5.5 Algorithm $\mathrm{U} \circ \mathrm{SDR}$

Requirements. To show the self-stabilization of $U \circ S D R$, we should first establish that $U$ meets the requirements 1 to 2d, given in Subsection 3.5.

Requirement 2a is satisfied since P_ICorrect $(u)$ does not involve any variable of SDR and is closed by U (Lemma 17). All other requirements directly follow from the code of U .

Self-stabilization and Move Complexity. We define the legitimate configurations of $U \circ S D R$ as the set of configurations satisfying P_Clean $(u) \wedge$ P_ICorrect $(u)$ for every process $u$. This set actually corresponds to the set of normal configurations (see Corollary 5, page 20) and is closed by Algorithm U $\circ$ SDR, by

Remark 2 (page 11), Theorem 1 (page 10), and Corollary 6. Then, from any normal configuration, the specification of the unison holds, by Corollary 7 and Lemma 19. So, it remains to show the convergence.

Let $u$ be a process. Let $e$ be an execution of $\mathrm{U} \circ$ SDR. By Corollary 3 (page 16), the sequence of rules executed by a process $u$ in a segment of $e$ belongs to the following language:

$$
(\text { rule_C }+\varepsilon) \text { words_I }(\text { rule_RB }+ \text { rule_R }+\varepsilon)(\text { rule_RF }+\varepsilon)
$$

where words_I be any sequence of rules of $U$.
Let assume that $e$ contains $s$ segments. Recall that $s \leq n+1$; see Remark 5, page 13. Let call regular segment any segment that starts in a configuration containing at least one abnormal alive root. A regular segment contains no normal configuration, so, by Lemma 10 (page 17) and Lemma 20, the sequence words_I of $u$ is bounded by $3 D$ in $S$. Thus, $u$ executes at most $3 D+3$ moves in $S$ and, overall a regular segment contains at most $(3 D+3) . n$ moves and necessarily ends by a step where the number of abnormal alive root decreases. Hence, all $s-1$ first segments are regular and the last one is not. Overall, the last segment $S_{\text {last }}$ starts after at most $(3 D+3) . n .(s-1)$ moves. $S_{\text {last }}$ contains no abnormal alive root and so, the sequence of rules executed by $u$ in $S_{\text {last }}$ belongs to the following language: (rule_C $+\varepsilon$ ) words_I. ${ }^{2}$ If the initial configuration of $S_{\text {last }}$ contains no process of status $R F$, then it is a normal configuration and so $s=1$, i.e., $e$ is initially in a normal configuration. Otherwise, let $v$ be a process such that $s t_{v}=R F$ in the initial configuration of $S_{\text {last }}$ and no other process executes rule_C later than $v$. Following the same reasoning as in Lemma 20, while $v$ does not execute rule_C, each process other than $v$ can execute at most $3 D$ rules of $U$ and one rule_C. Hence, there are at most $(3 D+1) \cdot(n-1)+1$ moves in $S_{\text {last }}$ before the system reaches a normal configuration.

Since, in the worst case $s=n+1$, overall $e$ reaches a normal configuration in at most $(3 D+3) \cdot n^{2}+$ $(3 D+1) \cdot(n-1)+1$ moves, and we have the following theorem.

Theorem $6 \mathrm{U} \circ$ SDR is self-stabilizing for the unison problem. Its stabilization time is in $O\left(D . n^{2}\right)$ moves.
Round Complexity. By Corollary 5 (page 20), follows.
Theorem 7 The stabilization time of $\mathrm{U} \circ \mathrm{SDR}$ is at most $3 n$ rounds.

## $6(f, g)$-alliance

### 6.1 The Problem

The $(f, g)$-alliance problem has been defined by Dourado et al. [29]. Given a graph $G=(V, E)$, and two non-negative integer-valued functions on nodes $f$ and $g$, a subset of nodes $A \subseteq V$ is an $(f, g)$-alliance of $G$ if and only if every node $u \notin A$ has at least $f(u)$ neighbors in $A$, and every node $v \in A$ has at least $g(v)$ neighbors in $A$. The $(f, g)$-alliance problem is the problem of finding a subset of processes forming an $(f, g)$-alliance of the network. The $(f, g)$-alliance problem is a generalization of several problems that are of interest in distributed computing. Indeed, consider any subset $S$ of processes/nodes:

1. $S$ is a domination set [ 9$]$ if and only if $S$ is a $(1,0)$-alliance;
2. more generally, $S$ is a $k$-domination set [9] if and only if $S$ is a $(k, 0)$-alliance;

[^2]3. $S$ is a $k$-tuple dominating set [38] if and only if $S$ is a $(k, k-1)$-alliance;
4. $S$ is a global offensive alliance [40] if and only if $S$ is a $(f, 0)$-alliance, where $f(u)=\left\lceil\frac{\delta_{u}+1}{2}\right\rceil$ for all $u$;
5. $S$ is a global defensive alliance [41] if and only if $S$ is a $(1, g)$-alliance, where $g(u)=\left\lceil\frac{\delta_{u}+1}{2}\right\rceil$ for all $u$;
6. $S$ is a global powerful alliance [46] if and only if $S$ is a $(f, g)$-alliance, such that $f(u)=\left\lceil\frac{\delta_{u}+1}{2}\right\rceil$ and $g(u)=\left\lceil\frac{\delta_{u}}{2}\right\rceil$ for all $u$.

We remark that $(f, g)$-alliances have applications in the fields of population protocols [3] and server allocation in computer networks [32].

Ideally, we would like to find a minimum $(f, g)$-alliance, namely an $(f, g)$-alliance of the smallest possible cardinality. However, this problem is $\mathcal{N} \mathcal{P}$-hard, since the ( 1,0 )-alliance (i.e., the domination set problem) is known to be $\mathcal{N} \mathcal{P}$-hard [30]. We can instead consider the problem of finding a minimal $(f, g)$ alliance. An $(f, g)$-alliance is minimal if no proper subset of $A$ is an $(f, g)$-alliance. Another variant is the 1-minimal $(f, g)$-alliance. $A$ is a 1-minimal $(f, g)$-alliance if deletion of just one member of $A$ causes $A$ to be no more an $(f, g)$-alliance, i.e., $A$ is an $(f, g)$-alliance but $\forall u \in A, A \backslash\{u\}$ is not an $(f, g)$-alliance. Surprisingly, a 1-minimal $(f, g)$-alliance is not necessarily a minimal $(f, g)$-alliance [29]. However, we have the following property:

## Property 1 (Dourado et al. [29]) Given two non-negative integer-valued functions $f$ and $g$ on nodes

1. Every minimal $(f, g)$-alliance is a 1-minimal $(f, g)$-alliance, and
2. if $f(u) \geq g(u)$ for every process $u$, then every 1-minimal $(f, g)$-alliance is a minimal $(f, g)$-alliance.

