On Self-stabilizing Leader Election in Directed Networks

Karine Altisen, Alain Cournier, Geoffrey Defalque, Stéphane Devismes

On Self-stabilizing Leader Election in Directed Networks*

Karine Altisen† Alain Cournier‡ Geoffrey Defalque‡
Stéphane Devismes‡

Abstract

In this paper, we consider identified directed networks where processes know an upper bound on the maximum ancestor distance. Under these settings, we study the conditions on the network topology allowing the self-stabilization of two fundamental problems: the leader election and the synchronous unison. Precisely, we show that those two problems can be self-stabilizingly solved in our settings if and only if the network contains a unique source component. In particular, to show that our condition is sufficient, we propose two algorithms and study their complexity. Notice that our topological condition covers a wide spectrum of digraphs since, for example, strongly connected digraphs, dipaths, and out-trees have a unique source component.

Keywords: directed networks, self-stabilization, necessary and sufficient conditions, leader election, synchronous unison.

1 Introduction

1.1 Context

Fault tolerance, *i.e.*, the ability of a system to withstand or recover from failures, is a major concern in modern networks. Indeed, on the one hand, these distributed systems are broadly

*This study has been partially supported by the French ANR project ANR-22-CE25-0008-01 (SKYDATA).

†Université Grenoble Alpes, VERIMAG, Grenoble (France)
‡Université de Picardie Jules Verne, MIS, Amiens (France)

autonomous and large-scale; thus, human intervention to repair them is often difficult and even sometimes impossible. On the other hand, their lifespan and availability must be maximized. However, fault tolerance is usually hard to obtain in distributed computing and, when it can be achieved, it often comes at the the price of sacrificing efficiency; see, *e.g.*, [DR82, CYZG14]. To circumvent this issue, one can consider *self-stabilization* [Dij74], a general *lightweight* fault tolerance paradigm [ADDP19]. Precisely, a distributed system achieving this property inherently tolerates *any* finite number of transient faults.[1] Indeed, starting from an arbitrary configuration, which may be the result of such faults, a self-stabilizing system recovers *within finite time*, and without any external intervention, a so-called *legitimate configuration* from which it satisfies its specification.

In many today's networks —such as Wireless Sensor Networks (WSNs), Optical Transport Networks, or Internet of Things— the topology is a *directed graph* (digraph for short) since several communication links may not be bidirectional. In other words, the possibility of sending information to some process does not necessarily imply the possibility of receiving information from that process. For example, in WSNs, communication is made by radio waves. Now, antennas may have different ranges due to the heterogeneity of radio supplies making then some communication channels one-way only. Now, despite the pioneer work of Dijkstra [Dij74] deals with unidirectional rings, most of the self-stabilizing literature actually focuses on bidirectional networks [KP93, CD94, CDV05, CDV06, DLV11a].

In bidirectional networks, most of the tasks can be made self-stabilizing as soon as the topology is connected [KP93]. When considering directed graph topologies, the connectivity is declined into two widely different notions: the weak connectivity and the strong connectivity. Most of distributed computing problems cannot be solved in an arbitrary weakly connected topology since several nodes may not be able to exchange information each other in any direction. In contrast, most of existing self-stabilization solutions working in directed graphs assume strong connectivity; see, *e.g.*, [AB98, KY02]. Now, there is a huge gap between these two notions and the main question tackles in this paper is where to place the line between what is feasible and what is not, *i.e.*, what are the necessary and sufficient conditions for self-stabilization in directed networks?

In this paper, we partially answer this question. We consider identified directed networks where processes know an upper bound α on the maximum ancestor distance MAD(*i.e.*, the maximum distance from a ancestor to one of its descendent). In such networks, we propose a condition on the network topology that is necessary and sufficient for the self-stabilization of two important benchmark problems of the literature: the leader election and the synchronous unison. Interestingly, these two problems are two agreement problems

[1] A *transient fault* occurs at an unpredictable time, but does not result in a permanent hardware damage. Moreover, as opposed to intermittent faults, the frequency of transient faults is considered to be low.

but of widely different nature in the sense that leader election is a *static problem*, while synchronous unison is a *dynamic problem*. As opposed to dynamic problems such as token circulation, a static problem (*e.g.*, computing a spanning tree) defines a task of calculating a function that depends on the system in which it is evaluated [Tix06].

1.2 Contribution

We consider the atomic-state model, the most commonly used model in the self-stabilizing area. In self-stabilization, deterministic solutions often require unique identifiers and the knowledge of some global network parameter is usually mandatory to bound process local memories. Recall that we deal here with identified directed networks where processes know an upper bound α on MAD, the maximum ancestor distance.

In these settings, we first show that there exists a self-stabilizing leader election algorithm if and only if the network topology contains exactly one source component, *i.e.*, a strongly connected component where all nodes have no predecessor out of the component. This necessary and sufficient condition holds both in asynchronous (precisely, the distributed unfair daemon, the most general scheduling assumption of the model) and synchronous settings. The sufficient condition is constructive as we propose a (silent) self-stabilizing leader election algorithm that stabilizes in $O(\alpha)$ rounds using $O(\log \alpha + B)$ bits per process (B being the number of bits required to store any identifier) under the distributed unfair daemon.

Then, from our leader election algorithm, we easily derive a self-stabilizing synchronous unison algorithm for networks with a unique source component. This latter problem is a clock synchronization problem: each process holds a local clock and at each step all clocks have to synchronously increment modulo K (with $K \geq 2$) so that all clocks of the network are always equal. This latter specification requires the system to be synchronous. Our algorithm stabilizes in $O(\alpha)$ synchronous rounds using $O(\log \alpha + B + \log K)$ bits per process. From this outcome and the results in [ACDD23], we deduce that the fact that the network should have a unique source component is both necessary and sufficient for solving the self-stabilizing synchronous unison.

Overall, our results show, maybe surprisingly, that in directed networks the weakest topological condition to solve two widely different problems —namely the "static" leader election and the "dynamic" synchronous unison— is the same.

Notice that our topological condition covers a wide spectrum of digraphs since, for example, strongly connected digraphs, dipaths, and out-trees have a unique source component.

1.3 Technical Overview

To simplify the design of our solutions and increase the soundness of their proof, we design our solutions as compositions of several simple building blocks. To that goal, we extensively use a composition technique called the *hierarchical collateral composition* [DLD+13]. Up to now, this composition was only used assuming a distributed weakly fair daemon. Hence, we had to develop a few general properties to prove the self-stabilization of a composite algorithm under the more general distributed unfair daemon. Actually, our leader election algorithm is a composition of three different instances of the same basic building block called Algorithm C-MAI$(\mathcal{D}, \prec, \mathcal{I})$. Algorithm C-MAI$(\mathcal{D}, \prec, \mathcal{I})$ is a (silent) self-stabilizing algorithm that simply computes at each process the minimum value among the inputs of all its ancestors; the input of each process p being $\mathcal{I}(p) \in \mathcal{D}$, where $\mathcal{D}$ is a set totally ordered by $\prec$. Notice that to prove the termination of Algorithm C-MAI$(\mathcal{D}, \prec, \mathcal{I})$ we use the notion of *limit inferior* of an infinite sequence, a powerful general tool that allows us to drastically refine the proof.

1.4 Related Work

Self-stabilization in Directed Networks. In directed networks, the maybe asymmetric communication between surrounding processes often makes problems more complicated, and sometimes even impossible to solve. A folklore example is the token circulation that cannot be solved in any arbitrary weakly connected network: the network topology should be at least strongly connected. In such a context, a preliminary important task is then to properly define the minimum assumptions for which the considered problem is solvable. Furthermore, switching from undirected to directed topologies may also impact the complexity when the problem remains solvable. For example, some problems, such as the self-stabilizing vertex coloring in anonymous networks under a central scheduler, can be implemented in undirected networks with a space requirement linear on the degree of vertices [ADDP19], while the space complexity lower bound becomes dependent on the number of processes in the directed case; see [BDPT09].

Maybe surprisingly, the self-stabilizing literature on directed networks is far less extensive than on undirected networks. Many works actually focus on restricted topologies, namely unidirectional rings [Dij74, MOY96, HL99, KY02]. Yet, some other works deal with more general topologies [AB98, DDT06, BDPT09]. More precisely, Afek and Bremler [AB98] consider strongly connected directed networks for which they propose efficient self-stabilizing solutions to leader election and rooted spanning trees. Delaët *et al.* [DT01, DDT06] propose a method to design (silent) self-stabilizing algorithms for a class of fix-point problems, namely fix-point problems which can be expressed using r-operators. This method applies as soon as the network topology admits a solution to the con-

sidered problem. In [DT01], they consider the link-register, while in [DDT06], they generalize their approach to asynchronous message-passing systems. In both papers, they establish a stabilization time in $O(MAD + |\mathbb{S}|)$ rounds where $\mathbb{S}$ is the set on which the r-operator applies.[2] However, this bound is proven for the synchronous case only. Finally, self-stabilizing vertex-coloring of arbitrary directed networks is considered in [BDPT09, ADDP19]. Overall, all these aforementioned papers [AB98, DDT06, BDPT09] deal with static problems. To the best of our knowledge, self-stabilization for dynamic problems have never been addressed in wide classes of directed networks, except in [ACDD23]. Precisely, token circulation [Dij74], clock synchronization [HL99], and mutual exclusion [KY02] have been considered in the restricted case of unidirectional rings. In [ACDD23], the synchronous unison is investigated in wide classes of anonymous directed networks. We will further elaborate on [ACDD23] in the last part of this related work, which focuses on the synchronous unison problem.

Self-stabilizing Leader Election. There are numerous (deterministic) self-stabilizing leader election algorithms for identified networks [AG94, DH97, AB98, AK93, BK07, DLP10, DLV11a, DLV11b, KK13, ACD$^+$17]. They all assume any identifier can be stored in $O(\log n)$ bits. Most of these solutions consider bidirectional links; more precisely, the network topology is modeled as an undirected connected graph.

Actually, to the best of our knowledge, until now only the algorithm of Afek and Bremler [AB98] considered directed networks. Yet, they assume the topology is strongly connected and consider the message-passing model where the link-capacity is bounded by a value c, known by all processes. In these settings, they propose an algorithm that stabilizes in $O(\mathscr{D})$ rounds using $O(\log n)$ bits per process, where $\mathscr{D}$ is the network diameter and n is the number of processes.

Self-stabilizing leader election algorithms for bidirectional networks have been proposed in the message-passing model [AK93, BK07]. Both solutions assume processes know some upper bound $\mathtt{D}$ on $\mathscr{D}$ and stabilizes in $O(\mathscr{D})$ rounds using $O(\log \mathtt{D} \log n)$ bits per process.

Dolev and Herman [DH97] consider the self-stabilizing leader election in the link-register model. They assume that all processes know an upper bound N on n. Their solution stabilizes in $O(\mathscr{D})$ rounds using $O(N \log N)$ bits per process.

Several solutions are also given in the atomic-state model [AG94, DLP10, DLV11a, DLV11b, KK13, ACD$^+$17]. The solution in [DLP10] uses infinite local memories. All other solutions need the knowledge of an upper bound N on n and achieve a memory requirement in $O(\log N)$ bits per process. The algorithm of Arora and Gouda [AG94]

[2] In [DT01, DDT06], the claimed complexity is $O(\mathscr{D} + |\mathbb{S}|)$, where $\mathscr{D}$ is called the diameter and is defined as the maximum of the distances between all couples of vertices for which a distance is defined; this latter notion being equivalent to MAD.

works under a weakly fair daemon and stabilizes in $O(N)$ rounds. The algorithm proposed by Kravchik and Kutten [KK13] assumes a synchronous daemon and stabilizes in $O(\mathcal{D})$ rounds. The solutions in [DLP10, DLV11a, DLV11b, ACD+17] assume a distributed unfair daemon and have a stabilization time in $O(n)$ rounds.

Self-stabilizing Synchronous Unison. The synchronous unison problem has been introduced by Even and Rajsbaum [ER90]. In their paper, Even and Rajsbaum consider the problem in a non-fault-tolerant context, yet assuming that processes do not necessarily start at the same time. The network is assumed to be a strongly connected directed graph.

Gouda and Herman [GH90] have proposed the first self-stabilizing synchronous unison. Their algorithm works in anonymous synchronous systems of arbitrary bidirectional and connected topology using infinite clocks. A solution working with the same settings, yet implementing bounded clocks, is proposed in [ADG91]. Both solutions stabilize in $O(\mathscr{D})$ rounds.

To the best of our knowledge, self-stabilizing synchronous unison in directed networks has been only considered in [HL99, ACDD23]. [HL99] focuses on the restricted case of unidirectional rings of odd size. [ACDD23] studies the impact in terms of both requirements and efficiency when extending the self-stabilizing synchronous unison of [ADG91] to directed network topologies. In particular, our condition on network topology is demonstrated to be necessary in anonymous settings, but it is not proven to be sufficient: there is still a little gap between the necessary and the sufficient conditions proposed in [ADG91].

The synchronous unison has been generalized to tackle asynchronous networks. The so-called *asynchronous unison* requires neighboring clocks to differ from at most one increment. To the best of our knowledge, this problem has been only investigated in bidirectional networks. Yet, in [JADT02], a solution is proposed for networks already containing a spanning tree and only the tree links toward its root are used. Hence, the proposed algorithm also works in directed trees. This solution uses only two states per process and stabilizes in 0 round since using two clock values no configuration is illegitimate.

Then, there are several self-stabilizing solutions for general connected anonymous networks [CFG92, AK93, BPV04, EK21]. The first one has been proposed by Couvreur *et al.* [CFG92]. However, no complexity analysis was given. Another solution which stabilizes in $O(n)$ (asynchronous) rounds has been presented by Boulinier *et al.* in [BPV04]. Both solutions require $O(\log N)$ bits per process, where N is an upper bound on n known by all processes, to work on any bidirectional topology. Boulinier proposed in his PhD thesis [Bou07] a parametric solution which generalizes both the solutions of [CFG92] and [BPV04]. In particular, the complexity analysis of this latter algorithm reveals an upper bound in $O(\mathscr{D}.n)$ rounds on the stabilization time of the Couvreur *et al.*'s algorithm.

Awerbuch *et al.* [AK93] gives a self-stabilizing asynchronous unison (called clock

synchronizer in their paper) that uses unbounded state space and stabilizes in $O(\mathscr{D})$ rounds. Another self-stabilizing asynchronous unison algorithm is presented in [DJ19]. It stabilizes in $O(n)$ rounds using unbounded local memories. Emek and Keren [EK21] come up with a self-stabilizing asynchronous unison that stabilizes in $O(\mathtt{D}^3)$ rounds, where $\mathtt{D}$ is an upper bound on $\mathscr{D}$ known by all processes. Their solution requires $O(\log(\mathtt{D}))$ bits per processes.

Notice that most of the aforementioned asynchronous unison algorithms for general connected graphs are not suited for directed networks since they use reset mechanism that inherently need bidirectional links for synchronization purposes.

1.5 Roadmap

The rest of the paper is organized as follows. Section 2 is dedicated to basic definitions and computational model. In particular, the section includes the definition of hierarchical collateral composition and some of its properties. We present in Section 3 our basic building block, Algorithm $\texttt{C-MAI}(\mathcal{D}, \prec, \mathcal{I})$. In Section 4, we demonstrate our necessary and sufficient condition for the self-stabilizing leader election, in particular we propose our silent self-stabilizing leader election algorithm, Algorithm PEL. In Section 5, we propose our self-stabilizing synchronous unison algorithm, Algorithm $\mathtt{U}(K)$. This algorithm allows us to conclude that our condition on network topologies is also necessary and sufficient for the self-stabilization of the synchronous unison in our settings. In Section 6, we make concluding remarks and perspectives.

2 Preliminaries

2.1 Graph Definitions

Let $\mathcal{G}$ be a directed graph (digraph for short). We denote by $V(\mathcal{G})$ (resp. $E(\mathcal{G})$) the node set (resp. the arc set) of $\mathcal{G}$. Let p be a node. We denote by $\Gamma^-_{\mathcal{G}}(p)$ the set of p's predecessors in $\mathcal{G}$. A node q is an *ancestor* of p in $\mathcal{G}$ if there is a path in $\mathcal{G}$ from q to p (*n.b.*, by definition, there is an empty path from p to p, so p is one of its own ancestors). We denote by $\mathcal{A}nc_{\mathcal{G}}(p)$ the set of p's ancestors in $\mathcal{G}$. The *distance* from any node q to p in $\mathcal{G}$, denoted by $\|q,p\|_{\mathcal{G}}$, is the length of the shortest (directed) path from q to p in $\mathcal{G}$. By convention, we let $\|q,p\|_{\mathcal{G}} = \infty$ if there is no path in $\mathcal{G}$ from q to p. Let $MAD_{\mathcal{G}} = \max\{\|q,p\|_{\mathcal{G}} \mid p \in V(\mathcal{G}) \wedge q \in \mathcal{A}nc_{\mathcal{G}}(p)\}$ be the *Maximum Ancestor Distance*. Notice that $MAD_{\mathcal{G}} \leq |V(\mathcal{G})| - 1$. In the sequel, we systematically omit the subscript $\mathcal{G}$ when it is clear from the context.

