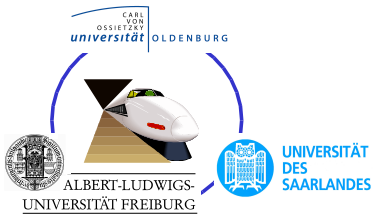


AVACS – Automatic Verification and Analysis of Complex Systems

REPORTS

of SFB/TR 14 AVACS

Editors: Board of SFB/TR 14 AVACS

Introducing Recurrence in Self-Stabilization (Revised Version)

by
Oday Jubran and Oliver Theel

AVACS Technical Report No. 101 (Revised Version)
April 2015
ISSN: 1860-9821

Publisher: Sonderforschungsbereich/Transregio 14 AVACS
(Automatic Verification and Analysis of Complex Systems)
Editors: Bernd Becker, Werner Damm, Bernd Finkbeiner, Martin Fränzle,
Ernst-Rüdiger Olderog, Andreas Podelski
ATRs (AVACS Technical Reports) are freely downloadable from `www.avacs.org`

Author(s) contact: Oday Jubran (`jubran@informatik.uni-oldenburg.de`).

Introducing Recurrence in Self-Stabilization*
(Revised Version)

Oday Jubran and Oliver Theel

Carl von Ossietzky Universität Oldenburg
26111 Oldenburg, Germany
{jubran,theel}@informatik.uni-oldenburg.de

Abstract. Self-stabilization ensures that a system converges to a legitimate execution in finite time, where a legitimate execution comprises a sequence of configurations satisfying some condition. In this work, we present the notion of recurrence, which denotes how frequent a condition is satisfied in an execution of a system. We use this notion in self-stabilization to address the convergence of a system to a behavior that guarantees a minimum recurrence. We apply this notion to show how the design of distributed mutual exclusion algorithms can be altered, to achieve a high recurrence of granting unique privilege to processes, under various time and space requirements.
As a particular contribution, we present a self-stabilizing mutual exclusion algorithm with stabilization time complexity of $\lceil D/2 \rceil - 1$ for synchronous executions and under any topology, where D is the diameter of the topology. This time complexity improves the state-of-the-art $\lceil D/2 \rceil$. In addition, we rectify an earlier proof that $\lceil D/2 \rceil$ is a lower bound, to conclude that $\lceil D/2 \rceil - 1$ is optimal for synchronous executions.

Keywords: Self-Stabilization · Recurrence · Dependability · Mutual Exclusion

1 Introduction

Self-stabilization [1, 2] ensures that a system's desired behavior is eventually obtained and never voluntarily violated regardless of the system's initial behavior. Self-stabilization comprises two basic properties: (1) Convergence: each execution reaches a configuration satisfying a desired property, regardless of the initial configuration. (2) Closure: Starting from a configuration satisfying the desired property, any execution does not reach a configuration violating the property. Towards efficient self-stabilizing algorithms, the related work concentrates on minimizing the convergence time and space of the algorithms.

The performance of systems, in general, is based not only on the space requirements and the convergence time to achieve safety, but also on the

* This work was partially supported by the German Research Foundation (DFG) as part of the Transregional Collaborative Research Center "Automatic Verification and Analysis of Complex Systems" (SFB/TR 14 AVACS, http://www.avacs.org/).

ability to perform high throughput. Indeed, minimizing stabilization time and space requirements may reduce the system throughput. As a concrete example, consider mutual exclusion algorithms with a synchronous daemon: the algorithms of Dijkstra [1] stabilize in n steps, where n is the number of processes. If the algorithms are applied on ring topologies, then in each step, one process is privileged to access the critical section. In contrast, given a graph, the algorithm in [3] stabilizes in $\lceil D/2 \rceil$ steps, where D is the diameter of the graph. This time complexity improves the one of Dijkstra's algorithms. However, the algorithm in [3] performs many steps in which no process is privileged. This example raises the need to focus on the properties that do not necessarily need to be satisfied in all configurations of an execution, in self-stabilization.