### 6.2 Contribution

We first propose a distributed algorithm called FGA. Starting from a pre-defined configuration, FGA computes a 1-minimal $(f, g)$-alliance in any identified network where $\delta_{u} \geq \max (f(u), g(u))$, for every process $u$. Notice that this latter assumption ensures the existence of a solution. FGA is not self-stabilizing, however we show that the composite algorithm FGA $\circ$ SDR is actually an efficient self-stabilizing 1-minimal $(f, g)$-alliance algorithm.

### 6.3 Related Work

Recall that the $(f, g)$-alliance problem has been introduced by Dourado et al. [29]. In that paper, the authors give several distributed algorithms for that problem and its variants, but none of them is self-stabilizing.

In [16], Carrier et al. proposes a silent self-stabilizing algorithm that computes a minimal $(f, g)$-alliance in an asynchronous network with unique node IDs, assuming that every node $u$ has a degree at least $g(u)$ and satisfies $f(u) \geq g(u)$. Their algorithm is also safely converging in the sense that starting from any configuration, it first converges to a (not necessarily minimal) $(f, g)$-alliance in at most four rounds, and then continues to converge to a minimal one in at most $5 n+4$ additional rounds, where $n$ is the size of the network. The algorithm is written in the locally shared memory model with composite atomicity. It is proven assuming a distributed unfair daemon and takes $O\left(n \cdot \Delta^{3}\right)$ moves to stabilize, where $\Delta$ is the degree of the network.

There are several other self-stabilizing solutions for particular instances of $(f, g)$-alliances proposed in the locally shared memory model with composite atomicity, e.g., [25, 35, 43, 44, 45, 46].

Algorithms given in [43, 45] work in anonymous networks, however, they both assume a central daemon. More precisely, Srimani and Xu [43] give several algorithms which compute minimal global offensive and 1-minimal defensive alliances in $O\left(n^{3}\right)$ moves. Wang et al. [45] give a self-stabilizing algorithm to compute a minimal $k$-dominating set in $O\left(n^{2}\right)$ moves.

All other solutions [25, 35, 44, 46] consider arbitrary identified networks. Turau [44] gives a selfstabilizing algorithm to compute a minimal dominating set in $9 n$ moves, assuming a distributed unfair daemon. Yahiaoui et al. [46] give self-stabilizing algorithms to compute a minimal global powerful alliance. Their solution assumes a distributed unfair daemon and stabilizes in $O(n \cdot m)$ moves, where $m$ is the number of edges in the network.

A safely converging self-stabilizing algorithm is given in [35] for computing a minimal dominating set. The algorithm first computes a (not necessarily minimal) dominating set in $O(1)$ rounds and then safely stabilizes to a minimal dominating set in $O(D)$ rounds, where $D$ is the diameter of the network. However, a synchronous daemon is required. A safely converging self-stabilizing algorithm for computing minimal global offensive alliances is given in [25]. This algorithm also assumes a synchronous daemon. It first computes a (not necessarily minimal) global offensive alliance within two rounds, and then safely stabilizes to a minimal global offensive alliance within $O(n)$ additional rounds.

To the best of our knowledge, until now there was no self-stabilizing algorithm solving the 1-minimal $(f, g)$-alliance without any restriction on $f$ and $g$.

### 6.4 Algorithm FGA

Overview. Recall that we consider any network where $\delta_{u} \geq \max (f(u), g(u))$, for every process $u$. Moreover, we assume that the network is identified, meaning that each process $u$ can be distinguished using a unique constant identifier, here noted $i d_{u}$. The formal code of FGA is given in Algorithm 3. Informally, each process $u$ maintains the following four variables.
$c o l_{u}$ : a Boolean variable, the output of FGA. Process $u$ belongs to the $(f, g)$-alliance if and only if $\operatorname{col}_{u}$.
$s c r_{u}$ : a variable, whose domain is $\{-1,0,1\} . s c r_{u} \leq 0$ if and only if no $u$ 's neighbor can quit the alliance. $\operatorname{can} Q_{u}:$ a Boolean variable. $\neg \operatorname{can} Q_{u}$ if $u$ cannot quit the alliance (in particular, if $u$ is out of the alliance).
$p t r_{u}$ : a pointer variable, whose domain is $N[u] \cup\{\perp\}$. Either $p t r_{u}=\perp$ or $p t r_{u}$ designates the member of its closed neighborhood of smallest identifier such that $\operatorname{can} Q_{u}$.

In the following, we assume that the system is initially in the configuration $\gamma_{\text {init }}$ where every process $u$ has the following local state:

$$
\operatorname{col}_{u}=\operatorname{true}, \mathrm{scr}_{u}=1, \operatorname{can}_{u}=\operatorname{true},^{\mathrm{ptr}}{ }_{u}=\perp, s t_{u}=C .
$$

In particular, this means that all processes are initially in the alliance. Then, the idea of the algorithm is reduced the alliance until obtaining a 1 -minimal $(f, g)$-alliance. A process $u$ leaves the alliance by executing rule_Clr$(u)$. To leave the alliance, $u$ should have enough neighbors in the alliance ( $\# \operatorname{In} \operatorname{All}(u) \geq f(u)$ ), approve itself, and have a full approval from all neighbors. Process $v$ approves $u$ if $p r_{v}=u$. Moreover, the approval of $v$ is full if $s c r_{v}=1$. Notice that, the $p t r$ pointers ensure that removals from the alliance are locally central: in the closed neighborhood of any process, at most one process leaves the alliance at each step.

## Algorithm 3 Algorithm FGA, code for every process $u$

## Inputs:

- $s t_{u} \in\{C, R B, R F\} \quad$ : variable of SDR
- P_Clean $(u)$ : predicate of SDR
- $i d_{u} \quad: \quad$ identifier of $u$, constant from the system


## Variables:

- col $_{u}$ : Boolean
- $\operatorname{scr}_{u} \in\{-1,0,1\} \quad: \quad$ The score of $u$
- $\operatorname{can} Q_{u}$ : Boolean
- $\operatorname{ptr}_{u} \in N[u] \cup\{\perp\}$ : Closed Neighborhood Pointer


## Predicates:

- P_ICorrect $(u) \equiv \operatorname{realScr}(u) \geq 0 \wedge$
$\left[\left(s c r_{u}=\operatorname{realScr}(u)=1\right) \vee p t r_{u}=\perp \mathrm{V}\right.$ $\left.\left(p \operatorname{tr}_{u} \neq \perp \wedge \operatorname{scr}_{u}=1 \wedge \neg \operatorname{col}_{p t r_{u}}\right)\right]$
- P_reset $(u) \equiv \operatorname{col}_{u} \wedge p t r_{u}=\perp \wedge \operatorname{can}_{u} \wedge s c r_{u}=1$
- P_canQuit $(u) \equiv \operatorname{col}_{u} \wedge \# \operatorname{InAll}(u) \geq f(u) \wedge\left(\forall v \in N(u), s c r_{v}=1\right)$
- P_toQuit $(u) \equiv$ P_canQuit $(u) \wedge\left(\forall v \in N[u], p t r_{v}=u\right)$
- P_updPtr $(u) \equiv \neg \mathbf{P}$ _toQuit $(u) \wedge p t r_{u} \neq \operatorname{bestPtr}(u)$