$\mathcal{G}$ is *strongly connected* if $\forall p, q \in V(\mathcal{G})$, there is a directed path in $\mathcal{G}$ from p to q. Let $S \subseteq V(\mathcal{G})$. We denote by $\mathcal{G}(S)$ the subgraph of $\mathcal{G}$ induced by S. $\mathcal{G}(S)$ is a *strongly connected component* of $\mathcal{G}$ if $\mathcal{G}(S)$ is strongly connected and S is maximal, meaning that

the subgraph induced by any proper superset of S is not strongly connected. We call *source component* of $\mathcal{G}$ any strongly connected component SC of $\mathcal{G}$ where all nodes have no predecessor out of the component, namely $\forall p \in V(SC), \forall q \in \Gamma^-_{\mathcal{G}}(p), q \in V(SC)$. Any non-empty digraph has at least one source component.

2.2 Distributed Systems

We consider distributed systems made of $n \geq 1$ interconnected processes running on the top of a communication *network* conveniently modeled as a digraph $\mathcal{N}$, where $V(\mathcal{N})$ represents the n processes and $E(\mathcal{N})$ represents the direct *unidirectional* communication links connecting processes. We assume processes are endowed with unique constant (*i.e.*, immutable) *identifiers* and a common constant input value $\alpha \in \mathbb{N}$ such that $\alpha \geq MAD$. The identifier of any process p is denoted by $p.\mathcal{I}d$ and belongs to an arbitrary domain denoted by $IDSET$, ordered by $<_I$, and satisfying $|IDSET| > |V(\mathcal{N})|$.

2.3 Computational Model

We consider the *atomic-state model* introduced by Dijkstra [Dij74] in which processes communicate using locally shared registers, called *variables*. Each variable named v at a given process p will be denoted by $p.v$. Each process can read its own variables and those of its predecessors but they can only write its own variables. The *state* of a process is defined as the value of its variables (including its inputs). A *configuration* of the system is a vector consisting of the states of each process. Let $\mathcal{C}$ be the set of configurations of the system. For every configuration $\gamma \in \mathcal{C}$, we denote by $\gamma(p)$ the state of p in γ. Moreover, we denote by $\gamma(p).v$ the value of the variable $p.v$ in $\gamma(p)$.

Processes run according to a (deterministic) *distributed algorithm*. A distributed algorithm is a collection of n local programs, one per process. The *local program* of a process p consists of the set of its variables and a set of actions modifying them. Each action of p is of the following form: $\langle label \rangle :: \langle guard \rangle \rightarrow \langle statement \rangle$. *Labels* are only used to identify actions in the reasoning. The *guard* is a boolean predicate involving variables of p and its predecessors. The *statement* is a sequence of assignments modifying process variables. We exclude trivial actions that do not modify the state of the executing process. An action can be executed only if its guard evaluates to *true*; in which case, the action is said to be *enabled*. A process is said to be enabled if at least one of its actions is enabled. Recall that the exact values of inputs are not pre-defined, only their requirements are *a priori* known (*e.g.*, identifiers are constant, unique, and belong to $IDSET$). Consequently, a distributed algorithm should be insensitive to its inputs meaning that it should correctly operate as long as the input variables satisfy their requirements.

Let $\gamma_i \in \mathcal{C}$. We denote by $Enabled(\gamma_i)$ the set of enabled processes in configuration γ_i. If $Enabled(\gamma_i) = \emptyset$, then γ_i is said to be *terminal*. Otherwise, a *step* is performed as follows: a non-empty subset S of $Enabled(\gamma_i)$ is nondeterministically activated by an adversary called *daemon*. Then, every process p in S atomically executes one of its enabled actions in γ_i, leading the system to a new configuration γ_{i+1}. The step from γ_i to γ_{i+1} is denoted by $\gamma_i \mapsto \gamma_{i+1}$: $\mapsto$ is the binary relation over the configurations. An *execution* (of the distributed algorithm in the network) is a maximal sequence $e = \gamma_0\gamma_1 \ldots \gamma_i \ldots$ of configurations such that $\gamma_{i-1} \mapsto \gamma_i$ (and so, $\gamma_{i-1} \neq \gamma_i$) for all $i > 0$. The term *maximal* means that the execution e is either infinite or ends at a terminal configuration. We mainly consider two daemons: the *distributed unfair* daemon (the most general one) and the *synchronous* daemon. A distributed daemon chooses at least one process (maybe more) at each step. An unfair daemon might never select a process, unless it is the only enabled one. The synchronous daemon activates all enabled processes at each step of the execution. When an execution satisfies the requirements of the daemon D, we said that it is *an execution under* D.

2.4 Time Complexity

To measure the time complexity, we use the notion of *round*. A round computes the execution time according to the speed of the slowest processes. The definition of round uses the concept of *neutralization*. If a process p is enabled in a configuration γ_i but not enabled in γ_{i+1} and does not execute any action between γ_i and γ_{i+1}, then p is said to be neutralized during the step $\gamma_i \mapsto \gamma_{i+1}$. Neutralization of p is caused by the following situation: at least one predecessor of p changes its state between γ_i and γ_{i+1}, and this change makes the guards of all actions of p false. Then, rounds are inductively defined as follows. The first round of an execution $e = \gamma_0\gamma_1 \ldots$ is its minimal prefix e' such that every process that is enabled in γ_0 either executes an action or is neutralized during a step of e'. If e' is finite, then the second round of e is the first round of the suffix $\gamma_t\gamma_{t+1} \ldots$ of e starting from the last configuration γ_t of e', and so on and so forth. Notice that, under the synchronous daemon, rounds coincide with steps.

2.5 Self-stabilization and Silence

Below, we define a specification as a predicate over sequences of configurations.

Definition 1 (Self-Stabilization) *An algorithm* A *is self-stabilizing in the network $\mathcal{N}$ under the daemon D for a specification SP if there exists a non-empty subset of configurations, called the legitimate configurations, satisfying the following two conditions:*

Convergence: *Starting from any configuration, every execution of* A *in $\mathcal{N}$ under D contains a legitimate configuration.*

Partial correctness: *Every execution of* $\mathtt{A}$ *in* $\mathcal{N}$ *under* D *that starts from a legitimate configuration satisfies* SP.

In the atomic-state model, an algorithm is *silent* in the network $\mathcal{N}$ under the daemon D if all its executions in $\mathcal{N}$ under D are finite [DGS99, ADDP19].

When an algorithm is both self-stabilizing and silent, the specification can be reformulated as a predicate over configurations since this algorithm eventually reaches a terminal configuration; see [ADDP19] for a detailed explanation.

Definition 2 (Silent Self-Stabilization) *Let* $Pred$ *be a predicate over configurations. An algorithm* $\mathtt{A}$ *is silent self-stabilizing in the network* $\mathcal{N}$ *under the daemon* D *for* $Pred$ *if*

Termination: *starting from any configuration, every execution of* $\mathtt{A}$ *in* $\mathcal{N}$ *under* D *is finite, and*

Partial Correctness: *every terminal configuration satisfies* $Pred$.

2.6 Hierarchical Collateral Composition

Below, we recall the definition of *hierarchical collateral composition* introduced in [DLD$^+$13].

Definition 3 (Hierarchical Collateral Composition) *Let* $\mathtt{A}$ *and* $\mathtt{B}$ *be two distributed algorithms. The hierarchical collateral composition of* $\mathtt{A}$ *and* $\mathtt{B}$ *is the distributed algorithm* $\mathtt{B} \circ \mathtt{A}$*, where the local algorithm of every process* p*, noted (*$\mathtt{B} \circ \mathtt{A}$*)(*$p$*), is defined as follows:*

1. *(*$\mathtt{B} \circ \mathtt{A}$*)(*$p$*) contains all variables of* $\mathtt{A}(p)$ *and* $\mathtt{B}(p)$*,*[3]
2. *(*$\mathtt{B} \circ \mathtt{A}$*)(*$p$*) contains all actions of* $\mathtt{A}(p)$*, and*
3. *every action* $L_i :: G_i \rightarrow S_i$ *of* $\mathtt{B}(p)$ *is rewritten in (*$\mathtt{B} \circ \mathtt{A}$*)(*$p$*) as the action*

$$L_i :: \neg C_p \wedge G_i \rightarrow S_i$$

where C_p *is the disjunction of all guards of all actions in* $\mathtt{A}(p)$.

By convention, $\circ$ *is* right-associative.

[3] *N.b.* some variables may be common to $\mathtt{A}(p)$ and $\mathtt{B}(p)$, *e.g.*, some outputs of $\mathtt{A}(p)$ may be inputs in $\mathtt{B}(p)$.

Let $\mathtt{A}_0$ and $\mathtt{A}_1$ be two distributed algorithms. Let γ be a configuration of $\mathtt{A}_1 \circ \mathtt{A}_0$. The $\mathtt{A}_i$-projection $\gamma_{|\mathtt{A}_i}$ (with $i \in \{0,1\}$) is the configuration of $\mathtt{A}_i$ obtained by removing from γ the values of all variables that do not exist in $\mathtt{A}_i$. Let $s = \gamma_0 \ldots \gamma_j \ldots$ be a sequence of configurations of $\mathtt{A}_1 \circ \mathtt{A}_0$. By extension, we also call $\mathtt{A}_i$-projection the sequence of configurations $\gamma_{0|\mathtt{A}_i} \ldots \gamma_{j|\mathtt{A}_i} \ldots$ (with $i \in \{0,1\}$). We also denote by $\mathcal{SQ}(s)$ the maximal subsequence of s where no two consecutive configurations are identical.

Algorithms have to deal with input variables (inputs for short). Inputs are not written by the algorithm and are directly encoded in local states. For example, in our model, identifiers and α are inputs. There may have requirements on the values of inputs, *e.g.*, identifiers are immutable and unique. Of course, all executions are assumed to satisfy all such requirements. In the case of the composition $\mathtt{B} \circ \mathtt{A}$ of two silent algorithms A and B, some inputs of B may be computed by A. Of course, if one considers the execution of B alone, inputs coming from A are constant (by definition, B cannot modify them). The following two results give properties for the composition $\mathtt{B} \circ \mathtt{A}$ in cases where B has no requirement on the value of its inputs provided by A.

Lemma 1 *Let* A *and* B *be two algorithms that are silent in the network $\mathcal{N}$ under a distributed unfair daemon such that (1) no variable written by* B *appears in* A *and (2)* B *has no requirement on the values of its inputs written by* A*. Then,* $\mathtt{B} \circ \mathtt{A}$ *is silent in $\mathcal{N}$ under a distributed unfair daemon.*

Proof. Let e be any execution of $\mathtt{B} \circ \mathtt{A}$ in $\mathcal{N}$ under a distributed unfair daemon. By *(1)*, $\mathcal{SQ}(e_{|\mathtt{A}})$ is an execution prefix of A. Since A is silent under a distributed unfair daemon, all executions of A under a distributed unfair daemon are finite, and so are their prefix. Thus, e has a suffix e' containing no execution of any action of A: only actions of B can be executed in e'. Moreover, by *(2)*, B has no requirement on the values of its inputs that may have been modified by A, hence all requirements of B for its inputs are satisfied in $e'_{|\mathtt{B}}$ and, thus, $e'_{|\mathtt{B}}$ is an execution prefix of B in $\mathcal{N}$ under a distributed unfair daemon. Again, since B is silent under a distributed unfair daemon, $e'_{|\mathtt{B}}$ is finite and so e' is (indeed, $e'_{|\mathtt{B}}$ and e' have the same length, by definition). Hence, e is finite, and we are done. □

Corollary 1 *Let* A *and* B *be two algorithms that are silent in the network $\mathcal{N}$ under a distributed unfair daemon. If (1) no variable written by* B *appears in* A*, (2)* B *has no requirement on the values of its inputs written by* A*, (3)* A *reaches a terminal configuration in $\mathcal{N}$ in at most R_A rounds, and (4)* B *reaches a terminal configuration in $\mathcal{N}$ in at most R_B rounds, then* $\mathtt{B} \circ \mathtt{A}$ *reaches a terminal configuration in $\mathcal{N}$ in at most $R_A + R_B$ rounds.*

Proof. Let $e = \gamma_0 \ldots$ be any execution of $\mathtt{B} \circ \mathtt{A}$ in $\mathcal{N}$. By *(1)*, *(2)*, and Lemma 1, e is finite. Let $e' = \gamma_i \ldots$ be the maximum suffix of e containing no execution of any action of

A. By *(2)*, B has no requirement on the values of its inputs that may have been modified by A, so $e'_{|\mathtt{B}}$ is an execution of B in $\mathcal{N}$ under a distributed unfair daemon. Hence, by *(4)*, $e'_{|\mathtt{B}}$ reaches a terminal configuration in at most R_B rounds, and so e' does. By construction, for every $\gamma_j \in e$ and every process p, p is enabled for an action (of A) in $\gamma_{j|\mathtt{A}}$ if and only if p is enabled for the same action in γ_j. Moreover, by *(1)*, B does not modify A's variables, so γ_i is reached in e within at most as many rounds as $\gamma_{i|\mathtt{A}}$ in $\mathcal{SQ}(\gamma_{0|\mathtt{A}} \dots \gamma_{i|\mathtt{A}})$ which is an execution prefix of A, *i.e.*, at most R_A rounds by *(3)*, and we are done. □

3 Minimum Ancestors' Input

In this section, we propose a generic silent self-stabilizing algorithm that computes at each process the minimum value among the inputs of all its ancestors. Actually, our silent self-stabilizing leader election algorithm will consist of a composition of three different instances of this basic building block.

3.1 The Problem

Let $\mathcal{D}$ be an arbitrary domain totally ordered by $\prec$. We assume that each process p has a constant input $\mathcal{I}(p)$, where $\mathcal{I} : V(\mathcal{N}) \to \mathcal{D}$. Our goal is to make every process p compute into an output variable $p.M$ the smallest input according to $\prec$ among those of its ancestors (including p itself); let $MinI(p) = \min_{\prec}\{\mathcal{I}(q) \mid q \in \mathcal{A}nc(p)\}$ be this input value. We additionally require each process to compute in another output variable, $p.d$, the minimum distance $Mind(p)$ from an ancestor having $MinI(p)$ as input. That is, $p.d$ should be set to $Mind(p) = \min\{\|q, p\| \mid \mathcal{I}(q) = MinI(p) \wedge q \in \mathcal{A}nc(p)\}$.

Overall, we require the system to eventually reach a configuration from which each process p forever satisfies the predicate $MinOk(p)$, where $MinOk(p) \equiv p.M = MinI(p) \wedge p.d = Mind(p)$. We call this problem the *minimum ancestors' input problem*.

To be self-stabilizing for the minimum ancestors' input problem, our algorithm will have to deal with what we call *fake values*. A fake value for a process p (or simply, fake value when p is clear from the context) is any value $m \in \mathcal{D} \setminus \{\mathcal{I}(q) \mid q \in \mathcal{A}nc(p)\}$, *i.e.*, any value (of the domain) which does not appear as an input at any p's ancestor.

3.2 The Algorithm

In Algorithm 1, we give the code of an algorithm that is silent and self-stabilizing for the minimum ancestors' input problem in an arbitrary anonymous network $\mathcal{N}$ (*n.b.*, $\mathcal{N}$ may even not be weakly connected). This algorithm is generic in the sense that it works given

any set $\mathcal{D}$ totally ordered by an operator $\prec$ and any constant returned by any function $\mathcal{I} : V(\mathcal{N}) \to \mathcal{D}$. In the following, we refer to this algorithm as C-MAI$(\mathcal{D}, \prec, \mathcal{I})$.

We first present a non self-stabilizing solution to the minimum ancestors' input problem and then explain how to gradually make it silent and self-stabilizing in order to finally obtain Algorithm C-MAI$(\mathcal{D}, \prec, \mathcal{I})$. In this non-stabilizing algorithm, each process p maintains two variables:

- $p.M$, a variable whose domain is $\mathcal{D}$.