Contribution. We introduce *recurrence* in self-stabilization, which addresses (1) how frequent a global condition is satisfied in an execution, and (2) the convergence time to reach a configuration, from which a minimum recurrence is guaranteed. We apply this notion to show how the design of particular synchronous mutual exclusion algorithms can be altered to achieve high recurrence of granting unique privilege to processes under various time and space requirements. As a particular contribution, we present a self-stabilizing mutual exclusion algorithm, whose convergence time to mutual exclusion is $\lceil D/2 \rceil - 1$ for synchronous executions and any topology, where D is the topology diameter. This complexity, to the best of our knowledge, improves the state-of-the-art $\lceil D/2 \rceil$, introduced in [3]. In fact, our complexity result contradicts a claim of [3], that $\lceil D/2 \rceil$ is a lower bound. We rectify the lower bound proof given in [3] to conclude that $\lceil D/2 \rceil - 1$ is optimal.

Outline. The remainder of this section explores related work. Section 2 presents the system model and notation. Section 3 introduces *recurrence* and its properties. Section 4 presents the mutual exclusion algorithms. Section 5 presents the correctness proofs. Section 6 discusses some aspects of recurrence. Section 7 presents the optimality proof. Section 8 states the conclusion. Later, three Appendices present detailed proofs of some Lemmas and a Theorem.

1.1 Related Work

The performance of a self-stabilizing system is usually measured according to the convergence time, the space requirement, and the generality of the underlying topology. Tremendous work considered minimizing convergence time of self-stabilizing algorithms, e.g. [3–5]. Other works consider minimizing the space requirement, as in [6–9]. Others exploit variant environments, e.g. underlying topologies or schedulers, to enhance the performance [4, 9, 10].

Some approaches considered other notions of performance. Examples are [11–13]. In [11], the authors present a metric to measure the expected mean value of the convergence time. This value denotes the average case of convergence time, and is computed by probabilistic model checking. The work [12] considers the occurrence of transient faults during the convergence, and their effect on the convergence time. The work [13] defines and applies fault tolerance

measurements, such as reliability and availability, to evaluate self-stabilizing systems, also under the assumption of ongoing transient faults.

We use recurrence to alter the design of self-stabilizing mutual exclusion algorithms. This class of algorithms was pioneered by Dijkstra [1], and was followed by many approaches, such as [14–16]. Recently, Dubois et. al. [3] applied the notion of phase clock, or unison [8, 9], to design a mutual exclusion algorithm that achieves a convergence time complexity of $\lceil D/2 \rceil$ in synchronous environments, which has not been achieved beforehand. Our approach in the design of self-stabilizing mutual exclusion algorithms is similar to [3], as we believe that this approach is a good candidate to demonstrate recurrence.

2 System Model and Notation

We consider the classic shared memory model of Dijkstra [1].

The **topology** is modelled as a connected graph $G = (P, E)$ where P is a non-empty set of *processes*, and $E \subseteq P \times P$ is a set of *edges*. For each $(p, q) \in E$, p and q are said to be *neighbors*. The set of neighbors of a process p is denoted by $\mathcal{N}_p$. We also use $\mathcal{N}_p^*$ to denote $\{p\} \dot{\cup} \mathcal{N}_p$. A *path* in G is a finite sequence $p_0, ..., p_{k-1} \in P^*$ where $(p_i, p_{i+1}) \in E$ for $0 \leq i < k-1$. p_0 is said to be *linked* to p_{k-1} by this path. For each path $p_0, ..., p_{k-1} \in P^*$, $k-1$ (the number of edges in the path) is said to be the *length* of the path. The *distance* between two processes p_1, p_2 through G, denoted by $dist(G, p_1, p_2)$, is the length of the shortest path that links them through G. For each $i \in \mathrm{N}_0^+$, we use $Neigh(p, i)$ to denote the set of processes, whose distance to p is less or equal to i (including p). The *diameter of* G, denoted by D, is the maximum distance between any two processes. The *size* of G is $|P|$, denoted by n. Since we handle distributed systems, we assume that $n \geq 2$, and therefore, $D \geq 1$.

A **distributed algorithm** A is a set of *sub-algorithms*, each of which is a set of *guarded commands* of the form $label :: guard \rightarrow action$, where *label* is a unique identity of the command, *guard* is a boolean expression, and *action* is a set of assignment functions.

Each **process** p has variables, and a unique id in $\{0, ..., n-1\}$. A process p of id i is denoted by p_i. We use r_i or r_{p_i} to denote a variable r that belongs to a process p_i. Each process p_i runs a sub-algorithm A_i, such that for each guarded command $label :: guard \rightarrow action$ of A_i, *guard* is a condition over the variables of all $p_j \in \mathcal{N}_{p_i}^*$, and *action* can modify only the variables of p_i.