## Macros:

- $\operatorname{reset}(u) \quad: \operatorname{col}_{u}:=$ true $^{\prime} \operatorname{ptr}_{u}:=\perp ; \operatorname{can}_{u}:=\operatorname{true}^{\prime} \operatorname{scr}_{u}:=1$;
- \# InAll $(u):\left|\left\{w \in N(u) \mid \operatorname{col}_{w}\right\}\right|$

| realScr $(u)=-1$ |  | $\# \operatorname{InAll}(u)<f(u)$ | $\wedge$ | $\neg \mathrm{col}$ |
| :---: | :---: | :---: | :---: | :---: |
| $\operatorname{realScr}(u)=-1$ | if | \# In All $(u)<g(u)$ | $\wedge$ |  |
| $\operatorname{realScr}(u)=0$ | if | $\# \operatorname{InAll}(u)=f(u)$ | $\wedge$ |  |
| $\operatorname{alScr}(u)=0$ | if | $\# \operatorname{InAll}(u)=g(u)$ | $\wedge$ |  |
| $\operatorname{Scr}(u)=1$ | if | $\because \operatorname{InAll}(u)>f(u)$ | $\wedge$ |  |
| rea |  | u) |  |  |

- cmpVar(u) : scr $r_{u}:=$ realScr $(u) ;$ can $Q_{u}:=\mathbf{P}_{\text {_canQuit }}(u)$;
- bestPtr $(u) \quad$ : if $\left(s c r_{u} \leq 0\right)$ return $\perp$;
if $\left(\forall v \in N[u], \neg \operatorname{can} Q_{u}\right)$ then return $\perp$;
else return $\operatorname{argmin}_{\left(v \in N[u] \mid \operatorname{can} Q_{u}\right)}\left(i d_{u}\right)$;
- upd(u) : cmpVar(u); ptr ${ }_{u}:=\operatorname{bestPtr}(u)$;

```
Rules:
rule_Clr \((u)\) : P_Clean \((u) \wedge\) P_ICorrect \((u) \wedge \quad \rightarrow \quad\) col \(u_{u}:=\) false;
    P_toQuit (u)
rule_P1 \((u) \quad: \quad\) P_Clean \((u) \wedge \mathbf{P}\) _ICorrect \((u) \wedge\)
    \(\mathbf{P}_{-} \operatorname{updP} \operatorname{tr}(u) \wedge p t r_{u} \neq \perp\)
\(\rightarrow \quad p t r_{u}:=\perp ;\)
    cmpVar (u);
rule_P2 \((u): \quad\) P_Clean \((u) \wedge\) P_ICorrect \((u) \wedge \quad \rightarrow \quad u p d(u)\);
    P_updPtr \((u) \wedge p t r_{u}=\perp\)
rule_Q \((u): \mathbf{P}_{2} \operatorname{Clean}(u) \wedge \mathbf{P}\) _ICorrect \((u) \wedge \quad \rightarrow \quad \operatorname{cmp} \operatorname{Var}(u)\);
    \(\neg \mathbf{P}\) _toQuit \((u) \wedge \neg \mathbf{P}\) _updPtr \((u) \wedge \quad\) if \(\operatorname{realScr}(u) \leq 0\) then
    \(\left(s c r_{u} \neq \operatorname{realScr}(u) \vee \operatorname{can} Q_{u} \neq \mathbf{P}_{\text {_canQuit }}(u)\right) \quad \quad \operatorname{ptr}_{u}=\perp\);
```

To ensure the liveness of the algorithm, a process $u$ gives its approval (by executing rule_P2 $(u)$, maybe preceded by rule_P1 $(u)$ ) to the member of its closed neighborhood having the smallest identifier among the ones requiring an approval (i.e., the processes satisfying $\operatorname{can} Q$ ).

To ensure that $\operatorname{realScr}(v) \geq 0$ is a closed predicate, a process $v$ gives its approval to another process $u$ only if $\operatorname{realScr}(v)=1$ and none of its neighbor can leave the alliance (i.e., $p t r_{u} \notin N(v)$ ). This latter condition ensures that no neighbor of $v$ leaves the alliance simultaneously to a new approval of $v$. It is mandatory since otherwise the cause for which $v$ gives its approval may be immediately outdated. Hence, any approval switching is done either in one step when the process leaves the alliance, or in two atomic steps where $p t r_{v}$ first takes the value $\perp$ (rule rule_P1(u)) and then points to the suitable process, (rule rule_P2(u)).

Finally, the rule rule_Q $(u)$ refreshes the values of $s c r_{u}, \operatorname{ptr}_{u}$, and $\operatorname{can} Q_{u}$ after a neighbor left the alliance or updated its $s c r$ variable.

Properties of FGA. Below, we show some properties of Algorithm FGA that will be used for showing both its correctness and the self-stabilization of its composition with Algorithm SDR.

First, by checking the rules of FGA, we can remark that each time a process $u$ sets $s c r_{u}$ to a value other than 1 , it also sets $p t r_{u}$ to $\perp$, in the same step. Hence, by construction we have the following lemma.

Lemma $21 \operatorname{scr}_{u}=1 \vee p t r_{u}=\perp$ is closed by FGA, for every process $u$.
Since Algorithm FGA does not modify any variable from Algorithm SDR, we have
Remark 8 P_Clean $(u)$ is closed by FGA, for every process $u$.
Lemma 22 Let $u$ be any process. Let $\gamma$ be a configuration where P_Clean $(u) \wedge$ P_ICorrect $(u)$ holds. Let $\gamma^{\prime}$ be any configuration such that $\gamma \mapsto \gamma^{\prime}$. In $\gamma^{\prime}$, $\operatorname{realScr}(u) \geq 0$.

Proof. By definition of $\mathbf{P}$ _toQuit, at most one process of $N[u]$ executes rule_Clr in $\gamma \mapsto \gamma^{\prime}$. If no process of $N[u]$ executes rule_Clr, we are done. If rule_Clr $(u)$ is executed, then $\# \operatorname{InAll}(u) \geq f(u)$ in $\gamma$ (see P_canQuit $(u)$ ) and so \# $\operatorname{InAll}(u) \geq f(u)$ in $\gamma^{\prime}$ too and thus $\operatorname{realScr}(u) \geq 0$ in $\gamma^{\prime}$. Otherwise, let $v \in N(u)$ such that rule_Clr $(v)$ is executed in $\gamma \mapsto \gamma^{\prime}$. In $\gamma, c_{v}=$ true and P_toQuit $(v)$ holds with, in particular, $p t r_{u}=v \neq \perp$, i.e., col $_{p t r_{u}}$ holds. Hence, P_ICorrect $(u), p t r_{u} \neq \perp$, and col $_{p t r_{u}}$ imply $\operatorname{realScr}(u)=1$ in $\gamma$, and so realScr $(u) \geq 0$ in $\gamma^{\prime}$.