  In this variable, p computes the minimum input at p's ancestors.

- $p.d \in \mathbb{N}$.

  In this variable, p computes the distance from a closest ancestor having an input equal to $p.M$.

The algorithm also uses the following two macros:

- Let $p.\mathcal{K} = (p.M, p.d)$ be the *key* of p.

  Keys are (totally) ordered by $\lhd$ following the lexicographic order: $\forall (a, b), (c, d) \in \mathcal{D} \times \mathbb{N}, (a, b) \lhd (c, d) \equiv a \prec c \vee (a = c \wedge b < d)$.

  In the following, $p.\mathcal{K}$ is said to be a *fake key* when $p.M$ is a fake value for p.

- Let $SelfKey(p) = (\mathcal{I}(p), 0)$ be the *selfkey* of p.

Assume initially $p.\mathcal{K} = SelfKey(p)$, for every process p. Then, each process p aims at minimizing its key according to its selfkey and the keys of its predecessors, if any. If p has no predecessor, its computation is already done. Otherwise, let q be the predecessor of p with the smallest key according to $\lhd$. If $p.\mathcal{K} \rhd (q.M, q.d + 1)$, then $p.\mathcal{K}$ is set to $(q.M, q.d + 1)$. Indeed, in this case, there is an ancestor of q, and so of p, with input $q.M$ which is at distance at most $q.d$ from q, and so at distance at most $q.d + 1$ from p. Using this approach, a terminal configuration satisfying $MinOk(p)$ for every process p is reached within at most MAD rounds; see Figure 1 for an illustrative example.

Unfortunately, this simple algorithm is not self-stabilizing for the minimum ancestors' input problem. Indeed, in an arbitrary initial configuration, the M-variable of some process p may contain a fake value m. Now, if m is smaller than all p's ancestors inputs, the system converges to a terminal configuration where $p.M \neq MinI(p)$; see Figure 2 for an illustrative example. To solve this issue, we can modify the algorithm so that $p.\mathcal{K}$ is computed as the minimum key between the selfkey of p and the keys of its predecessors with their distance increased by one. However, this time we can have a livelock as illustrated in Figure 3. Now, in this case, we can remark that distance in the keys involved into the

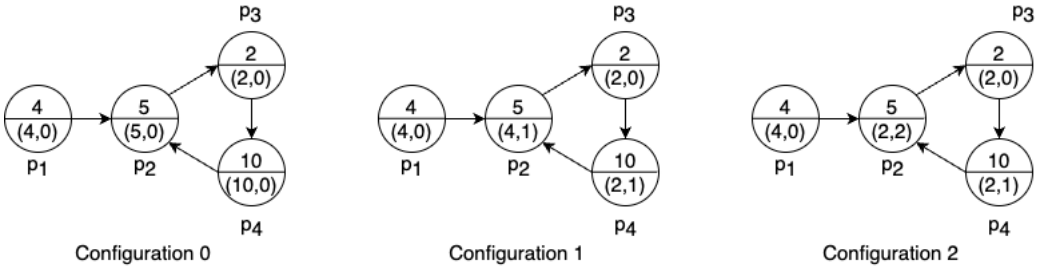

Figure 1: Example of execution of our non self-stabilizing algorithm with arbitrary integer inputs. Inside each circle, we give the input and the key of the process (*n.b.*, here $MAD = 3$).

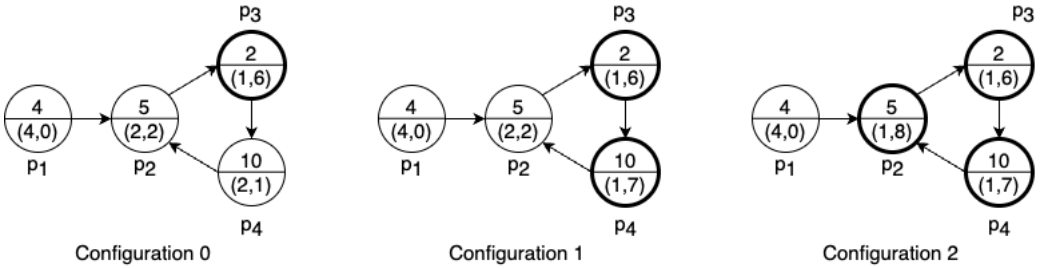

Figure 2: Example of execution of our non self-stabilizing algorithm with arbitrary integer inputs that terminates in an illegitimate configuration. Inside each circle, we give the input and the key of the process. Bold circles indicate processes with fake keys.

livelock increase infinitely often. Such a livelock can be easily stopped by forbidding a process to update its $\mathcal{K}$-variable with a key whose distance is greater than α; indeed $\alpha \geq MAD$.

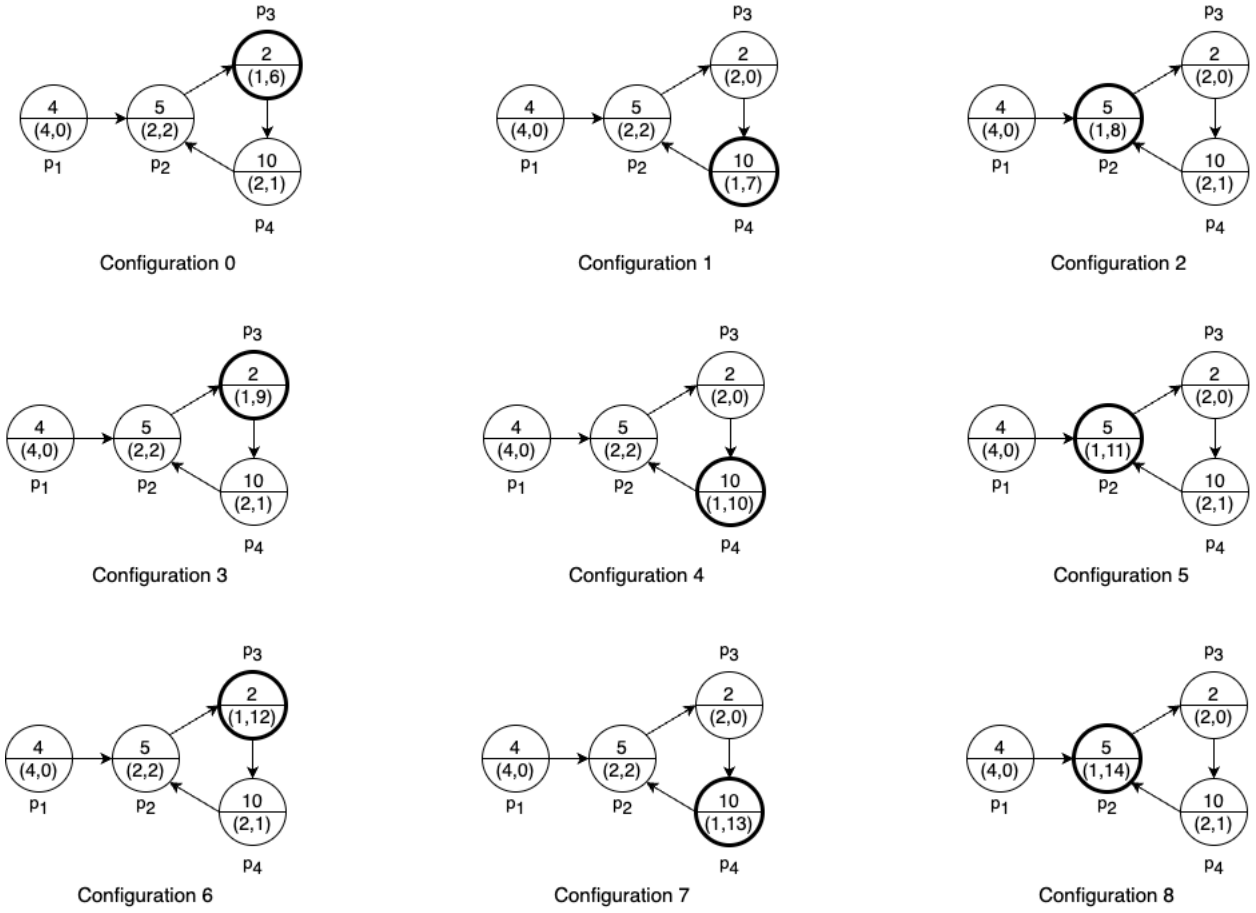

Figure 3: Sample of non-terminating execution of the second version of our non self-stabilizing algorithm with arbitrary integer inputs. Inside each circle, we give the input and the key of the process. Bold circles indicate processes with fake keys.

Hence, we obtain the code given in Algorithm 1. Let p be any process. In this silent self-stabilizing algorithm, the domain of $p.d$ is restricted to $[0..\alpha]$ and the action $ReplaceKey$ consists in computing in $p.\mathcal{K}$ the minimum value (according to $\lhd$) among the selfkey of p ($SelfKey(p)$) and the keys of its predecessors with their distance increased by one, but excluding those reaching the value α; see Macro $BestKey(p)$, and Figure 4 for an illustrative example.

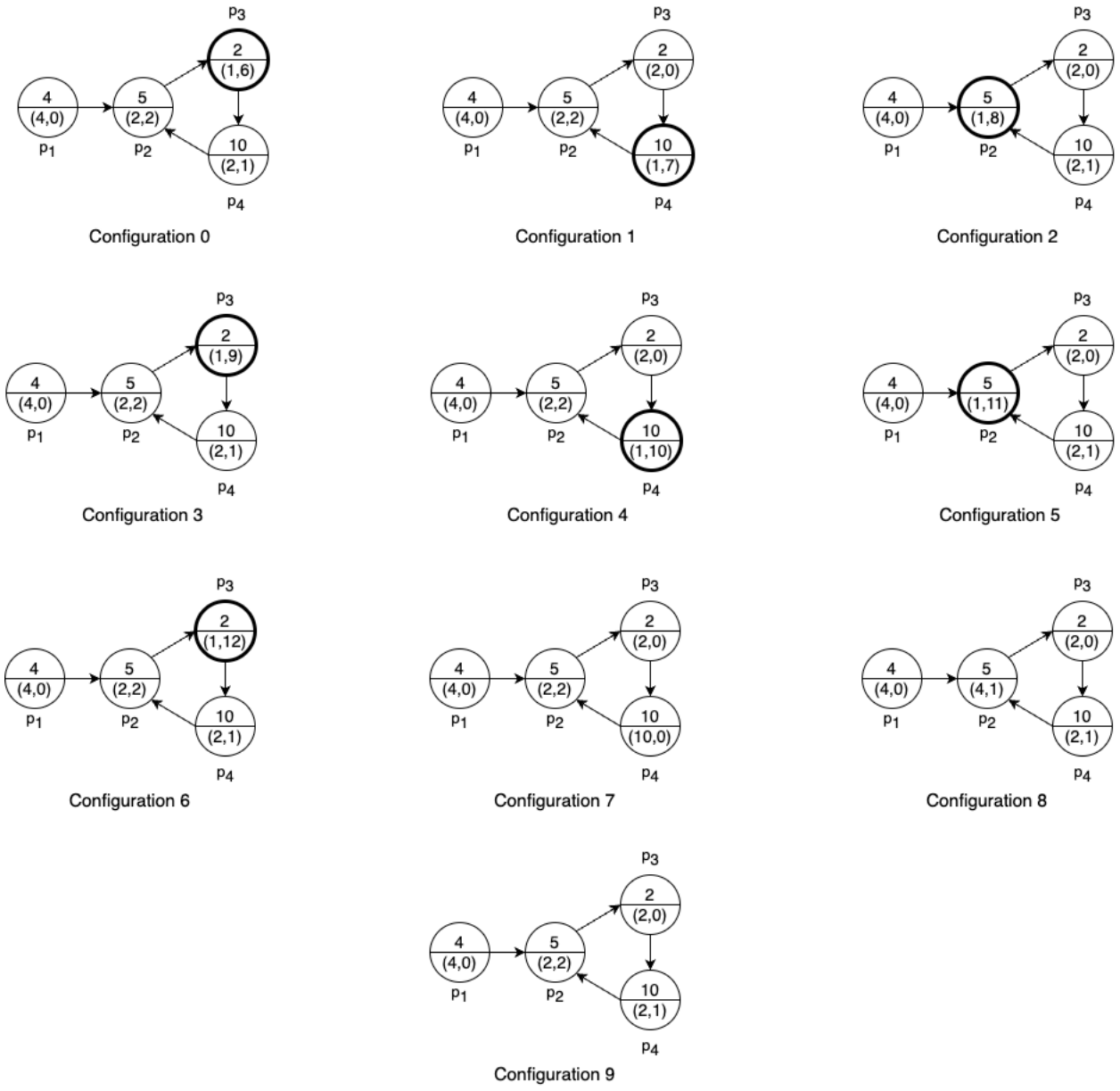

Figure 4: Example of execution of C-MAI$(\mathbb{N}, <, \mathcal{I})$ where $\mathcal{I}$ is an arbitrary function from $V(\mathcal{N})$ to $\mathbb{N}$. We assume $\alpha = 12$ (*n.b.*, $MAD = 3$). Inside each circle, we give the input and the key of the process. Bold circles indicate processes with fake keys.

Algorithm 1: Algorithm C-MAI$(\mathcal{D}, \prec, \mathcal{I})$, code for any process p.

Inputs:

$\mathcal{I}(p) \in \mathcal{D}$	:	a constant
$\Gamma^-(p)$	:	the set of p's predecessors
α	:	a natural number satisfying $\alpha \geq MAD$

Variable:

$p.M \in \mathcal{D}$	:	the estimated minimum ancestors' input
$p.d \in [0..\alpha]$	:	the estimated distance from an input equal to $p.M$

Macros:

$p.\mathcal{K}$	=	$(p.M, p.d)$
$SelfKey(p)$	=	$(\mathcal{I}(p), 0)$
$KeySet(p)$	=	$\{SelfKey(p)\} \cup \{(q.M, q.d+1) \mid q \in \Gamma^-(p) \wedge q.d < \alpha\}$
$BestKey(p)$	=	$\min_{\lhd}(KeySet(p))$

Predicate:

$Enhance(p) \equiv BestKey(p) \neq p.\mathcal{K}$

Action:

$ReplaceKey :: Enhance(p) \rightarrow p.\mathcal{K} \leftarrow BestKey(p)$

3.3 Proof of Silent Self-stabilization

In this subsection, we prove two main lemmas: Lemma 3 (partial correctness) and Lemma 4 (termination). From these two lemmas, we immediately obtain Theorem 1 below which establishes the correctness of our algorithm.

Theorem 1 *Algorithm* C-MAI$(\mathcal{D}, \prec, \mathcal{I})$ *is silent self-stabilizing in any directed network* $\mathcal{N}$ *under the distributed unfair daemon for the specification predicate:* $\forall p \in V(\mathcal{N}), MinOk(p)$.

Let $e = \gamma_0\gamma_1\ldots$ be any execution of Algorithm C-MAI$(\mathcal{D}, \prec, \mathcal{I})$. In the following, we will denote by $BestKey_i(p)$ (resp. $KeySet_i(p)$) the value of $BestKey(p)$ (resp. $KeySet(p)$) in the configuration γ_i, for every $i \geq 0$.

Partial Correctness. Let γ_t be any terminal configuration of Algorithm C-MAI$(\mathcal{D}, \prec, \mathcal{I})$ in the network $\mathcal{N}$. The following lemma is an intermediate result which establishes that the M-variable of any process p is not underestimated in γ_t.

Lemma 2 $\forall p \in V(\mathcal{N}), \gamma_t(p).\mathcal{K} \trianglerighteq (MinI(p), Mind(p))$.

Proof. Assume by contradiction that $\exists p \in V(\mathcal{N}), \gamma_t(p).\mathcal{K} \lhd (MinI(p), Mind(p))$. Let p_{min} be a process such that $\gamma_t(p_{min}).\mathcal{K} \lhd (MinI(p_{min}), Mind(p_{min}))$ where $p_{min}.\mathcal{K}$ is

minimum. So, $SelfKey(p_{min}) \trianglerighteq (MinI(p_{min}), Mind(p_{min})) \rhd \gamma_t(p_{min}).\mathcal{K}$. Then, $\forall q \in \Gamma^-(p_{min})$ such that $\gamma_t(q).d < \alpha$, q satisfies one of the following two cases.