A **state** of a process is a valuation of its variables. A **configuration** γ is a vector of the states of all processes. An **execution** is a sequence of configurations $\gamma_1, \gamma_2, ...$ such that for $i \geq 1$, γ_{i+1} is reachable from γ_i by executing a subset of guarded commands without faults. We call (γ_i, γ_{i+1}) an *execution step*. An execution is *synchronous* iff for each execution step, each process with at least one enabled guarded command executes an action.

We present the classic definition of self-stabilization. A distributed algorithm A is said to be **self-stabilizing** wrt. a condition *con* over configurations iff for each execution $\gamma_0, \gamma_1, ...$, there exists finally a configuration γ_j, for $j \geq 0$,

such that each of $\gamma_j, \gamma_{j+1}, ...$ satisfies *con*. The ***con*-Convergence time** of A is defined as the maximum value of j among all executions of A.

3 Recurrence in Self-Stabilization

In this section, we extend the definition of self-stabilization. The extension involves the conditions that do not necessarily need to be satisfied by all configurations after convergence, but are needed to be satisfied by a minimum ratio of configurations. We present the notion of recurrence which denotes the ratio of configurations satisfying a condition in a finite execution.

Definition 1 (Recurrence). Let $\Xi : \gamma_0, ..., \gamma_{k-1}$ be a finite execution, and let *con* be a condition. The *recurrence* of *con* in Ξ, denoted by $Rec_{con}(\Xi)$, is the ratio $\Delta \in [0, 1] \subset \mathrm{Q}$ of the configurations satisfying *con* in Ξ. ♢

For example, let $\underline{\gamma}$ denote that a configuration γ satisfies *con*, and let Ξ be the following finite execution:

$$\Xi : \underline{\gamma_0}, \underline{\gamma_1}, \gamma_2, \gamma_3, \underline{\gamma_4}, \gamma_5, \underline{\gamma_6}, \gamma_7, \gamma_9, \underline{\gamma_9}, \gamma_{10}, \underline{\gamma_{11}},$$

the recurrence of *con* in Ξ can be computed by a simple division operation:

$$Rec_{con}(\Xi) = \frac{6}{12} = 0.5.$$

Our aim is to use the notion of recurrence in self-stabilization. In general, many self-stabilizing systems have infinite executions. The main thing to be concerned is the worst-case convergence time to a configuration γ, from which any finite execution prefix satisfies a minimum recurrence Δ of some condition *con*. For convenience, we write con_Δ to denote a minimum recurrence Δ of a condition *con*. We denote the worst-case convergence time to achieve con_Δ by *con_Δ-convergence time*.

Definition 2 (con_Δ). An execution $\Xi : \gamma_0, \gamma_1, ...$ is said to satisfy con_Δ iff for each $i \geq 0$, the recurrence of *con* in $\gamma_0, ..., \gamma_i$ ($Rec_{con}(\gamma_0, ..., \gamma_i)$) is greater or equal to Δ. ♢

Definition 3 (con_Δ-Convergence Time). Given an algorithm A, a condition *con*, and $\Delta \in [0, 1] \subset \mathrm{Q}$.

- An execution $\Xi : \gamma_0, \gamma_1, ...$ of A is said to *have a con_Δ-convergence time of c steps* iff c is the minimum number, such that the execution suffix $\gamma_c, \gamma_{c+1}, ...$ of Ξ satisfies con_Δ.
  It is also said that for each $j \geq c$, the execution Ξ *guarantees con_Δ-convergence in j steps*.
- The algorithm A is said to have a *con_Δ-convergence time* of c *steps* iff c is the maximum con_Δ-convergence time among all executions of A. ♢

i	Execution i (Ξ_i)	$con_{0.25}$-Convergence time for Ξ_i
1	$\Xi_1 : \underline{\gamma_0}, \gamma_1, \gamma_2, \gamma_3, \underline{\gamma_4}, \gamma_5, \gamma_6, \gamma_7, \underline{\gamma_8}, \gamma_9, \gamma_{10}, \gamma_{11}, \underline{\gamma_{12}}, \gamma_{13}, ...$	0
2	$\Xi_2 : \gamma_0, \gamma_1, \gamma_2, \gamma_3, \gamma_4, \underline{\gamma_5}, \gamma_6, \gamma_7, \gamma_8, \underline{\gamma_9}, \gamma_{10}, \gamma_{11}, \gamma_{12}, \underline{\gamma_{13}}, ...$	5
3	$\Xi_3 : \gamma_0, \gamma_1, \gamma_2, \gamma_3, \gamma_4, \gamma_5, \gamma_6, \underline{\gamma_7}, \underline{\gamma_8}, \underline{\gamma_9}, \underline{\gamma_{10}}, \underline{\gamma_{11}}, \underline{\gamma_{12}}, \underline{\gamma_{13}}, ...$	7