By definition of FGA, we have
Remark 9 Let $u$ be any process. Let $\gamma$ be a configuration where $p t r_{u}=v$ with $v \neq u$. Let $\gamma^{\prime}$ be any configuration such that $\gamma \mapsto \gamma^{\prime}$. In $\gamma^{\prime}$, we have $\operatorname{ptr}_{u} \in\{v, \perp\}$.

Lemma 23 Let u be any process. Let $\gamma$ be a configuration where $\mathbf{P}$ _Clean $(u) \wedge \mathbf{P}$ _ICorrect $(u)$ holds. Let $\gamma^{\prime}$ be any configuration such that $\gamma \mapsto \gamma^{\prime}$ is a step where $v \in N(u)$ executes $\mathbf{r u l e}$ _Clr $(v)$. P_ICorrect $(u)$ holds in $\gamma^{\prime}$

Proof. Since rule_Clr$(v)$ is enabled in $\gamma$, we have $p t r_{u}=v$ and $s c r_{u}=1$. So, if $u$ does not move in $\gamma \mapsto \gamma^{\prime}$, we have $p t r_{u} \neq \perp$, scr $_{u}=1$, and $\neg$ col $_{\text {ptr }}^{u}$. Hence, P_ICorrect $(u)$ holds in $\gamma^{\prime}$ by Lemma 22.

Otherwise, either rule_P1 $(u)$ or rule_Q $(u)$ is executed in $\gamma \mapsto \gamma^{\prime}$. In the former case, $p t r_{u}=\perp$ in $\gamma^{\prime}$ and by Lemma 22, P_ICorrect $(u)$ holds in $\gamma^{\prime}$. In the latter case, in $\gamma^{\prime}$ either $p t r_{u}=v \neq \perp, s c r_{u}=1$ (by Lemma 21), and $\neg$ col $_{p t r_{u}}$, or $p t r_{u}=\perp$. In either case, P_ICorrect ( $u$ ) holds in $\gamma^{\prime}$ by Lemma 22, and we are done.

Lemma 24 Let $u$ be any process. Let $\gamma$ be a configuration where P_Clean $(u) \wedge$ P_ICorrect $(u)$ holds. Let $\gamma^{\prime}$ be any configuration such that $\gamma \mapsto \gamma^{\prime}$ is a step where $u$ executes an action and none of its neighbor executes rule_Clr. P_ICorrect $(u)$ holds in $\gamma^{\prime}$.

Proof. Since no neighbor of $u$ executes rule_Clr in $\gamma \mapsto \gamma^{\prime}$, we have $s c r_{u}=\operatorname{realScr}(u)$ in $\gamma^{\prime}$, and by Lemmas 21 and 22, we are done.

By Remark 8 and Lemmas 23-24, follows.
Corollary 8 P_Clean $(u) \wedge$ P_ICorrect $(u)$ is closed by FGA, for every process $u$.
Since $u$ is disabled in FGA if $\neg \mathbf{P}$ _Clean $(u)$ holds. $\neg \mathbf{P}$ _Clean $(u) \wedge \mathbf{P}$ _ICorrect $(u)$ is also closed by FGA, for every process $u$. Hence, follows.

Corollary 9 P_ICorrect $(u)$ is closed by FGA, for every process $u$.

Partial Correctness. As for Algorithm U, we focus on configurations of FGA satisfying P_ICorrect $(u) \wedge$ P_Clean $(u)$, every process $u$. Indeed, again, $\gamma_{\text {init }}$ belongs to this class of configurations, and a configuration of FGA $\circ$ SDR is normal if and only if $\mathbf{P} \_$Clean $(u) \wedge \mathbf{P}$ _ICorrect $(u)$ holds for every process $u$. So, the properties we exhibit now will be, in particular, satisfied at the completion of SDR.

Consider any terminal configuration of FGA where P_Clean $(u) \wedge$ P_ICorrect $(u)$ holds for every process $u$. By checking the code of Algorithm FGA, we can remark that every process $u$ satisfies

$$
\begin{gathered}
\operatorname{scr}_{u}=\text { realScr }(u) \wedge \operatorname{can} Q_{u}=\mathbf{P}_{u} \text { canQuit }(u) \wedge \\
\operatorname{ptr} r_{u}=\operatorname{bestPtr}(u) \wedge \neg \mathbf{P} \operatorname{toQuit}(u) .
\end{gathered}
$$

Based on this, one can easily establish that in such a terminal configuration, the set $A=\left\{u \in V \mid \operatorname{col}_{u}\right\}$ is a 1-minimal $(f, g)$ alliance of the network. Indeed, for every process $u$, since P_ICorrect $(u)$ holds, $\operatorname{realScr}(u) \geq 0$, which in turn implies that $A$ is an $(f, g)$ alliance. Assume then, by the contradiction, that $A$ is not 1-minimal in some terminal configuration of FGA where P_Clean $(u) \wedge \mathbf{P}$ ICorrect $(u)$ holds for every process $u$. Let $m$ be the process of minimum identifier such that $A-\{m\}$ is an $(f, g)$ alliance. First, by definition, $\operatorname{col}_{m} \wedge \# \operatorname{InAll}(m) \geq f(m)$ holds. Then, $\forall u \in N[m], s c r_{u}=\operatorname{realScr}(u)=1$ since $A-\{m\}$ is an $(f, g)$ alliance. So, P_canQuit $(m)$ holds, which implies that $c a n Q_{m}=$ true. Finally, by minimality of the $m$ 's identifier, $\forall u \in N[m], \operatorname{ptr}_{u}=\operatorname{bestPtr}(u)=m$. Hence, P_toQuit $(m)$ holds, which in turn implies that rule_Clr $(m)$ is enabled, a contradiction. Hence, follows.

Theorem 8 In any terminal configuration of FGA where P_Clean $(u) \wedge$ P_ICorrect $(u)$ holds for every process $u$, the set $A=\left\{u \in V \mid\right.$ col $\left._{u}\right\}$ is a 1-minimal $(f, g)$ alliance of the network.

Consider now any execution $e$ of FGA starting from $\gamma_{\text {init }}$. In $\gamma_{\text {init }}$, we have P_Clean $(u) \wedge$ P_ICorrect $(u)$ for every process $u$. Then, P_Clean $(u) \wedge \mathbf{P}$ _ICorrect $(u)$, for every process $u$, is invariant in $e$, by Corollary 8 . Hence, we have the following corollary.

Corollary 10 Let e be any execution of FGA starting from $\gamma_{\text {init }}$. If e terminates, then the set $A=\{u \in$ $V \mid$ col $\left._{u}\right\}$ is a 1-minimal $(f, g)$ alliance of the network in the terminal configuration of $e$.