- $\gamma_t(q).\mathcal{K} \trianglerighteq (MinI(p_{min}), Mind(p_{min}))$. So, $(\gamma_t(q).M, \gamma_t(q).d+1) \rhd (MinI(p_{min}), Mind(p_{min})) \rhd \gamma_t(p_{min}).\mathcal{K}$
- $\gamma_t(q).\mathcal{K} \lhd (MinI(p_{min}), Mind(p_{min}))$. In this case, by definition of p_{min}, $\gamma_t(q).\mathcal{K} \trianglerighteq \gamma_t(p_{min}).\mathcal{K}$, which implies $(\gamma_t(q).M, \gamma_t(q).d + 1) \rhd \gamma_t(p_{min}).\mathcal{K}$

By the above two cases, we can conclude that $(\gamma_t(q).M, \gamma_t(q).d + 1) \rhd \gamma_t(p_{min}).\mathcal{K}$, for any predecessor q of p_{min} such that $\gamma_t(q).d < \alpha$. From this latter claim and owing the fact that $SelfKey(p_{min}) \rhd \gamma_t(p_{min}).\mathcal{K}$, we can deduce that $BestKey_t(p_{min}) \rhd \gamma_t(p_{min}).\mathcal{K}$. Consequently, $ReplaceKey$ is enabled in γ_t, a contradiction. □

Now, we show that γ_t is legitimate, *i.e.*, every process p satisfies $MinOk(p)$ in γ_t. To that goal, we will use the property on distances given below.

Property 1 $\forall q \in \Gamma^-(p), (MinI(q), Mind(q)) \trianglerighteq (MinI(p), Mind(p) - 1))$.

Proof. Let $q \in \Gamma^-(p)$. By definition, $q \in \mathcal{A}nc(p)$ and so $MinI(q) \succeq MinI(p)$. If $MinI(q) \succ MinI(p)$, then $(MinI(q), Mind(q)) \rhd (MinI(p), Mind(p)-1))$. Otherwise, $MinI(q) = MinI(p)$ and $Mind(q) \geq Mind(p) - 1$ and so $(MinI(q), Mind(q)) \trianglerighteq (MinI(p), Mind(p) - 1))$. □

Lemma 3 *For every process p, $MinOk(p)$ holds in γ_t.*

Proof. To show this lemma, we now prove by induction on i that $\forall i \in \mathbb{N}$, for every process p such that $Mind(p) = i$, we have $\gamma_t(p).\mathcal{K} = (MinI(p), Mind(p))$.

Base Case. Let p be any process such that $Mind(p) = 0$. So, $MinI(p) = \mathcal{I}(p)$. By Lemma 2, we have $\gamma_t(p).\mathcal{K} \trianglerighteq (MinI(p), Mind(p)) = (\mathcal{I}(p), 0) = SelfKey(p)$. Assume by contradiction that $\gamma_t(p).\mathcal{K} \rhd SelfKey(p)$. So, $\gamma_t(p).\mathcal{K} \rhd BestKey_t(p)$. Thus, p is enabled, a contradiction. Consequently, $\gamma_t(p).\mathcal{K} = (MinI(p), Mind(p))$.

Induction Step. Let p be process such that $Mind(p) = i + 1$. We have $0 < i + 1 \leq MAD \leq \alpha$. Then, there exists a predecessor q of p such that $(MinI(q), Mind(q)) = (MinI(p), Mind(p) - 1) = (MinI(p), i)$. By induction hypothesis, we have $\gamma_t(q).\mathcal{K} = (MinI(q), Mind(q)) = (MinI(p), i)$ (*n.b.*, this implies that $\gamma_t(q).d < \alpha$). So, $(\gamma_t(q).M, \gamma_t(q).d+1) = (MinI(p), i + 1)$.

By Lemma 2 and Property 1, for any predecessor q' of p, we have $\gamma_t(q').\mathcal{K} \trianglerighteq (MinI(q'), Mind(q'))$ $\trianglerighteq (MinI(p), Mind(p)-1)$. Consequently, $(MinI(q'), Mind(q')+1) \trianglerighteq (MinI(p), Mind(p)) = (MinI(p), i+1)$.

Finally, by definition $SelfKey(p) \triangleright (MinI(p), Mind(p)) = (MinI(p), i+1)$. Hence, $BestKey_t(p) = (MinI(p), i+1) = (MinI(p), Mind(p))$. Since, γ_t is terminal, $\gamma_t(p).\mathcal{K} = BestKey_t(p) = (MinI(p), Mind(p))$, and we are done.

□

Termination. To show that Algorithm C-MAI$(\mathcal{D}, \prec, \mathcal{I})$ is silent self-stabilizing, it remains to show that all its executions are finite. To that goal, we use the notion of *limit inferior* of an infinite sequence $x_0 \dots x_i \dots$, denoted by $\liminf_{i\to\infty} x_i$ and defined as follows: $\liminf_{i\to\infty} x_i = \lim_{i\to\infty}(\inf_{j\geq i} x_j)$. Remark that if the domain of x_i is finite, then the limit inferior corresponds to the minimum value among those that appear infinitely often in the sequence.

Lemma 4 *Every execution of Algorithm* C-MAI$(\mathcal{D}, \prec, \mathcal{I})$ *in any network* $\mathcal{N}$ *under the distributed unfair daemon is finite.*

Proof. Let $e = \gamma_0\gamma_1 \dots$ be an execution of Algorithm C-MAI$(\mathcal{D}, \prec, \mathcal{I})$ in $\mathcal{N}$ under the distributed unfair daemon. Assume, by contradiction, that e is infinite, meaning that Action $ReplaceKey$ is executed infinitely many times in e. For every process p, let $\lambda(p) = \liminf_{i\to\infty} \gamma_i(p).\mathcal{K}$ be the limit inferior of the sequence of values $\gamma_0(p).\mathcal{K}\ \gamma_1(p).\mathcal{K} \dots$

Let $S = \{\mathcal{I}(q) \mid q \in V(\mathcal{N})\} \cup \{\gamma_0(q).M \mid q \in V(\mathcal{N})\}$, *i.e.*, the set containing the initial values of M-variables and the input values of all processes. By definition of Algorithm C-MAI$(\mathcal{D}, \prec, \mathcal{I})$, the set of possible values that M-variables can take during e belong to S. By definition, $|S| \leq 2n$ (recall that n is the number of process in $\mathcal{N}$). Moreover, by definition of our algorithm, $\forall i \geq 0, \forall p \in V(\mathcal{N}), \gamma_i(p).\mathcal{K} \in S \times [0..\alpha]$. Now, $|S \times [0..\alpha]\ | \leq 2n \times (\alpha + 1)$. Consequently, the number of distinct keys any process can take during e is bounded. So, $\forall p \in V(\mathcal{N})$, $\lambda(p)$ exists and $\lambda(p) \in S \times [0..\alpha]$.

Let IV be the subset of $V(\mathcal{N})$ such that for every p in IV, p modifies $p.\mathcal{K}$ infinitely often by executing Action $ReplaceKey$ in e. Since e is infinite and $V(\mathcal{N})$ is finite, $IV \neq \emptyset$.

Let p be a process in IV. Let q be an ancestor of p such that $q \in IV$ and $\lambda(q)$ is minimum. Since $p \in \mathcal{A}nc(p) \wedge p \in IV$, $IV \cap \mathcal{A}nc(p) \neq \emptyset$. So, q is well-defined. Since $q \in IV$, q executes Action $ReplaceKey$ infinitely often and we have the following two possible cases for $\lambda(q)$:

- $\lambda(q) = SelfKey(q)$. Since $q \in IV$, we have the following two facts:

1. There exist infinitely many configurations γ_i of e satisfying $q.\mathcal{K} \leftarrow \Bbbk_i$ with $\Bbbk_i \neq SelfKey(q)$ in $\gamma_i \mapsto \gamma_{i+1}$. From the algorithm, we can deduce that, in each of this configurations γ_i, $\exists x \in \Gamma^-(q)$ such that $(\gamma_i(x).M, \gamma_i(x).d + 1) \lhd SelfKey(q) \wedge \gamma_i(x).d < \alpha$, which implies that $\gamma_i(x).M \prec \mathcal{I}(q) \wedge \gamma_i(x).d < \alpha$. Hence, the predicate $P(q, i) = \exists x \in \Gamma^-(q), \gamma_i(x).M \prec \mathcal{I}(q) \wedge \gamma_i(x).d < \alpha$ holds in infinitely many configurations γ_i of e.
2. There exist infinitely many configurations γ_j of e such that $q.\mathcal{K} \leftarrow SelfKey(q)$ in $\gamma_j \mapsto \gamma_{j+1}$. From the algorithm, we can deduce that $\forall x \in \Gamma^-(q)$, $(\gamma_j(x).M, \gamma_j(x).d+1) \trianglerighteq SelfKey(q) \vee \gamma_j(x).d \geq \alpha$, which implies that $\gamma_j(x).M \succeq \mathcal{I}(q) \vee \gamma_j(x).d \geq \alpha$.
   Hence, $\neg P(q, j)$ holds in infinitely many configurations γ_j of e.

Since the number of predecessors of q is finite and the set of possible keys is bounded, we can deduce from the above two cases that there exists a predecessor x of q such that $x \in IV$ and $\lambda(x) \lhd SelfKey(q) = \lambda(q)$. Finally, since x is the predecessor of q, $x \in \mathcal{A}nc(p)$. Thus, $x \in IV, x \in \mathcal{A}nc(p)$, and $\lambda(x) \lhd \lambda(q)$, a contradiction.

- $\lambda(q) \neq SelfKey(q)$. There exist infinitely many configurations γ_i of e such that $q.\mathcal{K} \leftarrow \lambda(q)$ in $\gamma_i \mapsto \gamma_{i+1}$. From the algorithm, in every such a configuration γ_i, there exists $x \in \Gamma^-(q)$ such that $\gamma_i(x).\mathcal{K} = (\lambda(q).M, \lambda(q).d - 1)$ with $\lambda(q).d - 1 < \alpha$. Since the number of predecessors of q is finite, there exists $y \in \Gamma^-(q)$ such that $y.\mathcal{K} = (\lambda(q).M, \lambda(q).d - 1) \wedge \lambda(q).d - 1 < \alpha$ in infinitely many configurations of e. So, $\lambda(y) \lhd \lambda(q)$ and, by definition of q, $y \notin IV$. Thus, eventually $y.\mathcal{K} = (\lambda(q).M, \lambda(q).d-1) \wedge \lambda(q).d-1 < \alpha$ forever. From this moment, $BestKey(q) \trianglelefteq \lambda(q)$ forever and since $q \in IV$, eventually $q.\mathcal{K} \trianglelefteq \lambda(q)$ forever. Now, by definition of $\lambda(q)$, eventually $q.\mathcal{K} \trianglerighteq \lambda(q)$ forever. Hence, eventually $q.\mathcal{K} = \lambda(q)$ forever, implying that $q \notin IV$, a contradiction.

From the above two contradictions, we can deduce that every execution is finite and we are done.

□

3.4 Complexity Analysis

Given an arbitrary network $\mathcal{N}$, we now prove an upper bound on the stabilization time in rounds of Algorithm C-MAI$(\mathcal{D}, \prec, \mathcal{I})$ (Theorem 2). To obtain this bound we proceed in two main steps. First, we show in Lemma 9 that after at most $\alpha + 1$ rounds there is no fake value anymore in $\mathcal{N}$. Then, Lemma 10 claims that starting from any configuration containing no

fake value, the system reaches a terminal configuration within at most $MAD + 1$ rounds. Hence, from these two lemmas, we immediately obtain the theorem below:

Theorem 2 *The stabilization time of Algorithm* C-MAI$(\mathcal{D}, \prec, \mathcal{I})$ *is at most* $\alpha + MAD + 2$ *rounds.*

In the remainder of the section, we consider an arbitrary execution (under the distributed unfair daemon) $e = \gamma_0 \dots \gamma_i \dots$ of C-MAI$(\mathcal{D}, \prec, \mathcal{I})$ in $\mathcal{N}$. By Lemma 4, e is finite. So, let t be the index of the last configuration of e.

The next four technical lemmas (Lemmas 5-8) allow to establish Lemma 9, which states that fake values disappear from $\mathcal{N}$ within at most $\alpha + 1$ rounds. Those lemmas use the following three definitions:

- Let $\mathcal{F}_i(p) = \{q \in \mathcal{A}nc(p) \mid \forall q' \in \mathcal{A}nc(q), \mathcal{I}(q') \neq \gamma_i(q).M\}$, with $i \in [0..t]$, be the set of p's ancestor holding a fake key in γ_i.
- Let $\mathcal{M}df_i(p) = \min(\{\gamma_i(q).d \mid q \in \mathcal{F}_i(p)\} \cup \{\infty\})$, with $i \in [0..t]$, be the smallest distance value stored in a fake key held in γ_i by an ancestor of p.

  Note that $\mathcal{M}df_i(p) = \infty$ if $\mathcal{F}_i(p)$ is empty. Hence, $\mathcal{M}df_i(p)$ is defined as long as $i \in [0..t]$.
- Let $\mathcal{B}ad_i^j(p, q) \equiv q \in \mathcal{F}_j(p) \wedge \gamma_j(q).d = \mathcal{M}df_i(p)$, with $i, j \in [0..t]$ and $j \geq i$, be the property that is true if in configuration γ_j, the key of the ancestor q of p consists of a fake value and a distance value d such that d was the smallest distance value stored in a fake key of an ancestor of p in γ_i.

The first technical lemma below (Lemma 5) shows that fake values cannot decrease along e.

Lemma 5 *Let p be any process.* $\forall i \in [0..t-1], \mathcal{M}df_{i+1}(p) \geq \mathcal{M}df_i(p)$.

Proof. Assume by contradiction that $\mathcal{M}df_{i+1}(p) < \mathcal{M}df_i(p)$ for some $i \in [0..t-1]$. By definition, $\exists q \in \mathcal{F}_{i+1}(q)$ such that $\gamma_{i+1}(q).d = \mathcal{M}df_{i+1}(p)$. We have two cases during the step $\gamma_i \mapsto \gamma_{i+1}$:

- q does not move, hence $\gamma_i(q).\mathcal{K} = \gamma_{i+1}(q).\mathcal{K}$. So, $\mathcal{M}df_{i+1}(p) = \gamma_{i+1}(q).d = \gamma_i(q).d \geq \mathcal{M}df_i(p)$, a contradiction.
- q moves, and we have two subcases:

  1. $\gamma_{i+1}(q).\mathcal{K} = SelfKey(q)$ so $q \notin \mathcal{F}_{i+1}(p)$, a contradiction.

2. $\gamma_{i+1}(q).\mathcal{K} = (\gamma_i(q').M, \gamma_i(q').d + 1)$ such that $q' \in \Gamma^-(q)$ and we have two possibilities:
   - $q' \notin \mathcal{F}_i(p)$, so $q \notin \mathcal{F}_{i+1}(p)$, a contradiction.
   - $q' \in \mathcal{F}_i(p)$, so $\gamma_{i+1}(q).d = \gamma_i(q').d + 1$. By definition, $\gamma_i(q').d \geq \mathcal{M}df_i(p)$ thus $\gamma_{i+1}(q).d = \gamma_i(q').d + 1 > \mathcal{M}df_i(p) > \mathcal{M}df_{i+1}(p)$. So, $\gamma_{i+1}(q).d \neq \mathcal{M}df_{i+1}(p)$, a contradiction.

□

Lemma 6 below claims that if two processes p and q verify the property $\mathcal{B}ad_i^j(p,q)$ for some $i, j \in [0..t]$ such that $j \geq i$, then we can conclude that q is enabled in γ_j.

Lemma 6 *Let $i, j \in [0..t]$ such that $j \geq i$. Let p, q be two processes. If $\mathcal{B}ad_i^j(p,q)$, then $q \in Enabled(\gamma_j)$.*

Proof. Let $i, j \in [0..t]$ such that $j \geq i$. Let p and q be two processes such that $\mathcal{B}ad_i^j(p,q)$. We have the following two claims:

Claim 1: $\gamma_j(q).\mathcal{K} \neq SelfKey(q)$.

Proof of the claim: Since $\mathcal{B}ad_i^j(p,q)$, $q \in \mathcal{F}_j(p)$, so $\gamma_j(q).M \neq \mathcal{I}(q)$. Consequently, $\gamma_j(q).\mathcal{K} \neq SelfKey(q)$.

Claim 2: $\forall q' \in \Gamma^-(q), \gamma_j(q).\mathcal{K} \neq (\gamma_j(q').M, \gamma_j(q').d + 1)$.