Table 1. Executions converging wrt. $con_{0.25}$ - $\underline{\gamma}$ denotes that γ satisfies con.

Note that the con_{Δ}-convergence time involves finite and infinite executions. To illustrate the definitions, consider the example given in Table 1. There exist three infinite executions of some algorithm A, explained as follows:

1. Execution Ξ_1: the condition con is satisfied in the configurations $\gamma_0, \gamma_4, \gamma_8, \gamma_{12}, ..$; i.e. once in every four subsequent configurations, starting from γ_0. By Definitions 2 and 3, a recurrence of 0.25 is achieved in 0 steps in Ξ_1. Indeed, it is the minimum recurrence guaranteed in Ξ_1.
2. Ξ_2: the condition con is satisfied in every four subsequent configurations, starting from γ_5. This implies that the recurrence 0.25 is achieved after 5 steps.
3. Ξ_3: starting from γ_7, the recurrence of con is 1. This case models the classic self-stabilization, where the desired condition to be satisfied holds in each state once the system has converged. Note that by Definition 3, the $con_{0.25}$-Convergence time is also 7 in Ξ_3.

Assuming that the three executions Ξ_1, Ξ_2, Ξ_3 are the unique ones for some algorithm A, the con_{Δ}-convergence time of A is 7 by Definition 3.

Considering distributed algorithms defined by guarded commands, measuring the frequency of running particular commands is performed by analyzing the recurrence of their guards. In the following, we apply recurrence to alter the design of synchronous mutual exclusion algorithms where the particular condition implies granting unique privilege to access the critical section.

4 Mutual Exclusion Algorithms

In this section, we use [9, Algorithm 3] to design synchronous mutual exclusion (ME) algorithms, with different recurrences of granting a unique privilege, under several space and convergence time complexities. Note that we consider showing the trade-offs in one environment: the synchronous environment. Our approach is similar to [3] in the design of mutual exclusion algorithms.

4.1 Finite Incrementing System

[9, Algorithm 3] implements an incrementing system with a finite domain. The aim of this system is to repeatedly increment a value belonging to each process, in

each step, while keeping the values of all processes equal. The finite incrementing system is shown in Figure 1. We explain it in detail as follows:

Let $K, \alpha \in \mathrm{N}_0^+$. Define the finite set $\mathcal{X} = \{-\alpha, ..., 0, ..., K-1\}$. Define a function $\varphi : \mathcal{X} \to \mathcal{X}$ such that $\varphi(a) = a + 1 \mod K$ if $a \geq 0$, and $\varphi(a) = a + 1$ otherwise. Now the pair $(\mathcal{X}, \varphi)$ is called a *finite incrementing system*, whose domain is $\mathcal{X}$, and its incrementing function is φ. Let $tail_{\mathcal{X}} = \{-\alpha, ..., 0\}$, $tail^*_{\mathcal{X}} = tail_{\mathcal{X}} \backslash \{0\}$, and $stab_{\mathcal{X}} = \{0, ..., K-1\}$.

This system is applied on connected graphs. Each process p has one variable r_p. The aim of this system is to achieve a *synchronous unison*: (1) to keep the values of r equal for all processes, and (2) to increment the value of r_p for each process p in each step, within $\{0, ..., K-1\}$.