Termination. We now show that any execution of FGA (starting from any arbitrary configuration) eventually terminates. Let $v$ be any process. Let $e$ be any execution of FGA. First, rule_Clr $(v)$ switches colv from true to false and no rule of FGA sets $\mathrm{col}_{v}$ from false to true. So,
(1) $\operatorname{rule}$ _ $\mathbf{C l r}(v)$ is executed at most once in $e$.

Moreover, this implies that
(2) the value of the macro $\# \operatorname{In} \operatorname{All}(v)$ is monotonically non-increasing in $e$.

If $\operatorname{realScr}(u)<0$ holds for some process $u$ in some configuration of $e$, then $\mathbf{P}_{\text {_IC }}$ Correct $(u)$ does not hold and so $u$ is disabled. Moreover, by (2), $u$ is disabled forever in $e$. Assume, otherwise, that realScr $(u) \geq 0$. Then, realScr $(u)$ may increase at most once in $e$ : when rule_Clr$(u)$ is executed while $\# \operatorname{InAll}(u)>$ $f(u)$. So,
(3) Every process $u$ updates the value of $s c r_{u}$ at most 4 times in $e$.

Hence, overall (by (1)-(3)), the value of $\mathbf{P}_{-}$canQuit $(v)$ changes at most $4 \delta_{v}+2$ in $e$, and thus
(4) $v$ updates the value of $\operatorname{can} Q_{v}$ at most $4 \delta_{v}+3$ in $e$.

By (3) and (4)
(5) rule $\mathbf{Q}(v)$ is executed at most most $4 \delta_{v}+7$ times in $e$.

The value of bestPtr $(v)$ may change only when a process $u$ in the closed neighborhood of $v$ changes the value of its variable $\operatorname{can} Q_{u}$ or when $v$ updates $s c r_{v}$. So, by (3) and (4), the value of bestPtr $(v)$ changes at most $\delta_{v} \cdot(4 \Delta+3)+4 \delta_{v}+7$ times. So,
(6) $v$ executes rule_P1 and rule_P2 at most $\delta_{v} \cdot(4 \Delta+3)+4 \delta_{v}+8$ times each in $e$.

Overall (by (1), (5), and (6)), follows.
Lemma 25 A process $v$ executes at most $8 \delta_{v} \Delta+18 \delta_{v}+24$ moves in an execution of FGA .
Corollary 11 Any execution of FGA contains at most $16 . \Delta . m+36 . m+24 . n$ moves, i.e., $O(\Delta . m)$ moves.
By Corollaries 10 and 11 , we can conclude with the following theorem.
Theorem 9 FGA is distributed (non self-stabilizing) 1-minimal $(f, g)$-alliance algorithm which terminates in at most $O(\Delta . m)$ moves.

Round Complexity. We already know that in any execution of FGA, each process executes rule_Clr at most once. So, along any execution there are at most $n$ steps containing the execution of some rule_Clr. We now say that a step is Color-restricted, if no rule rule_Clr is executed during that step. Similarly, we say that a round is Color-restricted if it only consists of Color-restricted steps. In the sequel, we show that any execution that starts from a configuration where $\mathbf{P}_{-} \mathbf{C l e a n}(u) \wedge \mathbf{P} \_\operatorname{ICorrect}(u)$ holds for every process $u$ contains at most 4 consecutive Color-restricted rounds. Hence, the number of rounds in any execution starting from $\gamma_{\text {init }}$ is bounded by $5 n+4$. To that goal, we first specialize the notion of closure. A predicate $P$ over configurations of FGA is Color-restricted closed if for every Color-restricted step $\gamma \mapsto \gamma^{\prime}$, $P(\gamma) \Rightarrow P\left(\gamma^{\prime}\right)$.

We then consider the following predicates over configurations of FGA.

- $\mathcal{P}_{5}$ is true if and only if every process $u$ satisfies $\mathbf{P}_{-}$Clean $(u) \wedge$ P_ICorrect $(u)$.
- $\mathcal{P}_{6}$ is true if and only if $\mathcal{P}_{5}$ holds and every process $u$ satisfies $\operatorname{scr}_{u}=\operatorname{realScr}(u)$.
- $\mathcal{P}_{7}$ is true if and only if $\mathcal{P}_{6}$ holds and every process $u$ satisfies $\operatorname{can} Q_{u}=\mathbf{P}_{-}$canQuit $(u)$.
- $\mathcal{P}_{8}$ is true if and only if $\mathcal{P}_{7}$ holds and every process $u$ satisfies $\operatorname{ptr}_{u} \in\{$ bestPtr $(u), \perp\}$.
- $\mathcal{P}_{9}$ is true if and only if $\mathcal{P}_{8}$ holds and every process $u$ satisfies $\operatorname{ptr}_{u}=\operatorname{bestPtr}(u)$.

Lemma $26 \mathcal{P}_{6}$ is Color-restricted closed. Moreover, after one Color-restricted round from any configuration satisfying $\mathcal{P}_{5}$, a configuration satisfying $\mathcal{P}_{6}$ is reached.

Proof. Let $u$ be a process. The value of $\operatorname{realScr}(u)$ stays unchanged during a Color-restricted step. So, the predicate $s c r_{u}=$ realScr $(u)$ is Color-restricted closed, and so $\mathcal{P}_{6}$ is. Let $\gamma$ be a configuration satisfying $\mathcal{P}_{5}$. Recall that $\mathcal{P}_{5}$ is closed, by Corollary 8 . So, P_Clean $(u) \wedge$ P_ICorrect $^{\prime}(u)$ holds forever from $\gamma$. If $s c r_{u} \neq \operatorname{realScr}(u)$ in $\gamma$, then $u$ is enabled in FGA. Now, if $u$ moves in a Color-restricted step, then $s c r_{u}=$ realScr $(u)$ in the reached configuration. Hence, $s c r_{u}=\operatorname{realScr}(u)$ holds, for every process $u$, within at most one round from $\gamma$, and we are done.

Lemma $27 \mathcal{P}_{7}$ is Color-restricted closed. Moreover, after one Color-restricted round from any configuration satisfying $\mathcal{P}_{6}$, a configuration satisfying $\mathcal{P}_{7}$ is reached.