Proof of the claim: Let $q' \in \Gamma^-(q)$:

- If $q' \notin \mathcal{F}_j(p)$, then $\gamma_j(q).M \neq \gamma_j(q').M$. Consequently, $\gamma_j(q).\mathcal{K} \neq (\gamma_j(q').M, \gamma_j(q').d + 1)$.
- If $q' \in \mathcal{F}_j(p)$, then by definition, $\gamma_j(q').d \geq \mathcal{M}df_j(p)$. By Lemma 5, $\mathcal{M}df_j(p) \geq \mathcal{M}df_i(p)$. Moreover, since $\mathcal{B}ad_i^j(p,q)$ then, $\gamma_j(q).d = \mathcal{M}df_i(p)$. Thus, $\gamma_j(q).d < \mathcal{M}df_i(p) + 1 \leq \mathcal{M}df_j(p) + 1 \leq \gamma_j(q').d + 1$. Consequently, $\gamma_j(q).d \neq \gamma_j(q').d + 1$. Hence, $\gamma_j(q).\mathcal{K} \neq (\gamma_j(q').M, \gamma_j(q').d + 1)$.

By Claims 1 and 2, $\gamma_j(q).\mathcal{K} \neq BestKey_j(q)$. Hence, $q \in Enabled(\gamma_j)$.

□

Lemma 7 below states that if $\mathcal{B}ad_i^j(p,q)$ is verified by a process p and its ancestor q, and q moves during $\gamma_j \mapsto \gamma_{j+1}$, then $\mathcal{B}ad_i^{j+1}(p,q)$ is false.

Lemma 7 *Let p be any process. $\forall i, j \in [0..t-1]$ such that $j \geq i, \forall q \in \mathcal{A}nc(p)$, if q moves during $\gamma_j \mapsto \gamma_{j+1}$, then $\neg\mathcal{B}ad_i^{j+1}(p,q)$.*

Proof. Let $i, j \in [0..t-1]$ such that $j \geq i$. By definition, if q moves, we have two possibilities:

- $\gamma_{j+1}(q).\mathcal{K} = SelfKey(q)$ so $q \notin \mathcal{F}_{j+1}(p)$ and consequently $\neg\mathcal{B}ad_i^{j+1}(p,q)$.
- $\exists q' \in \Gamma^-(q), \gamma_{j+1}(q).\mathcal{K} = (\gamma_j(q').M, \gamma_j(q').d + 1)$ and we have two subcases:
  1. $q' \notin \mathcal{F}_j(p)$. So $q \notin \mathcal{F}_{j+1}(p)$. Consequently $\neg\mathcal{B}ad_i^{j+1}(p,q)$.
  2. $q' \in \mathcal{F}_j(p)$. By definition $\gamma_j(q').d \geq \mathcal{M}df_j(p)$ and $\mathcal{M}df_j(p) \geq \mathcal{M}df_i(p)$, by Lemma 5. Thus, $\gamma_j(q').d + 1 > \mathcal{M}df_i(p)$, and so $\gamma_{j+1}(q).d > \mathcal{M}df_i(p)$. Consequently, $\neg\mathcal{B}ad_i^{j+1}(p,q)$.

Hence, in either cases, $\mathcal{B}ad_i^{j+1}(p,q)$ is false if q moves during $\gamma_j \mapsto \gamma_{j+1}$.

□

Below, we show that Properties $\neg\mathcal{B}ad_i^j(p,q)$ are closed under steps of Algorithm C-MAI$(\mathcal{D}, \prec, \mathcal{I})$.

Lemma 8 *Let p be any process. $\forall i, j \in [0..t-1]$ such that $j \geq i, \forall q \in \mathcal{A}nc(p)$, if $\neg\mathcal{B}ad_i^j(p,q)$, then $\neg\mathcal{B}ad_i^{j+1}(p,q)$.*

Proof. Let $i, j \in [0..t-1]$ such that $j \geq i$. Let p be a process and $q \in \mathcal{A}nc(p)$. Assume $\neg\mathcal{B}ad_i^j(p,q)$. We have two cases:

- q moves during the step $\gamma_j \mapsto \gamma_{j+1}$ and $\neg\mathcal{B}ad_i^{j+1}(p,q)$, by Lemma 7.
- q does not move during the step $\gamma_j \mapsto \gamma_{j+1}$ and we have two subcases:
  1. $q \notin \mathcal{F}_j(q)$. Then, $q \notin \mathcal{F}_{j+1}(q)$, and so $\neg\mathcal{B}ad_i^{j+1}(p,q)$.
  2. $q \in \mathcal{F}_j(q) \wedge \gamma_j(q).d > \mathcal{M}df_i(p)$. Then, $\gamma_{j+1}(q).d = \gamma_j(q).d > \mathcal{M}df_i(p)$, and consequently $\neg\mathcal{B}ad_i^{j+1}(p,q)$.

Hence, in either cases, if $\neg\mathcal{B}ad_i^j(p,q)$ holds, then $\neg\mathcal{B}ad_i^{j+1}(p,q)$ holds too.

□

Recall that e is finite in terms of steps (Lemma 4), so any round in e is finite. Let $R : \mathbb{N} \mapsto \mathbb{N}$ be a function such that $\forall i \in \mathbb{N}$:

- $R(i) = 0$ **if** $i = 0$;
- $R(i) = j$, where γ_j is the last configuration of the i^{th} round of e, **otherwise**.

Using the previous four technical lemmas, we now demonstrate that all fake values vanish from the network $\mathcal{N}$ within at most $\alpha + 1$ rounds.

Lemma 9 *From any initial configuration, in at most $\alpha + 1$ rounds there is no fake value in $\mathcal{N}$.*

Proof.
Let $p \in V(\mathcal{N})$. We first prove by induction on i that $\forall i \in \mathbb{N}, \mathcal{M}df_{R(i)}(p) \geq i$.

Base Case: $\gamma_{R(0)} = \gamma_0$. Now, by definition $\mathcal{M}df_{R(0)}(p) = \mathcal{M}df_0(p) \geq 0$.

Induction Step: If $\exists x \in [R(j)..R(j+1)], \mathcal{M}df_x(p) \geq j + 1$, then, by Lemma 5, $\mathcal{M}df_{R(j+1)}(p) \geq j + 1$.

Assume now, by contradiction, that $\forall x \in [R(j)..R(j+1)]$, $\mathcal{M}df_x(p) < j + 1$. Thus, $\mathcal{M}df_x(p) \leq j$. By induction hypothesis, $\mathcal{M}df_{R(j)}(p) \geq j$. So, by Lemma 5, $\forall x \in [R(j)..R(j+1)], \mathcal{M}df_x(p) \geq j$. Consequently, $\forall x \in [R(j)..R(j+1)], \mathcal{M}df_x(p) = j$.

Let $q \in \mathcal{A}nc(p)$ and consider the following two cases:

- $\exists x \in [R(j)..R(j+1)], \neg\mathcal{B}ad_j^x(p, q)$. By Lemma 8, $\neg\mathcal{B}ad_j^{R(j+1)}(p, q)$.
- $\forall x \in [R(j)..R(j+1)], \mathcal{B}ad_j^x(p, q)$. By Lemma 6, $q \in Enabled(\gamma_x)$ for any x in $[R(j)..R(j+1)]$, *i.e.*, q is continuously enabled during a complete round. By definition of a round, $\exists y \in [R(j)..R(j+1) - 1]$, such that q moves during the step $\gamma_y \mapsto \gamma_{y+1}$. By Lemma 7, $\neg\mathcal{B}ad_j^{y+1}(p, q)$, a contradiction. So, this case is impossible.

Hence, $\forall q \in \mathcal{A}nc(p), \neg\mathcal{B}ad_j^{R(j+1)}(p, q)$. Consequently, $\mathcal{M}df_j^{R(j+1)}(p) \neq j$, a contradiction. Thus, the induction holds.

By letting $i = \alpha+1$, we obtain that $\mathcal{M}df_{R(\alpha+1)}q(p) \geq \alpha+1$ from the previous induction. Now, since $\mathcal{M}df_i(p) \in [0..\alpha] \cup \{\infty\}$, $\mathcal{M}df_{R(\alpha+1)}(p) = \infty$ necessarily. Consequently, after $\alpha + 1$ rounds from initial configuration, there is no fake value at p's ancestors forever, by Lemma 5. As p is an arbitrary process, the theorem follows.

□

We now conclude the complexity analysis by showing that the system reaches a terminal configuration within $MAD + 1$ rounds from any configuration containing no fake value.

Beforehand, we should observe that each time a process p modifies its key, either p sets $p.M$ to $\mathcal{I}(p)$ or to $q.M$ with $q \in \Gamma^-(p)$. Hence, follows.

Remark 1 *The set of configurations containing no fake value is closed,* i.e.*, for every step $\gamma \mapsto \gamma'$ of Algorithm* C-MAI$(\mathcal{D}, \prec, \mathcal{I})$*, if γ contains no fake value, then so is γ'.*

Lemma 10 *The system reaches a terminal configuration within at most $MAD + 1$ rounds from any configuration containing no fake value.*

Proof. Assume the first configuration γ_0 of the execution $e = \gamma_0 \dots \gamma_i \dots$ contains no fake value. We first prove by induction on i that $\forall i \in \mathbb{N}^*, \forall p \in V(\mathcal{N})$, at the end of the i^{th} round of e:

- **If** $MinI(p) < i$, **then** $p.\mathcal{K} = (MinI(p), Mind(p))$ forever,
- $p.\mathcal{K} \trianglerighteq (MinI(p), i)$ forever, **otherwise**.

Base Case: $i = 1$.

- Let $p \in V(\mathcal{N})$ such that $Mind(p) = 0$. By definition, $MinI(p) = \mathcal{I}(p)$. Let $j \geq 0$. Since there is no fake value at all (by Remark 1), $BestKey_j(p) = SelfKey(p) = (\mathcal{I}(p), 0) = (MinI(p), Mind(p))$. Then, we have the following three claims for $\gamma_j(p).\mathcal{K}$:

  1. If $\gamma_j(p).\mathcal{K}$ is already equal to $SelfKey(p)$, then $\forall l \geq j, \gamma_l(p).\mathcal{K} = SelfKey(p)$.
  2. If $\gamma_j(p).\mathcal{K}$ is not equal to $SelfKey(p)$, then $\gamma_j(p).\mathcal{K} \neq BestKey_j(p)$. Hence, p is enabled in γ_j.
  3. By Algorithm C-MAI$(\mathcal{D}, \prec, \mathcal{I})$, if p moves during $\gamma_j \mapsto \gamma_{j+1}$, then $\gamma_{j+1}(p).\mathcal{K} \leftarrow BestKey_j(p) = SelfKey(p)$.

  By Claims 1-3 and the definition of round, $\forall j \geq R(1), \gamma_j(p).\mathcal{K} = SelfKey(p) = (MinI(p), Mind(p))$.

- Let $p \in V(\mathcal{N})$ such that $Mind(p) \geq 1$. Let $j \geq 0$. Since there is no fake value at all (by Remark 1), $\forall q \in \Gamma^-(p), \gamma_j(q).M \succeq MinI(p)$. So, $(\gamma_j(q).M, \gamma_j(q).d + 1) \trianglerighteq (MinI(p), 1)$. Moreover, $Mind(p) \geq 1$ implies that $\mathcal{I}(p) \succ MinI(p)$. Thus, $SelfKey(p) \rhd (MinI(p), 1)$. Hence, $BestKey_j(p) \trianglerighteq (MinI(p), 1)$.

  Then, we have two cases:

  1. p performs at least one action during the round, say during $\gamma_\ell \mapsto \gamma_{\ell+1}$. By definition of Algorithm C-MAI$(\mathcal{D}, \prec, \mathcal{I})$, $p.\mathcal{K} \leftarrow BestKey_\ell(p)$. So, from $\gamma_{\ell+1}$, $q.\mathcal{K} \trianglerighteq (MinI(p), 1)$ forever.

2. p does not perform any action during the round and we have two subcases :
   - $\exists \ell \in [0..R(1)]$ such that $\gamma_{\ell+1}(p).\mathcal{K} = BestKey_\ell(p)$ and we are done.
   - Otherwise, p is continuously enabled during the round without executing $ReplaceKey$, a contradiction: this case is impossible.

Induction Step:

- Let $p \in V(\mathcal{N})$ such that $Mind(p) < i+1$. If $Mind(p) < i$, then we are done by induction hypothesis. Assume now that $Mind(p) = i$. Let $j \geq R(i)$. We have the following three facts:

  1. $\exists q \in \Gamma^-(p)$ such that $MinI(q) = MinI(p)$ and $Mind(q) = Mind(p) - 1 < i$. So, by induction hypothesis, $\gamma_j(q).\mathcal{K} = (MinI(q), Mind(q)) = (MinI(p), Mind(p) - 1)$.
  2. Since $Mind(p) = i > 0$, $\mathcal{I}(p) \succ MinI(p)$, which implies $SelfKey(p) \rhd (MinI(p), Mind(p))$.
  3. $\forall q' \in \Gamma^-(p)$, q' satisfies one of the following two cases:
     - $MinI(q') = MinI(p)$. Then, by definition, $Mind(q') \geq Mind(p) - 1 = i - 1$. Then, we have two subcases:
       * $Mind(q') = i-1$. By induction hypothesis, $\gamma_j(q').\mathcal{K} = (MinI(q'), Mind(q')) = (MinI(p), i-1)$.
       * $Mind(q') \geq i$. By induction hypothesis, $\gamma_j(q').\mathcal{K} \trianglerighteq (MinI(q'), i) = (MinI(p), i)$.
     - $MinI(q') \neq MinI(p)$. Then, since $q' \in \Gamma^-(p)$, $MinI(q') \succ MinI(p)$ by definition. Since there is no fake value at all (by Remark 1), $\gamma_j(q').M \succeq MinI(q') \succ MinI(p)$. Hence, $\gamma_j(q').\mathcal{K} \rhd (MinI(p), i-1)$.

     Overall, $\gamma_j(q').\mathcal{K} \trianglerighteq (MinI(p), i-1) = (MinI(p), Mind(p) - 1)$.

  By Claims 1-3, $BestKey^j(p) = (MinI(p), Mind(p))$. Hence, as previously we can deduce that $p.\mathcal{K} = (MinI(p), Mind(p))$ forever at the end of the $i+1^{th}$ round.

- Let $p \in V(\mathcal{N})$ such that $Mind(p) \geq i+1$. We have the following two facts:

  1. Since $Mind(p) > 1$, $\mathcal{I}(p) \succ MinI(p)$. So, $SelfKey(p) \rhd (MinI(p), i+1)$.
  2. Let $j \geq R(i)$. Let $q \in \Gamma^-(p)$, we have the following two cases:

- $MinI(q) = MinI(p)$. Then, $Mind(q) \geq i$ by definition. By induction hypothesis, $\gamma_j(q).\mathcal{K} \trianglerighteq (MinI(q), i) = (MinI(p), i)$.
- $MinI(q) \neq MinI(p)$. By definition, $MinI(q) \succ MinI(p)$. Thus, since there is no fake value at all (by Remark 1), $\gamma_j(q).\mathcal{K} \triangleright (MinI(p), i)$.

Overall, the two previous facts imply that $BestKey_j(p) \trianglerighteq (MinI(p), i+1)$.

Hence, as previously, we can deduce that $p.\mathcal{K} \trianglerighteq (MinI(p), i+1)$ forever at the end of the $i+1^{th}$ round and the induction holds.

By letting $i = MAD + 1$, the lemma follows from the previous induction since $MinI(p) < MAD + 1, \forall p \in V(\mathcal{N})$.

□

4 Leader Election

In this section, we propose a necessary and sufficient condition for solving the self-stabilizing leader election in any directed identified network where all processes know a common upper bound α on MAD. Recall that the sufficient part consists in a silent self-stabilizing algorithm built as a composition of three different instances of Algorithm C-MAI$(\mathcal{D}, \prec, \mathcal{I})$.

4.1 The Problem

In an identified network $\mathcal{N}$, the *public leader election problem*, leader election for short, consists in making processes agree on a unique process identifier. Thus, this problem can be specified using a (local) function $\mathcal{L} : V(\mathcal{N}) \rightarrow IDSET$ such that $\mathcal{L}(p)$ returns the identifier of the leader appointed by process p in the current configuration. An execution $e = \gamma_0 \ldots \gamma_i \ldots$ satisfies the specification of the leader election, denoted by $\mathcal{SP}_{\mathcal{L}}$, if:

1. in γ_0, $\forall p, q \in V(\mathcal{N}), \mathcal{L}(p) = \mathcal{L}(q)$, *i.e.*, all processes agree on the same identifier;
2. in γ_0, $\exists q \in V(\mathcal{N})$ such that $\mathcal{L}(q) = q.\mathcal{I}d$, *i.e.*, the chosen identifier is held by some process; and
3. $\forall i \geq 0, \forall p \in V(\mathcal{N})$, if γ_i is not terminal, then the value of $\mathcal{L}(p)$ is the same in γ_i and γ_{i+1}, *i.e.*, the designated leader is final.