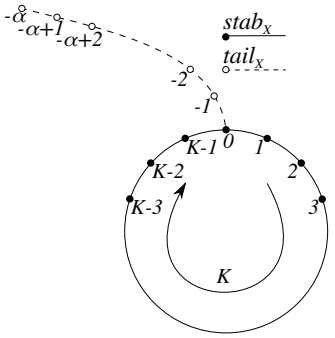


Fig. 1. $\alpha < 0$, $K > 0$

Definition 4 (Synchronous Unison - SU). Given a topology $G = (P, E)$ and an algorithm A, a specification SU, denoting *Synchronous Unison*, is said to be satisfied by an execution $\gamma_0, \gamma_1, ...$ of A over G iff:

1. Safety: in each configuration $\gamma_0, \gamma_1, ...$, for all $p, q \in P$, $r_p = r_q$.
2. Liveness: each process $p \in P$ increments r_p in each execution step of Ξ. $\diamondsuit$

There are 3 types of actions performed by each process p to achieve SU:

1. *NA* (*Normal Action*): if for all $q \in \mathcal{N}_p$, $r_p = r_q$, and $p \in stab_{\mathcal{X}}$, then p increments r_p.
2. *CA* (*Converge Action*): if there exists $q \in \mathcal{N}^*_p$ such that $r_q \in tail^*_{\mathcal{X}}$, then r_p is set to $r_g + 1$, where $g \in \mathcal{N}^*_p$ is the process with the minimum value of r among $\mathcal{N}^*_p$.
3. *RA* (*Reset Action*): if for all $q \in \mathcal{N}^*_p$, $r_q \in stab_{\mathcal{X}}$, and there exists $g \in \mathcal{N}_p$ such that $r_g \neq r_p$, then r_p is reset to $-\alpha$.

By [9], choosing $\alpha \geq D$, and $K \geq 2$, implies that the finite incrementing system is self-stabilizing wrt. SU in at most $2D$ steps.

Lemma 1. *([9, Corollary 3]).* Given a topology G, and a finite incrementing system $(\mathcal{X}, \varphi)$, where $\mathcal{X} = \{-\alpha, ..., 0, ..., K-1\}$, if $\alpha \geq D$ and $K \geq 2$, the system $(\mathcal{X}, \varphi)$ is self-stabilizing wrt. SU in $2D$ steps.

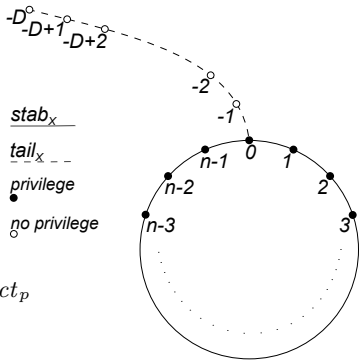
4.2 Algorithms

We use the finite incrementing system, given in Section 4.1, to design three self-stabilizing mutual exclusion algorithms. Mutual exclusion is a property satisfying that (1) in each configuration, at most one process is privileged, and (2) each process is privileged infinitely often.

Algorithm 1 *ME* Algorithm Based on the Classic Finite Incrementing System

Constants

$K = n$
$\alpha = D$
$stab_{\mathcal{X}} = \{0, ..., K-1\}$
$tail_{\mathcal{X}} = \{-\alpha, ..., 0\}$
$tail^*_{\mathcal{X}} = tail_{\mathcal{X}} \backslash \{0\}$



Predicates

$allCorrect_p \equiv \forall\, q \in \mathcal{N}_p \bullet r_p = r_q$
$privileged_p \equiv r_p = id \wedge allCorrect_p$
$normal_p \equiv r_p \in stab_{\mathcal{X}} \wedge allCorrect_p$
$converge_p \equiv \exists\, q \in \mathcal{N}_p \bullet r_q \in tail^*_{\mathcal{X}}$
$reset_p \equiv \forall\, q \in \mathcal{N}_p \bullet r_q \in stab_{\mathcal{X}} \wedge \neg allCorrect_p$

Guarded Commands

$NA :: normal_p \longrightarrow r_p := \varphi(r_p);$
$CA :: converge_p \longrightarrow r_p := \min\{\varphi(r_s) | s \in \mathcal{N}^*_p\};$
$RA :: reset_p \longrightarrow r_p := -\alpha;$

Definition 5 (Mutual Exclusion - *ME*). Let $G = (P, E)$ be a topology, let A be an algorithm, and let $privileged_p$ be a condition on each state of each process p. A specification *ME*, denoting Mutual Exclusion, is said to be satisfied by an execution $\Xi : \gamma_0, \gamma_1, ...$ of A over G iff:

1. Safety: if $privileged_p$ holds for a process $p \in P$ in a configuration γ_i, then for each process $q \in P \backslash \{p\}$, $privileged_q$ does not hold in γ_i.
2. Liveness: For each process $p \in P$, $privileged_p$ holds infinitely often in Ξ. $\diamondsuit$

Mutual Exclusion Algorithm Based on the Classic Finite Incrementing System We present Algorithm 1, which involves the incrementing system defined in Section 4.1. We have chosen $\alpha = D$, implying that Algorithm 1 is self-stabilizing wrt. *SU* in $2D$ steps. We set $K = n$. Hence, each value in $stab_{\mathcal{X}}$ corresponds to a process id, and vice versa. A process p_i is privileged iff two conditions hold: (1) $r_i = i$, and (2) r_i is equal to r_j, for each $p_j \in \mathcal{N}_{p_i}$.