Proof. Let $u$ be a process. Let $\gamma_{i} \mapsto \gamma_{i+1}$ be a Color-restricted step such that $\mathcal{P}_{6}\left(\gamma_{i}\right)$ holds. For every process $v$, the value of $c o l_{v}, s c r_{v}$, and $\# \operatorname{In} \operatorname{All}(v)$ stay unchanged during $\gamma_{i} \mapsto \gamma_{i+1}$. Therefore, the value of $\mathbf{P}^{c}$ canQuit $(u)$ stays unchanged during $\gamma_{i} \mapsto \gamma_{i+1}$ for every process $u$. So, the predicate $\mathcal{P}_{6} \wedge c a n Q_{u}=$ P_canQuit $(u)$ is Color-restricted closed, and so $\mathcal{P}_{7}$ is. Assume now that, can $Q_{u} \neq \mathbf{P}^{\prime}$ canQuit $(u)$ holds in $\gamma_{i}$. Then, $u$ is enabled in $\gamma_{i}$ and if $u$ moves in $\gamma_{i} \mapsto \gamma_{i+1}, \operatorname{can} Q_{u}=\mathbf{P}$ canQuit $(u)$ holds in in $\gamma_{i+1}$. Indeed, the value of $\mathbf{P}_{-}$canQuit $(u)$ stays unchanged during the step. Therefore, after at most one Color-restricted round from $\gamma_{i}$, we have $c a n Q_{u}=\mathbf{P}$ _canQuit $(u)$ for every process $u$, and so $\mathcal{P}_{7}$ holds.

Lemma $28 \mathcal{P}_{8}$ is Color-restricted closed. Moreover, after one Color-restricted round from any configuration satisfying $\mathcal{P}_{7}$, a configuration satisfying $\mathcal{P}_{8}$ is reached.

Proof. Let $u$ be a process. Let $\gamma_{i} \mapsto \gamma_{i+1}$ be a Color-restricted step such that $\mathcal{P}_{7}\left(\gamma_{i}\right)$ holds. During that step, only rules rule_P1 or rule_P2 are executed. Moreover, for every process $v$, the value of colv and $\operatorname{can} Q_{v}$ stay unchanged during $\gamma_{i} \mapsto \gamma_{i+1}$. So, the value of $\operatorname{bestPtr}(u)$ stays unchanged as well. So, the predicate $\mathcal{P}_{7} \wedge p \operatorname{tr}_{u} \in\{$ best $\operatorname{Ptr}(u), \perp\}$ is Color-restricted closed, and so $\mathcal{P}_{8}$ is. Assume now that, $p r_{u} \notin\{\operatorname{bestPtr}(u), \perp\}$ holds in $\gamma_{i}$. Then, $u$ is enabled in $\gamma_{i}$ and if $u$ moves in $\gamma_{i} \mapsto \gamma_{i+1}, p t r_{u} \in$ $\{\operatorname{bestPtr}(u), \perp\}$ in $\gamma_{i+1}$. Indeed, the value of $\operatorname{best} \operatorname{Ptr}(u)$ stays unchanged during the step. Therefore, after at most one Color-restricted round from $\gamma_{i}$, we have $\operatorname{ptr}_{u} \in\{$ bestPtr $(u), \perp\}$ for every process $u$, and so $\mathcal{P}_{8}$ holds.

Lemma $29 \mathcal{P}_{9}$ is Color-restricted closed. Moreover, after one Color-restricted round from any configuration satisfying $\mathcal{P}_{8}$, a configuration satisfying $\mathcal{P}_{9}$ is reached.

Proof. Let $u$ be a process. Let $\gamma_{i} \mapsto \gamma_{i+1}$ be a Color-restricted step such that $\mathcal{P}_{8}\left(\gamma_{i}\right)$ holds. During that step, only rules rule_P2 are executed. Moreover, for every process $v$, the value of $\operatorname{col}_{v}$ and $\operatorname{can}_{2} Q_{v}$ stay unchanged during $\gamma_{i} \mapsto \gamma_{i+1}$. So, the value of $\operatorname{bestPtr}(u)$ stays unchanged as well. So, the predicate $\mathcal{P}_{8} \wedge p \operatorname{tr} r_{u}=\operatorname{bestPtr}(u)$ is Color-restricted closed, and so $\mathcal{P}_{9}$ is. Assume now that, $\operatorname{ptr}_{u} \neq \operatorname{bestPtr}(u)$ holds in $\gamma_{i}$. Then, $u$ is enabled in $\gamma_{i}$ and if $u$ moves in $\gamma_{i} \mapsto \gamma_{i+1}, p \operatorname{tr} r_{u}=\operatorname{bestPtr}(u)$ in $\gamma_{i+1}$. Indeed, the value of bestPtr $(u)$ stays unchanged during the step. Therefore, after at most one Color-restricted round from $\gamma_{i}$, we have $p \operatorname{tr}_{u}=\operatorname{bestPtr}(u)$ for every process $u$, and so $\mathcal{P}_{8}$ holds.

Theorem 10 Starting from any configuration satisfying $\mathcal{P}_{5}$, Algorithm FGA terminates in at most $5 n+4$ rounds.

Proof. According to Lemmas 26-29, after any 4 consecutive Color-restricted rounds, every process $u$ satisfies:

$$
\operatorname{scr}_{u}=\operatorname{realScr}(u) \quad \wedge \quad \operatorname{can} Q_{u} \quad=\quad \mathbf{P}_{-} \text {canQuit }(u) \quad \wedge \neg \mathbf{P} \text { _updPtr}(u)
$$

So only rule_Clr $(u)$ may be enabled at $u$. We conclude that an execution of rule_Clr occurs at least every 5 rounds, unless the system reaches a terminal configuration. Since, along any execution, there are at most $n$ steps containing the execution of some rule_Clr, the theorem follows.

Since $\gamma_{\text {init }}$ satisfies $\mathcal{P}_{5}$, we have the following corollary.
Corollary 12 Starting from $\gamma_{\text {init }}$, Algorithm FGA terminates in at most $5 n+4$ rounds.

### 6.5 Algorithm $F G A \circ S D R$

Requirements. To show the self-stabilization of FGA $\circ S D R$, we should first establish that FGA meets the requirements 1 to 2d, given in Subsection 3.5.

1. From the code of FGA, we can deduce that Requirements $1,2 \mathrm{~b}, 2 \mathrm{c}$, and 2 e are satisfied.
2. Requirement 2 a is satisfied since $\mathbf{P}$ _ICorrect $(u)$ does not involve any variable of SDR and is closed by FGA (Corollary 9).
3. Finally, recall that $\delta_{u} \geq \max (f(u), g(u))$, for every process $u$. So, if $\mathbf{P} \_$reset $(v)$ holds, for every $v \in N[u]$, then $\operatorname{realScr}(u)=1$ and so P_ICorrect $(u)$ holds, by definition. Hence, Requirement 2d holds.

Partial Correctness. Let $\gamma$ be any terminal configuration of $F G A \circ S D R$. Then, $\gamma_{\mid S D R}$ is a terminal configuration of SDR and, by Theorem 1, P_Clean $(u) \wedge \mathbf{P} \_$ICorrect $(u)$ holds in $\gamma$, and so $\gamma_{\mid \text {SDR }}$, for every process $u$. Moreover, $\gamma_{\mid F G A}$ is a terminal configuration of FGA. Hence, by Theorem 8 , follows.