Notice that in the case of a silent self-stabilizing leader election algorithm, the partial correctness property has only to check that in a terminal configuration all processes agree on the same identifier and the chosen identifier is held by some process (more formally, $\exists \ell \in V(\mathcal{N})$ such that $\forall p \in V(\mathcal{N}), \mathcal{L}(p) = \ell.\mathcal{I}d$). Indeed, the fact that the designated leader is final is trivial in a terminal configuration.

4.2 Necessary Condition

We now propose a topology-based necessary condition for solving the self-stabilizing leader election. This condition claims that the network $\mathcal{N}$ should have exactly one source component; see Section 2 for the definition. Intuitively, processes in a source component cannot learn any identifier of processes outside the component. Consequently, they necessarily elect a process inside their component (Lemma 13). Hence, with at least two source components, we cannot have the uniqueness of the leader (Corollary 2).

Let X be a subset of processes. In the following, we denote by $\gamma_{|X}$ the projection of the configuration γ onto X. Let e be a sequence of configurations. We define the *infinite extension* of e, $Ext^{\infty}(e)$, as follows:

- $Ext^{\infty}(e) = e$ **if** e is infinite;
- $Ext^{\infty}(e) = e\gamma^{\omega}$, where γ is the last configuration of e, **otherwise**.

The infinite extension consists then in extending any finite execution with an infinite suffix only made of its terminal configuration.

Lemma 11 *Let $e = \gamma_0 \dots \gamma_i \dots$ be a sequence of configurations. $\mathcal{SP}_{\mathcal{L}}(e) \equiv \mathcal{SP}_{\mathcal{L}}(Ext^{\infty}(e))$.*

Proof. Since e and $Ext^{\infty}(e)$ begin with the same configuration, e satisfies the first two points of the definition of $\mathcal{SP}_{\mathcal{L}}$ if and only if $Ext^{\infty}(e)$ does. Moreover, since e is a prefix of $Ext^{\infty}(e)$ and contains all non-terminal configurations of $Ext^{\infty}(e)$, e satisfies the third point if and only if $Ext^{\infty}(e)$ does. □

The lemma below is inspired from Lemma 2.2 in [ACDD23].

Lemma 12 *Let $\mathtt{A}$ be a deterministic distributed algorithm. Let SC be any source component of $\mathcal{N}$. Let $\gamma_0 \dots \gamma_i \dots$ be the infinite extension of a synchronous execution of $\mathtt{A}$ in $\mathcal{N}$. Let $\pi_0 \dots \pi_i \dots$ be the infinite extension of a synchronous execution of $\mathtt{A}$ in $\mathcal{N}$ that starts from any configuration π_0 satisfying $\pi_{0|V(SC)} = \gamma_{0|V(SC)}$. We have $\pi_{i|V(SC)} = \gamma_{i|V(SC)}$, $\forall i \in \mathbb{N}$.*

Proof. By induction on i. The base case $i = 0$ is trivial by definition of π_0. Let $i \in \mathbb{N}$. Assume that $\pi_{i|V(SC)} = \gamma_{i|V(SC)}$. Let $p \in V(SC)$. We have $\pi_i(p) = \gamma_i(p)$. Moreover, $\forall q \in \Gamma^-(p), q \in V(SC)$, by definition of source component. So, $\pi_i(q) = \gamma_i(q)$. Thus, $p \in Enabled(\pi_i) \Leftrightarrow p \in Enabled(\gamma_i)$. Hence, $\pi_{i+1}(p) = \gamma_{i+1}(p)$ since A is deterministic and the two considered executions are synchronous. We can then conclude that $\pi_{i+1|V(SC)} = \gamma_{i+1|V(SC)}$ and we are done. □

Lemma 13 *Let* A *be a self-stabilizing algorithm for the leader election in $\mathcal{N}$ under the synchronous daemon. Let γ be a legitimate configuration of* A *in $\mathcal{N}$. Let SC be any source component of $\mathcal{N}$. The leader ℓ in γ necessarily belongs to SC.*

Proof. Assume, by contradiction, that $\ell \notin V(SC)$. Let $id = \gamma(\ell).\mathcal{I}d$ be the value of the ℓ's identifier in γ. Let $\gamma_0(= \gamma) \dots \gamma_i \dots$ be the infinite extension of a synchronous execution e of A in $\mathcal{N}$ that starts from γ. Since γ is legitimate, e verifies $\mathcal{SP}_{\mathcal{L}}$. From $\mathcal{SP}_{\mathcal{L}}$ and by definition of infinite extension, we have $\forall p \in V(\mathcal{N}), \forall i \in \mathbb{N}, \mathcal{L}(p) = id$ in γ_i. Let id' be any value in $IDSET \setminus \{\gamma(p).\mathcal{I}d \mid p \in V(\mathcal{N})\}$ (*n.b.*, id' is well-defined since $|IDSET| > |V(\mathcal{N})|$, by hypothesis). Let π be a configuration where $\ell.id = id'$ and $\pi_{|SC} = \gamma_{|SC}$. Notice that π is well-defined since $\ell \notin V(SC)$. Moreover, the uniqueness of identifiers is preserved since, by definition of id', $\forall p \in V(\mathcal{N}) \setminus \{\ell\}, p.\mathcal{I}d \neq id'$ in π. Let IDS_{SC} be the set of identifiers of processes in $V(SC)$ in π. Consider now, the infinite extension $\pi_0(= \pi) \dots \pi_i \dots$ of the synchronous execution e' of A in $\mathcal{N}$ that starts from π. By Lemma 12, we have $\forall i \in \mathbb{N}, \pi_{i|V(SC)} = \gamma_{i|V(SC)}$. So, $\forall p \in V(SC), \forall i \in \mathbb{N}$, $\mathcal{L}(p) = id \notin IDS_{SC}$ in π_i. Since $|V(\mathcal{N})| \geq 1$ (by hypothesis), $V(SC) \neq \emptyset$ and so no suffix of $\pi_0 \dots \pi_i \dots$ satisfies $\mathcal{SP}_{\mathcal{L}}$. By Lemma 11, e' is a synchronous execution of A in $\mathcal{N}$ where no suffix satisfies $\mathcal{SP}_{\mathcal{L}}$. Since A should be insensitive to its inputs, A is then not self-stabilizing for the leader election in $\mathcal{N}$ under the synchronous daemon, a contradiction. □

By definition, each process belongs to exactly one strongly connected component, so we have the following corollary:

Corollary 2 *Assuming unique process identifiers and the common knowledge of an upper bound α on MAD, an algorithm is self-stabilizing for leader election in $\mathcal{N}$ under the synchronous daemon only if $\mathcal{N}$ contains exactly one source component.*

4.3 The Algorithm

Our leader election algorithm is denoted by PEL. According to Corollary 2, it assumes the network $\mathcal{N}$ has exactly one source component. Algorithm PEL is a hierarchical collateral

composition of the three instances of Algorithm C-MAI presented below. In the sequel, for the sake of clarity, we denote by $p.\mathtt{X}.v$ the variable named v at process p in Instance X. We also denote by mid the smallest identifier of a process in the unique source component SC, *i.e.*, $mid = \min\{q.\mathcal{I}d \mid q \in V(SC)\}$ (mid is well-defined since $n \geq 1$ implies that $V(SC) \neq \emptyset$). The leader identifier computed by PEL will be mid.

1. The first instance $\mathtt{MI} = \text{C-MAI}(IDSET, <_I, \mathcal{I}_{\mathtt{MI}})$ computes at each process the minimum identifier of its ancestors, where $\mathcal{I}_{\mathtt{MI}} : V(\mathcal{N}) \to IDSET$ returns $\mathcal{I}_{\mathtt{MI}}(p) = p.\mathcal{I}d$ for each process p. An example of output identifiers (*i.e.*, the M-variables of MI) computed by MI is given in Configuration (a) of Figure 5.

   By definition, the output identifier of every process in SC will be mid (Corollary 4).

2. The second instance $\mathtt{A} = \text{C-MAI}(\mathbb{B} = \{0, 1\}, <, \mathcal{I}_{\mathtt{A}})$ evaluates at each process whether the process computed the same output identifier in MI as its ancestors.

   This instance uses the input function $\mathcal{I}_{\mathtt{A}} : V(\mathcal{N}) \to \mathbb{B}$ that evaluates whether the process computed the same output identifier in the first instance as its predecessors, *i.e.*, $\mathcal{I}_{\mathtt{A}}(p) = (\forall q \in \Gamma^-(p) \mid q.\mathtt{MI}.M = p.\mathtt{MI}.M)$.

   In this instance, the boolean output (*i.e.*, the M-variable) of a process will be 1 if and only if the identifier computed by the process in MI is mid (Lemma 14). Indeed, since the source component SC is unique, each process has at least one ancestor (maybe itself) in SC. So, if the output identifier of some process p in MI is not mid, then there is a path from a process of SC to p containing at least one process q whose output identifier in MI stabilizes to an identifier different from that computed by one of its predecessors. So, eventually $\mathcal{I}_{\mathtt{A}}(q) = 0$ forever and, consequently, the boolean output computed by p in A will be 0. Otherwise, all p's ancestors (including p) eventually agree on mid and the boolean output computed by p in A will be 1. An example of boolean output computed by A (*i.e.*, the M-variables of A) is given in Configuration (b) of Figure 5.

3. The third and last instance $\mathtt{E} = \text{C-MAI}(\mathbb{B} \times IDSET, <_C, \mathcal{I}_{\mathtt{E}})$ computes the leader identifier.

   The input of this instance is a couple in $\mathbb{B} \times IDSET$ made of the negation of the boolean output of A and the output identifier of MI. Thus, $\mathcal{I}_{\mathtt{E}} : V(\mathcal{N}) \to \mathbb{B} \times IDSET$ returns $(\overline{p.\mathtt{A}.M}, p.\mathtt{MI}.M)$ for each process p. Moreover, $<_C$ is the lexicographic order on $\mathbb{B} \times IDSET$, *i.e.*, $\forall (a, b), (c, d) \in \mathbb{B} \times IDSET, (a, b) <_C (c, d) \equiv [a < c \vee (a = c \wedge b <_I d)]$.

   By flipping the bit $p.\mathtt{A}.M$, we have the guarantee that each process p eventually forever satisfies $\mathcal{I}_{\mathtt{E}}(p) = (0, id)$ if and only if $id = mid$ (Lemma 15). In particular,

every process q in the source component SC will eventually forever satisfy $\mathcal{I}_{\texttt{E}}(q) = (0, mid)$, which is minimum. Now, every process has at least one ancestor in SC, by definition. Consequently, the output couple (*i.e.*, the M-variable of E) of the instance will be identical for all processes: $(0, mid)$; see Configuration (c) in Figure 5 for an illustrative example.

For each process p, we denote by $p.\texttt{E}.M.L$ (resp. $p.\texttt{E}.M.R$) the left (resp. right) member of the output couple $p.\texttt{E}.M$. Hence, in the terminal configuration, for every process p, the identifier of the elected leader will be $p.\texttt{E}.M.R$, *i.e.*, $\mathcal{L}(p) = p.\texttt{E}.M.R$.

Thus, we let $\texttt{PEL} = \texttt{E} \circ \texttt{A} \circ \texttt{MI}$.

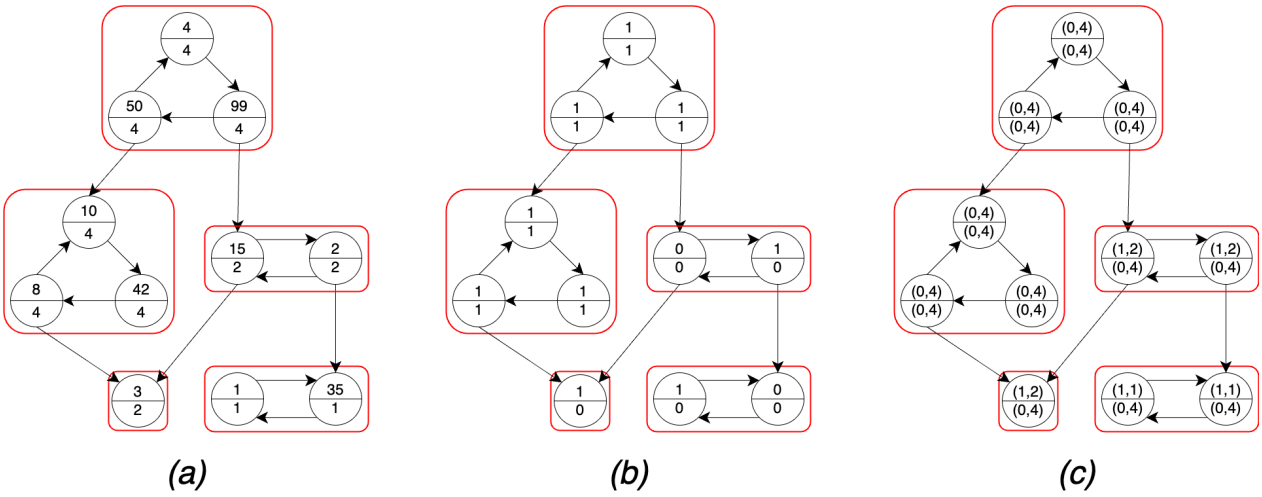

Figure 5: Configurations (a), (b), and (c) respectively show values of the inputs and M-outputs of MI, A, and E in a terminal configuration of PEL. Red rounded rectangles delimit strongly connected components. For each configuration, the input and M-output of each process are given inside the corresponding circle, respectively at the top and the bottom.

4.4 Proof of Silent Self-stabilization and Complexity Analysis

We now establish Theorem 3 below. To prove this theorem, we should first show the silent self-stabilization of Algorithm PEL. This property can be split into termination and partial correctness. Termination is immediate from Lemma 1 (page 11) and Theorem 1 (page 17). The partial correctness will be stated by Lemma 15. We should then conclude with the stabilization time in rounds of Algorithm PEL. This latter is direct from Corollary 1 (page 11) and Theorem 2 (page 21).

Theorem 3 *Assuming unique process identifiers and the common knowledge of an upper bound α on MAD, in any directed network $\mathcal{N}$ with a unique source component, Algorithm* PEL *is silent self-stabilizing under the distributed unfair daemon for the leader election. Its stabilization time is at most $3\alpha + 3MAD + 6$ rounds.*

From the above theorem and Corollary 2, we immediately obtain the following necessary and sufficient condition for solving the leader election in our settings.

Corollary 3 *Assuming unique process identifiers and the common knowledge of an upper bound α on MAD, there exists a silent self-stabilizing algorithm under the distributed unfair daemon (resp., the synchronous daemon) for the leader election in a directed network $\mathcal{N}$ if and only if $\mathcal{N}$ contains exactly one source component.*

Partial Correctness. From now on, we consider any network $\mathcal{N}$ with a unique source component. Recall that $mid = \min\{q.\mathcal{I}d \mid q \in V(SC)\}$. Let γ_t be any terminal configuration of PEL. By definition, we have:

Remark 2 *$\gamma_{t|\mathtt{MI}}$, $\gamma_{t|\mathtt{A}}$, and $\gamma_{t|\mathtt{E}}$ are terminal configurations of* MI*,* A*, and* E*, respectively.*

By definition of MI, Theorem 1 (page 17), and Remark 2, we have:

Corollary 4 *$\forall p \in V(SC), \gamma_t(p).\mathtt{MI}.M = mid$.*

Lemma 14 *$\forall p \in V(\mathcal{N}), \gamma_t(p).\mathtt{A}.M = 1$ if and only if $\gamma_t(p).\mathtt{MI}.M = mid$.*

Proof. Let $p \in V(\mathcal{N})$.

If Part. By Remark 2, Definition of A, and Theorem 1 (page 17), in γ_t, if $p.\mathtt{A}.M = 1$, then $\forall q \in \mathcal{A}nc(p), \mathcal{I}_{\mathtt{A}}(q) = 1$, which implies that $\forall x \in \Gamma^-(q)$, $x.\mathtt{MI}.M = q.\mathtt{MI}.M$. By transitivity, we have $\forall q \in \mathcal{A}nc(p), \gamma_t(q).\mathtt{MI}.M = \gamma_t(p).\mathtt{MI}.M$. By definition, $V(SC) \cap \mathcal{A}nc(p) \neq \emptyset$. Consequently, by Corollary 4, $\exists y \in \mathcal{A}nc(p), \gamma_t(y).\mathtt{MI}.M = mid$. Hence, $\gamma_t(p).\mathtt{MI}.M = mid$.