By this setting, since the incrementing system converges to an execution satisfying *SU* and increments the value of r of each process within $\{0, ..., n-1\}$, it follows that *ME* holds. Given that each element of $\{0, ..., n-1\}$ corresponds to a process id, the recurrence of the condition ($privileged_p$) is 1.0.

Algorithm 1 stabilizes to *ME* in $D-1$ steps (see Section 5). We construct an example showing that *ME* is not guaranteed in less than $D-1$ steps: Figure 2 presents a topology consisting of 6 processes having ids from $0, ..., 5$. The initial configuration is γ_0. In the step (γ_0, γ_1), all processes execute *CA*. In (γ_1, γ_2), p_1, p_3, and p_5 execute *CA*, and the others execute *NA*. In (γ_2, γ_3), p_1, p_2, p_3, p_4 execute *NA*, and p_0, p_5 execute *RA*. Note that in γ_3, p_1 and p_2 are privileged. This implies that *ME* is not guaranteed in $3 = D-2$ steps.

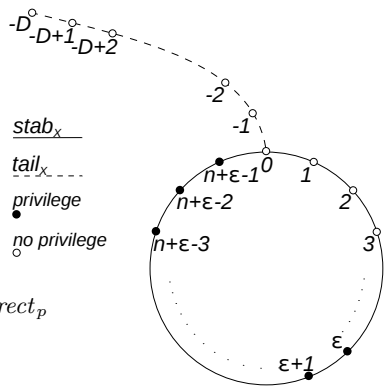
	1	3	5	0	4	2
γ_0	-2	-1	-1	-1	-1	-1
γ_1	-1	-1	0	0	0	0
γ_2	0	0	0	1	1	1
γ_3	1	1	-5	-5	2	2

Fig. 2. Example with $D = 5$

Algorithm 2 *ME* Algorithm with Optimal *ME*-Convergence Time $\lceil D/2 \rceil - 1$

Constants

$\epsilon = \lceil D/2 \rceil - 1$
$K = n + \epsilon$
$\alpha = D$
$stab_{\mathcal{X}} = \{0, ..., K-1\}$
$tail_{\mathcal{X}} = \{-\alpha, ..., 0\}$
$tail^*_{\mathcal{X}} = tail_{\mathcal{X}} \backslash \{0\}$



Predicates

$allCorrect_p \equiv \forall\, q \in \mathcal{N}_p \bullet r_p = r_q$
$privileged_p \equiv allCorrect_p \wedge r_p = id + \epsilon$
$normal_p \equiv r_p \in stab_{\mathcal{X}} \wedge allCorrect_p$
$converge_p \equiv \exists\, q \in \mathcal{N}^*_p \bullet r_q \in tail^*_{\mathcal{X}}$
$reset_p \equiv \forall\, q \in \mathcal{N}^*_p \bullet r_q \in stab_{\mathcal{X}} \wedge \neg allCorrect_p$

Guarded Commands

$NA :: normal_p \longrightarrow r_p := \varphi(r_p);$
$CA :: converge_p \longrightarrow r_p := \min\{\varphi(r_s) | s \in \mathcal{N}^*_p\};$
$RA :: reset_p \longrightarrow r_p := -\alpha;$

Mutual Exclusion Algorithm with $\lceil \frac{D}{2} \rceil - 1$ *ME*-Convergence Time We present Algorithm 2, whose significance is that its convergence time to *ME* is $\lceil D/2 \rceil - 1$ steps, using the finite incrementing system. From now on, we denote $\lceil D/2 \rceil - 1$ by ϵ in some positions for convenient presentation.