Theorem 11 In any terminal configuration of FGA $\circ$ SDR. the set $\left\{u \in V \mid \operatorname{col} l_{u}\right\}$ is a 1-minimal $(f, g)$ alliance of the network.

Termination and Self-Stabilization. Let $u$ be a process. Let $e$ be an execution of FGA $\circ$ SDR. By Corollary 3 (page 16), the sequence of rules executed by a process $u$ in a segment of $e$ belongs to the following language:

$$
(\text { rule_C }+\varepsilon) \text { words_I }(\text { rule_RB }+ \text { rule_R }+\varepsilon)(\text { rule_RF }+\varepsilon)
$$

where words_I be any sequence of rules of FGA. Moreover, by Lemma 10 (page 17) and Lemma 25, words_I is bounded by $8 \delta_{u} \Delta+18 \delta_{u}+24$. Thus, $u$ executes at most $8 \delta_{u} \Delta+18 \delta_{u}+27$ moves in any segment of $e$ and, overall each segment of $e$ contains at most $16 m \Delta+36 m+27 n$ moves. Since, $e$ contains at most $n+1$ segments (Remark 5, page 13), $e$ contains at most $(n+1) \cdot(16 m \Delta+36 m+27 n)$ moves, and we have the following theorem.

Theorem 12 Any execution of $\mathrm{FGA} \circ \mathrm{SDR}$ terminates in $O(\Delta . n . m)$ moves.
By Theorems 11 and 12, we can conclude:
Theorem 13 FGA $\circ$ SDR is self-stabilizing for the 1-minimal $(f, g)$-alliance problem. Its stabilization time is in $O(\Delta$.n.m) moves.

Round Complexity. Corollary 5 (page 20) establishes that after at most $3 n$ rounds a normal configuration of FGA $\circ S D R$ is reached. Then, since the set of normal configuration is closed (still by Corollary 5), all rules of $S D R$ algorithm are disabled forever from such a configuration, by Lemma 15 (page 20). Moreover, in a normal configuration, P_Clean $(u) \wedge \mathbf{P}$ _ICorrect $(u)$ holds, for every process $u$ (still by Corollary 5). Hence, after at most $5 n+4$ additional rounds, a terminal configuration of $F G A \circ S D R$ is reached, by Theorem 10, and follows.

Theorem 14 The stabilization time of $\mathrm{FGA} \circ \mathrm{SDR}$ is at most $8 n+4$ rounds.