Only If Part. Let $q \in \mathcal{A}nc(p)$. By Theorem 1, Remark 2, and definition of A, $\gamma_t(q).\mathtt{MI}.M \geq_I \gamma_t(p).\mathtt{MI}.M = mid$. By definition, $V(SC) \subseteq \mathcal{A}nc(q)$ and $V(SC) \neq \emptyset$. Consequently, by definition of mid, Remark 2, and Theorem 1, $\gamma_t(q).\mathtt{MI}.M \leq_I mid$. So, $\gamma_t(q).\mathtt{MI}.M = mid$. Thus, $\forall x \in \mathcal{A}nc(p), \gamma_t(x).\mathtt{MI}.M = mid$, which implies that $\forall x \in \mathcal{A}nc(p), \forall y \in \Gamma^-(x), \gamma_t(y).\mathtt{MI}.M = \gamma_t(x).\mathtt{MI}.M$, *i.e.*, $\mathcal{I}_{\mathtt{A}}(x) = 1$ in γ_t. Hence, by definition of A, Remark 2, and Theorem 1, $\gamma_t(p).\mathtt{A}.M = 1$.

□

Lemma 15 $\forall p \in V(\mathcal{N}), \gamma_t(p).\mathtt{E}.M = (0, mid)$, i.e., $\mathcal{L}(p) = mid$ *in* γ_t.

Proof. Let $p \in V(\mathcal{N})$. By definition, $V(SC) \cap \mathcal{A}nc(p) \neq \emptyset$. Let $q \in V(SC) \cap \mathcal{A}nc(p)$. By Corollary 4, Lemma 14, and definition of $\mathtt{E}$, in γ_t, we have $\mathcal{I}_{\mathtt{E}}(q) = (\overline{q.\mathtt{A}.M}, q.\mathtt{MI}.M) = (0, mid)$. Moreover, $\forall x \in \mathcal{A}nc(p)$, in γ_t, $\mathcal{I}_{\mathtt{E}}(x) = (\overline{x.\mathtt{A}.M}, x.\mathtt{MI}.M) \geq_C (0, mid)$, by Lemma 14 and definition of $\mathtt{E}$. Hence, by Remark 2, Theorem 1, and definition of $\mathtt{E}$, $\gamma_t(p).\mathtt{E}.M = (0, mid)$. □

5 Synchronous Unison

In this section, we propose a necessary and sufficient condition for solving the self-stabilizing synchronous unison in any directed identified network where all processes know a common upper bound α on MAD. Namely, the communication network should contain exactly one source component. Notice that the definition of this problem itself requires to assume a *synchronous* daemon. For the sufficient part of the condition, we propose an algorithm which uses our leader election algorithm, PEL, as basic building block.

5.1 The Problem

In [ADG91], Arora *et al.* define the *synchronous unison* as follows. Each process is endowed with a bounded integer variable, called its *clock*, whose domain is $[0..K-1]$, where K is a positive integer greater than 1 and called its *period*. An execution e satisfies the *synchronous unison specification* $\mathcal{SP}_{\mathcal{SU}}$ if the following two properties hold:

- In every configuration of e, all clocks have the same value (*safety*).
- All clocks are incremented modulo K at each round (*liveness*).

5.2 Necessary Condition

The necessary part of the condition has been already proven in [ACDD23]. Actually, in [ACDD23] networks are assumed to be anonymous, but in the paper it is also highlighted that the proposed proof still holds in identified networks since based on the fact that processes in two distinct source components cannot, even indirectly, communicate and so coordinate to agree on a common clock value. Of course, this claim remains true when processes also agree on a common value α. Hence, follows:

Theorem 4 *Assuming unique process identifiers and the common knowledge of an upper bound α on MAD, an algorithm is self-stabilizing for the synchronous unison in $\mathcal{N}$ under the synchronous daemon only if $\mathcal{N}$ contains exactly one source component.*

Below, we focus on the sufficient part. Precisely, we present a self-stabilizing synchronous unison algorithm built as a composition of PEL and a unison algorithm for dags (directed acyclic graphs) with a unique source.

5.3 The Algorithm

In PEL, processes do not only compute the leader identifier, but also their distance from the elected leader ℓ. As a by-product, those distances exhibit a dag whose single source is ℓ. More precisely, in a terminal configuration of PEL, an arc (q, p) of the network belongs to the dag if and only if $q \in \Gamma^-(p) \wedge q.\mathtt{E}.d < p.\mathtt{E}.d$ (*i.e.*, $\|\ell, q\| < \|\ell, p\|$); see Figure 6 for an illustrative example. So, to obtain a synchronous unison, we make clocks of all processes converge to the clock value of ℓ by broadcasting it along the dag using Algorithm U-DAG(K) whose code is given in Algorithm 2. Overall, our self-stabilizing synchronous unison algorithm U(K) is then defined as U(K) = U-DAG$(K) \circ$ PEL where K is any integer satisfying $K > 1$. In more detail, once PEL has terminated, the leader ℓ (the only process satisfying $\mathtt{E}.d = 0$) increments its clock modulo K at each synchronous step; see Macro $NextValue(p)$. Furthermore, at each synchronous step, each other process updates its clock to $(c + 1) \bmod K$, where c is the minimal value among the clocks of its predecessors in the dag; see Macros $Values(p)$[4] and $NextValue(p)$. A sample execution of U(K) starting from a terminal configuration of PEL is given in Figure 7.

5.4 Proof of Self-stabilization and Complexity Analysis

In this subsection, we assume a synchronous daemon and demonstrate the self-stabilization of U(K) for the synchronous unison problem in any directed network $\mathcal{N}$ containing a unique source component (Theorem 5). To obtain this result, we prove the partial correctness and convergence properties in Lemma 16 and Corollary 6, respectively. In Corollary 6, we also establish that the stabilization time of U(K) is at most $3\alpha + 4MAD + 6$. Theorem 5 below is immediate from those intermediate results.

Theorem 5 *Assuming unique process identifiers, the common knowledge of an upper bound α on MAD, and a synchronous daemon, Algorithm* U(K) *(with $K > 1$) is a self-stabilizing synchronous unison algorithm in any directed network $\mathcal{N}$ with a unique source component. Its stabilization time is at most $3\alpha + 4MAD + 6$ synchronous rounds.*

[4]We artificially add Value $K - 1$ to $Values(p)$ to ensure the set is never empty.

Algorithm 2: Algorithm U-DAG(K), code for any process p.

Inputs:

$\Gamma^{-}(p)$: the set of p's predecessors

$p.\mathtt{E}.d$: the distance computed by E

Variable:

$p.Clock \in [0..K-1]$: the clock of p

Macros:

$Values(p) = \{q.Clock \mid q \in \Gamma^{-}(p) \ \wedge q.\mathtt{E}.d < p.\mathtt{E}.d\} \cup \{K-1\}$

$NextValue(p) =$ **if** $p.\mathtt{E}.d = 0$ **then** $(p.Clock + 1) \bmod K$ **else** $(\min(Values(p)) + 1) \bmod K$

Action:

$Incr :: p.Clock \neq NextValue(p) \rightarrow p.Clock \leftarrow NextValue(p)$

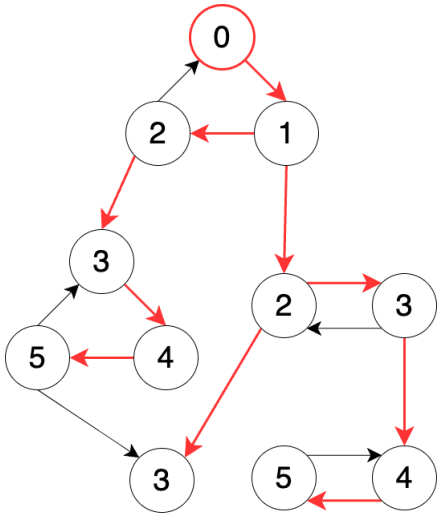

Figure 6: Distance values computed by E in the example of terminal configuration given in Figure 5. Inside each circle, we give the value of Variable E.d. The bold red circle indicates the chosen leader. The implicit DAG computed by PEL is exhibited with the bold red arcs.

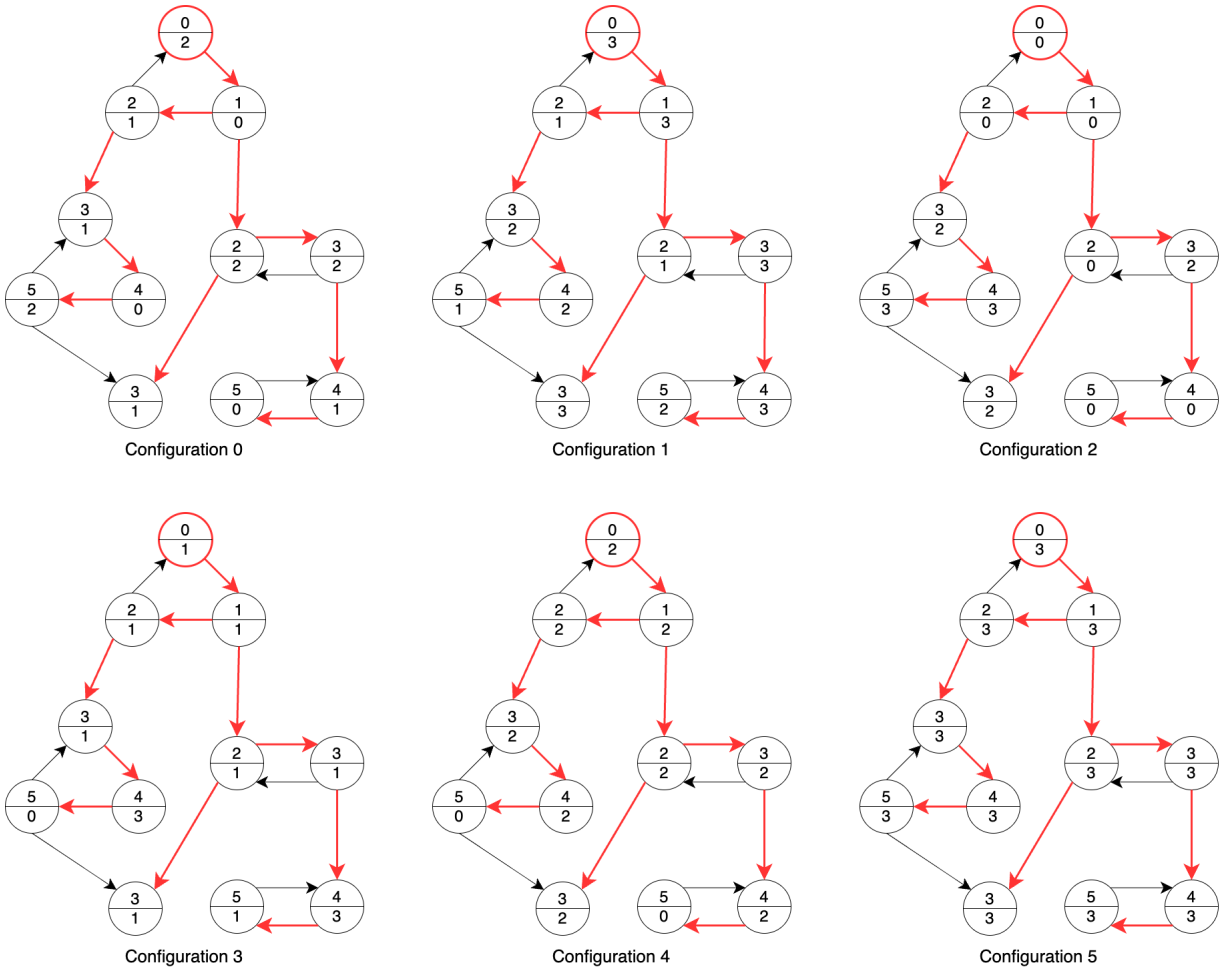

Figure 7: Sample of execution of U-DAG(4). For each configuration, the distance and clock value of each process are given inside the corresponding circle, respectively at the top and the bottom. The bold red circle indicates the leader. The implicit DAG computed by PEL is exhibited by the bold red arcs.

From Theorems 4 and 5, we immediately obtain the following necessary and sufficient condition for solving the synchronous unison in our settings.

Corollary 5 *Assuming unique process identifiers and the common knowledge of an upper bound α on MAD, there exists a self-stabilizing algorithm under the synchronous daemon for the synchronous unison problem in a directed network $\mathcal{N}$ if and only if $\mathcal{N}$ contains exactly one source component.*

To prove Theorem 5, we first need to define the legitimate configurations of $\mathtt{U}(K)$.

Definition 4 *A configuration γ of $\mathtt{U}(K)$ in the network $\mathcal{N}$ is* legitimate *if:*

- *γ is terminal for* PEL *and*
- *$\exists c \in [0..K-1]$ such that $\forall p \in V(\mathcal{N}), \gamma(p).Clock = c$.*

The lemma below establishes the partial correctness property.

Lemma 16 *Every synchronous execution of $\mathtt{U}(K)$ in the network $\mathcal{N}$ starting from a legitimate configuration satisfies $\mathcal{SP}_{\mathcal{SU}}$.*

Proof. Let γ be any legitimate configuration. By definition, no action of PEL is enabled in γ. So, from the definition of the composition, follows:

Claim 1: *A process p is enabled for $\mathtt{U}(K)$ in γ if and only if it is enabled for $\mathtt{U\text{-}DAG}(K)$ in γ, i.e., if and only if $p.Clock \neq NextValue(p)$ in γ.*

Moreover, the following claim is direct from the definition of legitimate configurations:

Claim 2: *$\exists c \in [0..K-1]$ such that $\forall p \in V(\mathcal{N})$, $\gamma(p).Clock = c$.*

Finally, we have

Claim 3: *In γ, every process p satisfies $NextValue(p) = (p.Clock + 1) \bmod K \neq p.Clock$.*

Proof of the Claim: Let p be any process. If $\gamma(p).\mathtt{E}.d = 0$, then in γ, $NextValue(p) = (p.Clock + 1) \bmod K \neq p.Clock$ since $K > 1$.

Otherwise, by Theorem 3, there exists a unique process ℓ such that $\forall v \in V(\mathcal{N}), \mathcal{L}(v) = \ell.\mathcal{I}d$ in γ, *i.e.*, $\gamma(v).\mathtt{E}.M = (0, \ell.\mathcal{I}d)$. Moreover, $\forall v \in V(\mathcal{N}), \gamma(v).\mathtt{E}.d = \|\ell, v\|$ by Theorem 1. Since $\gamma(p).\mathtt{E}.d > 0$, there exists $q \in \Gamma^-(p)$ such that $\gamma(q).\mathtt{E}.d = \|\ell, q\| = \|\ell, p\| - 1$. Hence, in γ, we have $\{q.Clock \mid q \in \Gamma^-(p) \ \wedge q.\mathtt{E}.d < p.\mathtt{E}.d\} \neq \emptyset$. By Claim 2 and owing the fact that $K > 1$, we can conclude that $NextValue(p) = (p.Clock + 1) \bmod K \neq p.Clock$.

By Claims 1 and 3, all processes are enabled and increment their clock to the same value in the following synchronous step. Thus, the configuration γ' reached from γ is still legitimate. Indeed, γ' is terminal for PEL, since U-DAG(K) does not write into PEL's variables, and all processes have the same clock value in γ'. Hence, inductively all requirements $\mathcal{SP}_{\mathcal{SU}}$ hold, and we are done. □

We now prove the convergence of U(K) (Corollary 6). To show this property, we propose the ad hoc proof below, indeed we cannot use Lemma 1 or Corollary 1 since U-DAG(K) is not silent.

Lemma 17 *No configuration of* U(K) *in* $\mathcal{N}$ *is terminal.*

Proof. Let γ be a configuration of U(K) in $\mathcal{N}$, we have two cases:

Case 1: $\gamma_{|\texttt{PEL}}$ *is not terminal for* PEL.

Since any action of PEL exists, and is not modified, in U(K), γ is not terminal for U(K).

Case 2: $\gamma_{|\texttt{PEL}}$ *is terminal for* PEL.