The idea behind this convergence time is as follows: (1) first, we define $stab_{\mathcal{X}}$ to include $n+\epsilon$ values. (2) Next, we assert that no process p is privileged if r_p has a value from $\{0, ..., \epsilon-1\}$; i.e. a processes may be privileged only if it has a value in $\{\epsilon, ..., n-1\}$. With this setting, the cases where mutual exclusion is violated within the $\{\epsilon, ..., D-2\}$ steps in Algorithm 1 - as in Figure 2 - are avoided.

However, since *NA* increments the variables within $\{0, ..., n+\epsilon-1\}$ and since no process p is privileged if $r_p \in \{0, ..., \epsilon-1\}$, the recurrence of *privileged* is less than 1.0 if $D > 2$. In fact, Algorithm 2 converges to an execution, where the minimum recurrence of *privileged* is $n/(n+\epsilon)$.

Mutual Exclusion Algorithm with Optimal *ME*-Convergence Time and 1.0 Recurrence of *privileged* We present Algorithm 3, whose significance

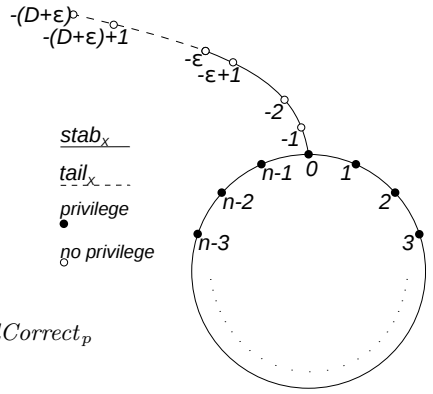
Algorithm 3 *ME* Algorithm with *ME*-Convergence Time $\lceil D/2 \rceil - 1$, and Achieving 1.0 Recurrence of *privileged*

Constants

$\epsilon = \lceil D/2 \rceil - 1$
$K = n$
$\alpha = D + \epsilon$
$stab_{\mathcal{X}} = \{-\epsilon, ..., 0, ..., K-1\}$
$tail_{\mathcal{X}} = \{-\alpha, ..., -\epsilon\}$
$tail^*_{\mathcal{X}} = tail_{\mathcal{X}} \backslash \{-\epsilon\}$

Predicates

$allCorrect_p \equiv \forall\, q \in \mathcal{N}_p \bullet r_p = r_q$
$privileged_p \equiv r_p = id \wedge allCorrect_p$
$normal_p \equiv r_p \in stab_{\mathcal{X}} \wedge allCorrect_p$
$converge_p \equiv \exists\, q \in \mathcal{N}^*_p \bullet r_q \in tail^*_{\mathcal{X}}$
$reset_p \equiv \forall\, q \in \mathcal{N}^*_p \bullet r_q \in stab_{\mathcal{X}} \wedge \neg allCorrect_p$



Guarded Commands

$NA :: normal_p \longrightarrow r_p := \varphi(r_p);$
$CA :: converge_p \longrightarrow r_p := \min\{\varphi(r_s) | s \in \mathcal{N}^*_p\};$
$RA :: reset_p \longrightarrow r_p := -\alpha;$

is that besides that it's convergence time to *ME* is $\lceil D/2 \rceil - 1$, the algorithm converges to an execution, in which the recurrence of granting a privilege is 1.0.

To obtain this significance, we extend the finite incrementing system as follows: (1) we define $tail_{\mathcal{X}} = \{-(D+\epsilon), ..., -\epsilon\}$. (2) Then, we extend the set $stab_{\mathcal{X}}$ to $\{-\epsilon, ..., 0, ..., n-1\}$. (3) We set that no process p may be privileged if r_p has a value from $\{-\epsilon, ..., -1\}$, and a process may be privileged if its value is within $\{0, ...n-1\}$. (4) After finite number of steps, The command *NA* increments the variables only within $\{0, ...n-1\}$ to guarantee 1.0 recurrence of *privileged*.

5 Correctness and Time Complexity

In this section, we present the correctness and complexity proofs of Algorithms 1-3: Section 5.1 considers the property *SU*, Section 5.2 considers *ME*, and Section 5.3 considers recurrence. For a convenient notation, for a process p_j, we use r^i_j or $r^i_{p_j}$ to denote the value of r_{p_j} in a configuration γ_i.

5.1 Self-Stabilization wrt. *SU*

We show that the algorithms 1-3 are self-stabilizing wrt. *SU* in $2D$ steps. Note that this holds straightforward for the algorithms 1 and 2 by Lemma 1. The following Lemmas, in this section, apply to the algorithms 1-3.