## References

[1] Yehuda Afek and Anat Bremler-Barr. Self-stabilizing unidirectional network algorithms by power supply. Chicago J. Theor. Comput. Sci., 1998, 1998.
[2] Karine Altisen, Alain Cournier, Stéphane Devismes, Anaïs Durand, and Franck Petit. Self-stabilizing leader election in polynomial steps. Inf. Comput., 254:330-366, 2017.
[3] Dana Angluin, James Aspnes, David Eisenstat, and Eric Ruppert. The computational power of population protocols. Distributed Computing, 20(4):279-304, 2007.
[4] Anish Arora and Mohamed G. Gouda. Distributed reset. IEEE Trans. Computers, 43(9):1026-1038, 1994.
[5] Baruch Awerbuch and Rafail Ostrovsky. Memory-efficient and self-stabilizing network \{RESET\} (extended abstract). In James H. Anderson, David Peleg, and Elizabeth Borowsky, editors, Proceedings of the Thirteenth Annual ACM Symposium on Principles of Distributed Computing, Los Angeles, California, USA, August 14-17, 1994, pages 254-263. ACM, 1994.
[6] Baruch Awerbuch, Boaz Patt-Shamir, and George Varghese. Self-stabilization by local checking and correction (extended abstract). In 32nd Annual Symposium on Foundations of Computer Science, San Juan, Puerto Rico, 1-4 October 1991, pages 268-277. IEEE Computer Society, 1991.
[7] Baruch Awerbuch, Boaz Patt-Shamir, George Varghese, and Shlomi Dolev. Self-stabilization by local checking and global reset (extended abstract). In Distributed Algorithms, 8th International Workshop, WDAG '94, Terschelling, The Netherlands, September 29 - October 1, 1994, Proceedings, volume 857 of Lecture Notes in Computer Science, pages 326-339. Springer, 1994.
[8] Jalel Ben-Othman, Karim Bessaoud, Alain Bui, and Laurence Pilard. Self-stabilizing algorithm for efficient topology control in wireless sensor networks. Journal of Computational Science, 4(4):199 208, 2013.
[9] C. Berge. The Theory of Graphs. Dover books on mathematics. Dover, 2001.
[10] Christian Boulinier. L'Unisson. PhD thesis, Université de Picardie Jules Vernes, France, 2007.
[11] Christian Boulinier, Franck Petit, and Vincent Villain. When graph theory helps self-stabilization. In Proceedings of the Twenty-Third Annual ACM Symposium on Principles of Distributed Computing (PODC), July 25-28, 2004, pages 150-159, 2004.
[12] Alain Bui, Ajoy Kumar Datta, Franck Petit, and Vincent Villain. Optimal PIF in tree networks. In Yuri Breitbart, Sajal K. Das, Nicola Santoro, and Peter Widmayer, editors, Distributed Data \& Structures 2, Records of the 2nd International Meeting (WDAS 1999), Princeton, USA, May 10-11, 1999, volume 6 of Proceedings in Informatics, pages 1-16. Carleton Scientific, 1999.
[13] Eddy Caron, Florent Chuffart, and Cédric Tedeschi. When self-stabilization meets real platforms: An experimental study of a peer-to-peer service discovery system. Future Generation Computer Systems, 29(6):1533-1543, 2013.
[14] Eddy Caron, Ajoy Kumar Datta, Benjamin Depardon, and Lawrence L. Larmore. A self-stabilizing k-clustering algorithm for weighted graphs. J. Parallel Distrib. Comput., 70(11):1159-1173, 2010.
[15] Eddy Caron, Frédéric Desprez, Franck Petit, and Cédric Tedeschi. Snap-stabilizing prefix tree for peer-to-peer systems. Parallel Processing Letters, 20(1):15-30, 2010.
[16] Fabienne Carrier, Ajoy Kumar Datta, Stéphane Devismes, Lawrence L. Larmore, and Yvan Rivierre. Self-stabilizing (f, g)-alliances with safe convergence. J. Parallel Distrib. Comput., 81-82:11-23, 2015.
[17] Nian-Shing Chen, Hwey-Pyng Yu, and Shing-Tsaan Huang. A self-stabilizing algorithm for constructing spanning trees. Inf. Process. Lett., 39(3):147-151, 1991.
[18] Alain Cournier, Ajoy K. Datta, Franck Petit, and Vincent Villain. Snap-stabilizing PIF algorithm in arbitrary networks. In Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS'02), pages 199-206, 2002.
[19] Jean-Michel Couvreur, Nissim Francez, and Mohamed G. Gouda. Asynchronous unison (extended abstract). In Proceedings of the 12th International Conference on Distributed Computing Systems, Yokohama, Japan, June 9-12, 1992, pages 486-493. IEEE Computer Society, 1992.
[20] Jean-Michel Couvreur, Nissim Francez, and Mohamed G. Gouda. Asynchronous unison (extended abstract). In Proceedings of the 12nd International Conference on Distributed Computing Systems (ICDCS'92), pages 486-493, 1992.
[21] Ajoy K. Datta, Lawrence L. Larmore, and Priyanka Vemula. An $O(N)$-time self-stabilizing leader election algorithm. JPDC, 71(11):1532-1544, 2011.
[22] Stéphane Devismes and Colette Johnen. Silent self-stabilizing BFS tree algorithms revisited. JPDC, 97:11-23, 2016.
[23] Stéphane Devismes and Franck Petit. On efficiency of unison. In 4th Workshop on Theoretical Aspects of Dynamic Distributed Systems, (TADDS '12), pages 20-25, 2012.
[24] Edsger W. Dijkstra. Self-stabilizing Systems in Spite of Distributed Control. Commun. ACM, 17(11):643-644, 1974.
[25] Yihua Ding, James Z. Wang, and Pradip K. Srimani. Self-stabilizing minimal global offensive alliance algorithm with safe convergence in an arbitrary graph. In T. V. Gopal, Manindra Agrawal, Angsheng Li, and S. Barry Cooper, editors, Theory and Applications of Models of Computation - 11th Annual Conference (TAMC), volume 8402 of Lecture Notes in Computer Science, pages 366-377, Chennai, India, April 11-13 2014. Springer.
[26] S Dolev, A Israeli, and S Moran. Self-stabilization of dynamic systems assuming only Read/Write atomicity. Distributed Computing, 7(1):3-16, 1993.
[27] Shlomi Dolev. Self-stabilizing routing and related protocols. Journal of Parallel and Distributed Computing, 42(2):122-127, 1997.
[28] Shlomi Dolev, Mohamed G. Gouda, and Marco Schneider. Memory requirements for silent stabilization. Acta Informatica, 36(6):447-462, 1999.
[29] Mitre Costa Dourado, Lucia Draque Penso, Dieter Rautenbach, and Jayme Luiz Szwarcfiter. The south zone: Distributed algorithms for alliances. In Xavier Défago, Franck Petit, and Vincent Villain, editors, Stabilization, Safety, and Security of Distributed Systems - 13th International Symposium, (SSS), volume 6976 of Lecture Notes in Computer Science, pages 178-192, Grenoble, France, October 10-12 2011. Springer.
[30] M. R. Garey and David S. Johnson. Computers and Intractability: A Guide to the Theory of NPCompleteness. W. H. Freeman, 1979.
[31] Christian Glacet, Nicolas Hanusse, David Ilcinkas, and Colette Johnen. Disconnected components detection and rooted shortest-path tree maintenance in networks. In SSS' 14 , pages 120-134, 2014.
[32] Anupam Gupta, Bruce M. Maggs, Florian Oprea, and Michael K. Reiter. Quorum placement in networks to minimize access delays. In Marcos Kawazoe Aguilera and James Aspnes, editors, Proceedings of the Twenty-Fourth Annual ACM Symposium on Principles of Distributed Computing (PODC), pages 87-96, Las Vegas, NV, USA, July 17-20 2005. ACM.
[33] Shing-Tsaan Huang and Nian-Shing Chen. A self-stabilizing algorithm for constructing breadth-first trees. IPL, 41(2):109-117, 1992.
[34] Shing-Tsaan Huang and Nian-Shing Chen. Self-stabilizing depth-first token circulation on networks. Distributed Computing, 7(1):61-66, 1993.
[35] Hirotsugu Kakugawa and Toshimitsu Masuzawa. A self-stabilizing minimal dominating set algorithm with safe convergence. In 20th International Parallel and Distributed Processing Symposium (IPDPS), pages 8.-, Rhodes Island, Greece, 25-29 April 2006. IEEE.
[36] Shmuel Katz and Kenneth J. Perry. Self-stabilizing extensions for message-passing systems. Distributed Computing, 7(1):17-26, 1993.
[37] Yonghwan Kim, Junya Nakamura, Yoshiaki Katayama, and Toshimitsu Masuzawa. A cooperative partial snapshot algorithm for checkpoint-rollback recovery of large-scale and dynamic distributed systems. In CANDAR'18, pages 285-291, 112018.
[38] Chung-Shou Liao and Gerard J. Chang. k-tuple domination in graphs. Inf. Process. Lett., 87(1):45-50, 2003.
[39] Gerry Siegemund, Volker Turau, Christoph Weyer, Stefan Lobs, and Jörg Nolte. Brief announcement: Agile and stable neighborhood protocol for wsns. In Proceedings of the 15th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS'13), pages 376-378, November 2013.
[40] J.M. Sigarreta and J.A. Rodríguez. On the global offensive alliance number of a graph. Discrete Applied Mathematics, 157(2):219-226, 2009.
[41] Jose Maria Sigarreta and Juan Alberto Rodríguez-Velazquez. On defensive alliances and line graphs. Appl. Math. Lett., 19(12):1345-1350, 2006.
[42] M Sloman and J Kramer. Distributed systems and computer networks. Prentice Hall, 1987.
[43] Pradip K. Srimani and Zhenyu Xu. Distributed protocols for defensive and offensive alliances in network graphs using self-stabilization. In International Conference on Computing: Theory and Applications (ICCTA), pages 27-31, Kolkata, India, March 5-7 2007. IEEE Computer Society.
[44] Volker Turau. Linear self-stabilizing algorithms for the independent and dominating set problems using an unfair distributed scheduler. Inf. Process. Lett., 103(3):88-93, 2007.
[45] Guangyuan Wang, Hua Wang, Xiaohui Tao, and Ji Zhang. A self-stabilizing algorithm for finding a minimal k-dominating set in general networks. In Yang Xiang, Mukaddim Pathan, Xiaohui Tao, and Hua Wang, editors, Data and Knowledge Engineering, Lecture Notes in Computer Science, pages 74-85. Springer Berlin Heidelberg, 2012.
[46] Saïd Yahiaoui, Yacine Belhoul, Mohammed Haddad, and Hamamache Kheddouci. Self-stabilizing algorithms for minimal global powerful alliance sets in graphs. Inf. Process. Lett., 113(10-11):365370, 2013.


[^0]:    *This study was partially supported by the French ANR projects ANR-16-CE40-0023 (DESCARTES) and ANR-16 CE25-0009-03 (ESTATE).

[^1]:    ${ }^{1}$ In case of several enabled actions at the activated process, the choice of the executed action is nondeterministic.

[^2]:    ${ }^{2}$ Otherwise, $s t_{u}=R B$ in some configuration of $S_{l a s t}$, and that configuration contains an abnormal root, by Lemma 7 (page 13), a contradiction.