By definition, $\gamma_{|\texttt{E}}$ is terminal for E. By Theorem 1, $\exists p \in V(\mathcal{N}), \gamma(p).\texttt{E}.d = 0$. In this case, we have $p.Clock \neq NextValue(p)$ (since $K > 1$) and consequently p is enabled in γ: γ is not terminal for U(K).

□

From the previous lemma, we know that every execution of U(K) in $\mathcal{N}$ is infinite. So, let us now consider an arbitrary synchronous infinite execution $e = \gamma_0 \dots \gamma_i \dots$ of U(K) in $\mathcal{N}$.

Lemma 18 $\exists i \leq 3\alpha + 3MAD + 6$ *such that* $\gamma_{i|\texttt{PEL}}$ *is terminal for* PEL.

Proof. Let γ_j be an arbitrary configuration of e. Assume that $\gamma_{j|\texttt{PEL}}$ is not terminal for PEL. We have the following three facts:

1. Every action of PEL exists, and is not modified, in U(K).
2. If a process is enabled for PEL, it will execute PEL in the next step by definition of the composition and owing the fact that the daemon is synchronous.
3. U-DAG(K) does not write into variables of PEL.

From those three facts, we can deduce that $\gamma_{j|\texttt{PEL}} \mapsto \gamma_{j+1|\texttt{PEL}}$ is a synchronous step of PEL and by Theorem 3, we are done. □

Definition 5 *$\forall \eta \in \mathbb{N}$, we let $PseudoLeg(\eta)$ be the set of configurations γ such that:*

1. *$\gamma_{|\texttt{PEL}}$ is terminal for* PEL *and*
2. *$\exists c \in [0..K-1]$ such that $\forall p \in V(\mathcal{N}), \|\ell, p\| \leq \eta \Rightarrow \gamma(p).Clock = c$, where ℓ is the leader in γ (ℓ exists by Theorem 1).*

Remark 3 *Every configuration γ such that $\gamma_{|\texttt{PEL}}$ is terminal for* PEL *belongs to $PseudoLeg(0)$. $\forall i \in \mathbb{N}^*$, $PseudoLeg(i) \subseteq PseudoLeg(i-1)$. Since the elected leader ℓ necessarily belongs to the (unique) source component (*cf.*, Lemma 12), for every process p, $\|\ell, p\| \leq MAD$ and so $\forall i \geq MAD, \forall \gamma \in \mathcal{C}$, $\gamma \in PseudoLeg(i) \Leftrightarrow \gamma$ is legitimate.*

Lemma 19 *Let γ_i be a configuration of e such that $\gamma_{i|\texttt{PEL}}$ is terminal for* PEL. *$\gamma_{i+MAD|\texttt{PEL}}$ is legitimate.*

Proof. Let γ_i be a configuration of e such that $\gamma_{i|\texttt{PEL}}$ is terminal for PEL. Since, U-DAG(K) does not write into PEL's variables, $\forall j \geq i, \gamma_{j|\texttt{PEL}}$ is terminal for PEL. By Theorem 3, there exists a unique process ℓ such that $\forall j \geq i, \forall p \in V(\mathcal{N}), \mathcal{L}(p) = \ell.\mathcal{I}d$ in γ_j, *i.e.*, $\gamma_j(p).\texttt{E}.M = (0, \ell.\mathcal{I}d)$. Moreover, $\forall j \geq i, \forall p \in V(\mathcal{N}), \gamma_j(p).\texttt{E}.d = \|\ell, p\|$ by Theorem 1. We now show by induction on j that $\forall j \in \mathbb{N}, \gamma_{i+j} \in PseudoLeg(j)$.

Base case: $\gamma_i \in PseudoLeg(0)$ by Remark 3.

Induction step: Assume that $\gamma_{i+j} \in PseudoLeg(j)$ from some $j \in \mathbb{N}$. By definition of $PseudoLeg(j)$, $\exists c \in [0..K-1]$ such that for every process p satisfying $\|\ell, p\| \leq j$, we have $\gamma_{i+j}(p).Clock = c$. Let p be a process such that $\|\ell, p\| \leq j+1$. We have two cases:

$\|\ell, p\| = 0$: Then, $p = \ell$ and $\gamma_{i+j}(p).\texttt{E}.d = \|\ell, p\| = 0$. So, in γ_{i+j}, $NextValue(p) = (\ell.clock + 1) \bmod K = (c+1) \bmod K$, by induction hypothesis. Consequently, $\gamma_{i+j+1}(p).Clock = (c+1) \bmod K$.

$\|\ell, p\| > 0$: By definition, $\exists q \in \Gamma^-(p)$ such that $\|\ell, q\| = \|\ell, p\| - 1 = j$. Now, $\gamma_{i+j}(q).\texttt{E}.d = \|\ell, q\| < \|\ell, p\| = \gamma_{i+j}(p).\texttt{E}.d$. Hence, in γ_{i+j}, $\{q.Clock \mid q \in \Gamma^-(p) \ \wedge q.\texttt{E}.d < p.\texttt{E}.d\} \neq \emptyset$. Moreover, $\forall q \in \Gamma^-(p)$ such that $\gamma_{i+j}(q).\texttt{E}.d < \gamma_{i+j}(p).\texttt{E}.d$, we have $\gamma_{i+j}(q).\texttt{E}.d = \|\ell, q\| < \gamma_{i+j}(p).\texttt{E}.d = \|\ell, p\| = j+1$, and so $\gamma_{i+j}(q).Clock = c$ by induction hypothesis. Thus, $NextValue(p) = (c+1) \bmod K$ in γ_{i+j} and, consequently, $\gamma_{i+j+1}(p).Clock = (c+1) \bmod K$.

Hence, $\forall p \in V(\mathcal{N})$ such that $\|\ell, p\| \leq j+1$, we have $\gamma_{i+j+1}(p).Clock = (c+1) \bmod K$, which implies that $\gamma_{i+j+1} \in PseudoLeg(j+1)$, and the induction holds.

By letting $j = MAD$, we have $\gamma_{i+MAD} \in PseudoLeg(MAD)$ and the lemma holds, by Remark 3. □

From Lemmas 18 and 19, we immediately obtain the corollary below. This latter establishes both the convergence property and the stabilization time of $\mathtt{U}(K)$.

Corollary 6 $\exists i \leq 3\alpha + 4MAD + 6$ *such that* γ_i *is legitimate.*

6 Conclusion

In this paper, we have proposed a tight topological condition, namely the fact that the network should contain exactly one source component, allowing the design of self-stabilizing solutions in directed identified networks where processes know an upper bound α on the maximum ancestor distance. Our results show, maybe surprisingly, that some fundamental static and dynamic problems can be self-stabilizingly solved in networks of very general topologies which are not necessarily strongly connected.

The immediate perspective of this work deals with time complexity. Since our leader election algorithm works under the distributed unfair daemon, its stabilization time in moves (*i.e.*, the number of state updates) is finite. So, it is worth analyzing this time complexity.

Another interesting perspective concerns α. We use this knowledge for two main reasons: (1) we need global knowledge on the network to bound process memories, and (2) our mechanism to remove fake values also requires some global knowledge on the network. A side-effect of (2) is that the stabilization time of our algorithms depends on α which is a bound on MAD that could potentially be very far from MAD. Of course, the absence of global knowledge on any network parameter would imply solutions using infinite local memories. In contrast, it would be interesting to study whether there exist solutions to our problems that use some global knowledge, such as α, to have bounded process memories but whose stabilization times depend on actual parameters of the network rather than this knowledge.

Finally, a long-term perspective would be to investigate necessary and sufficient conditions for the self-stabilization of more problems and classes of problems in directed identified networks. Regarding this, a natural and interesting candidate is the asynchronous unison, which is a powerful tool to emulate synchronous solutions in asynchronous settings.

References

[AB98] Yehuda Afek and Anat Bremler-Barr. Self-stabilizing unidirectional network algorithms by power supply. *Chic. J. Theor. Comput. Sci.*, 1998, 1998.

[ACD+17] Karine Altisen, Alain Cournier, Stéphane Devismes, Anaïs Durand, and Franck Petit. Self-stabilizing leader election in polynomial steps. *Inf. Comput.*, 254:330–366, 2017.

[ACDD23] Karine Altisen, Alain Cournier, Geoffrey Defalque, and Stéphane Devismes. Self-stabilizing synchronous unison in directed networks. In *24th International Conference on Distributed Computing and Networking, ICDCN 2023, Kharagpur, India, January 4-7, 2023*, pages 115–124. ACM, 2023.

[ADDP19] Karine Altisen, Stéphane Devismes, Swan Dubois, and Franck Petit. *Introduction to Distributed Self-Stabilizing Algorithms*. Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool Publishers, 2019.

[ADG91] Anish Arora, Shlomi Dolev, and Mohamed G. Gouda. Maintaining digital clocks in step. *Parallel Processing Letters*, 1:11–18, 1991.

[AG94] Anish Arora and Mohamed G. Gouda. Distributed reset. *IEEE Trans. Computers*, 43(9):1026–1038, 1994.

[AK93] Sudhanshu Aggarwal and Shay Kutten. Time optimal self-stabilizing spanning tree algorithms. In R. K. Shyamasundar, editor, *Foundations of Software Technology and Theoretical Computer Science, 13th Conference, Bombay, India, December 15-17, 1993, Proceedings*, volume 761 of *Lecture Notes in Computer Science*, pages 400–410. Springer, 1993.

[BDPT09] Samuel Bernard, Stéphane Devismes, Maria Gradinariu Potop-Butucaru, and Sébastien Tixeuil. Optimal deterministic self-stabilizing vertex coloring in unidirectional anonymous networks. In *23rd IEEE International Symposium on Parallel and Distributed Processing, IPDPS 2009, Rome, Italy, May 23-29, 2009*, pages 1–8. IEEE, 2009.

[BK07] Janna Burman and Shay Kutten. Time optimal asynchronous self-stabilizing spanning tree. In Andrzej Pelc, editor, *Distributed Computing, 21st International Symposium, DISC 2007, Lemesos, Cyprus, September 24-26, 2007, Proceedings*, volume 4731 of *Lecture Notes in Computer Science*, pages 92–107. Springer, 2007.

[Bou07] Christian Boulinier. *L'unisson. (The unison)*. PhD thesis, University of Picardie Jules Verne, Amiens, France, 2007.

[BPV04] C. Boulinier, F. Petit, and V. Villain. When graph theory helps self-stabilization. In *23rd Annual Symposium on Principles of Distributed Computing, (PODC'04)*, pages 150–159, 2004.

[CD94] Zeev Collin and Shlomi Dolev. Self-stabilizing depth-first search. *Inf. Process. Lett.*, 49(6):297–301, 1994.

[CDV05] Alain Cournier, Stéphane Devismes, and Vincent Villain. A snap-stabilizing DFS with a lower space requirement. In Ted Herman and Sébastien Tixeuil, editors, *Self-Stabilizing Systems, 7th International Symposium, SSS 2005, Barcelona, Spain, October 26-27, 2005, Proceedings*, volume 3764 of *Lecture Notes in Computer Science*, pages 33–47. Springer, 2005.

[CDV06] Alain Cournier, Stéphane Devismes, and Vincent Villain. Snap-stabilizing PIF and useless computations. In *12th International Conference on Parallel and Distributed Systems, ICPADS 2006, Minneapolis, Minnesota, USA, July 12-15, 2006*, pages 39–48. IEEE Computer Society, 2006.

[CFG92] J.-M. Couvreur, N. Francez, and M. G. Gouda. Asynchronous unison (extended abstract). In *12th International Conference on Distributed Computing Systems, (ICDCS'92)*, pages 486–493, 1992.

[CYZG14] B. Chen, H. Yu, Y. Zhao, and P. B. Gibbons. The cost of fault tolerance in multi-party communication complexity. *Journal of the ACM*, 61(3):1–64, 2014.

[DDT06] Sylvie Delaët, Bertrand Ducourthial, and Sébastien Tixeuil. Self-stabilization with r-operators revisited. *Journal of Aerospace Computing, Information, and Communication (JACIC)*, 3(10):498–514, 2006.

[DGS99] Shlomi Dolev, Mohamed G. Gouda, and Marco Schneider. Memory requirements for silent stabilization. *Acta Informatica*, 36(6):447–462, 1999.

[DH97] Shlomi Dolev and Ted Herman. Superstabilizing protocols for dynamic distributed systems. *Chic. J. Theor. Comput. Sci.*, 1997, 1997.

[Dij74] Edsger W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Commun. ACM*, 17(11):643–644, 1974.

[DJ19] S. Devismes and C. Johnen. Self-stabilizing distributed cooperative reset. In *39th International Conference on Distributed Computing Systems, (ICDCS'19)*, pages 379–389, 2019.

[DLD+13] Ajoy Kumar Datta, Lawrence L. Larmore, Stéphane Devismes, Karel Heurtefeux, and Yvan Rivierre. Self-stabilizing small k-dominating sets. *Int. J. Netw. Comput.*, 3(1):116–136, 2013.

[DLP10] Ajoy Kumar Datta, Lawrence L. Larmore, and Hema Piniganti. Self-stabilizing leader election in dynamic networks. In Shlomi Dolev, Jorge Arturo Cobb, Michael J. Fischer, and Moti Yung, editors, *Stabilization, Safety, and Security of Distributed Systems - 12th International Symposium, SSS 2010, New York, NY, USA, September 20-22, 2010. Proceedings*, volume 6366 of *Lecture Notes in Computer Science*, pages 35–49. Springer, 2010.

[DLV11a] Ajoy Kumar Datta, Lawrence L. Larmore, and Priyanka Vemula. An o(n)-time self-stabilizing leader election algorithm. *JPDC*, 71(11):1532–1544, 2011.

[DLV11b] Ajoy Kumar Datta, Lawrence L. Larmore, and Priyanka Vemula. Self-stabilizing leader election in optimal space under an arbitrary scheduler. *Theor. Comput. Sci.*, 412(40):5541–5561, 2011.

[DR82] Danny Dolev and Rüdiger Reischuk. Bounds on information exchange for byzantine agreement. In Robert L. Probert, Michael J. Fischer, and Nicola Santoro, editors, *ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing, Ottawa, CanadaAugust 18-20, 1982*, pages 132–140. ACM, 1982.

[DT01] Bertrand Ducourthial and Sébastien Tixeuil. Self-stabilization with r-operators. *Distributed Comput.*, 14(3):147–162, 2001.

[EK21] Y. Emek and E. Keren. A thin self-stabilizing asynchronous unison algorithm with applications to fault tolerant biological networks. In *40th Annual Symposium on Principles of Distributed Computing, (PODC'23)*, pages 93–102, 2021.

[ER90] Shimon Even and Sergio Rajsbaum. Unison in distributed networks. In Renato M. Capocelli, editor, *Sequences*, pages 479–487, New York, NY, 1990. Springer New York.

[GH90] Mohamed G. Gouda and Ted Herman. Stabilizing unison. *Inf. Process. Lett.*, 35(4):171–175, 1990.

[HL99] Shing-Tsaan Huang and Tzong-Jye Liu. Self-stabilizing 2^m-clock for unidirectional rings of odd size. *Distributed Comput.*, 12(1):41–46, 1999.

[JADT02] Colette Johnen, Luc Onana Alima, Ajoy Kumar Datta, and Sébastien Tixeuil. Optimal snap-stabilizing neighborhood synchronizer in tree networks. *Parallel Processing Letters*, 12(3-4):327–340, 2002.

[KK13] Alex Kravchik and Shay Kutten. Time optimal synchronous self stabilizing spanning tree. In Yehuda Afek, editor, *Distributed Computing - 27th International Symposium, DISC 2013, Jerusalem, Israel, October 14-18, 2013. Proceedings*, volume 8205 of *Lecture Notes in Computer Science*, pages 91–105. Springer, 2013.

[KP93] Shmuel Katz and Kenneth J. Perry. Self-stabilizing extensions for message-passing systems. *Distributed Comput.*, 7(1):17–26, 1993.

[KY02] Hirotsugu Kakugawa and Masafumi Yamashita. Uniform and self-stabilizing fair mutual exclusion on unidirectional rings under unfair distributed daemon. *JPDC*, 62(5):885–898, 2002.

[MOY96] Alain J. Mayer, Rafail Ostrovsky, and Moti Yung. Self-stabilizing algorithms for synchronous unidirectional rings. In Éva Tardos, editor, *Proceedings of the Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, 28-30 January 1996, Atlanta, Georgia, USA*, pages 564–573. ACM/SIAM, 1996.

[Tix06] Sébastien Tixeuil. *Vers l'auto-stabilisation des systèmes à grande échelle*. Habilitation à diriger des recherches, Université Paris Sud - Paris XI, May 2006.