Lemma 2 (*SU* Closure). If *SU* holds in a configuration γ_i, then *SU* holds also in γ_{i+1}.

PROOF: If SU holds in γ_i, then by Def. 4, for all $p, q \in p$, $r_p^i = r_q^i$. We distinguish two cases: (1) $r_p^i \in tail_{\mathcal{X}}^*$. Each process increments its value by running CA, implying that $r_p^{i+1} = r_q^{i+1}$. (2) $r_p^i \in stab_{\mathcal{X}}$. Each process increments its value by running NA, implying that $r_p^{i+1} = r_q^{i+1}$. The safety property of SU holds. If the safety property holds in a configuration, then (1) each process has exactly one enabled command from NA and CA, and (2) RA is not enabled. Each of the possibly enabled commands increments the value of its corresponding process. The liveness property of SU holds. □

LEMMA 3. Given an execution $\gamma_0, \gamma_1, ...$, if no process executes RA within $(\gamma_0, \gamma_1), ..., (\gamma_{D-1}, \gamma_D)$, SU holds in γ_D.

PROOF: Let $(\gamma_0, \gamma_1), ..., (\gamma_{D-1}, \gamma_D)$, be an execution, within which no process executes RA. If SU holds in any γ_i, then SU holds in all $\gamma_{i+1}, \gamma_{i+2}, ...$ Assume that SU does not hold in γ_0. Let $p_i \in p$ be a process, such that r_i^0 is the minimum among all processes. We distinguish two cases. (1) $r_i^0 \in stab_{\mathcal{X}}$. In this case, for each process $p' \in p \backslash \{p_i\}$, $p' \in stab_{\mathcal{X}}$ holds. Since γ_0 does not satisfy SU, there exist two neighboring processes p_a, p_b, such that $r_a^0 \neq r_b^0$, and $r_a^0, r_b^0 \geq \min(stab_{\mathcal{X}})$. By definition, p_a and p_b execute RA in (γ_0, γ_1), which contradicts the assumption. This case does not exist. (2) $r_i^0 \in tail_{\mathcal{X}}^*$. Given that no process executes RA in (γ_0, γ_1) by assumption, p_i executes either NA or CA, implying that $r_i^1 = r_i^0 + 1$. If $r_i^1 \in stab_{\mathcal{X}}$, then SU holds, following the argument of the first case. Otherwise, $r_i^1 \in tail_{\mathcal{X}}^*$, and by definition of CA and NA, each process $p_j \in \mathcal{N}_{p_i}$ executes CA in (γ_0, γ_1), and therefore, $r_j^1 = r_i^1$. All other processes $p_s \notin \mathcal{N}_{p_i}^*$ execute either CA or NA. By definition of these guarded commands, and since r_i^0 is the minimum, it follows that $r_s^1 \geq r_i^1$. Now all processes at distance 1 of p_i have the same value of r_i^1, and r_i^1 is the minimum. Inductively, the processes at distance 2, 3,... of p_i perform similar steps. Since the distance between p_i and any process is bounded by D, all processes have the same value of r_i after D steps. □

LEMMA 4. For each execution $\gamma_0, \gamma_1, ...$, no process executes RA after γ_D.

PROOF: The algorithms 1 and 2 implement the incrementing system [9]. By [9, Theorem 9], starting from any configuration γ_0, choosing $\alpha \geq D$ implies that there is no process that executes RA after D steps. This implies that the Lemma holds for the algorithms 1 and 2. Regarding Algorithm 3: by [9], the values of α and k correspond to the sizes of $tail_{\mathcal{X}}^*$ and $stab_{\mathcal{X}}$, respectively. Choosing $\alpha \geq D$ corresponds to choosing $|tail_{\mathcal{X}}^*| \geq D$, which holds for Algorithm 3. The Lemma holds for Algorithm 3.

LEMMA 5. The algorithms 1-3 are self-stabilizing wrt. SU in $2D$ steps.

Proof Sketch (cf. Appendix-1): The variables α and K correspond to $|tail_{\mathcal{X}}^*|$ and $|stab_{\mathcal{X}}|$, respectively. Lemma 1 states that choosing $\alpha \geq D$ and $K \geq 2$ implies the Lemma. In the algorithms 1-3, $|tail_{\mathcal{X}}^*| \geq D$ and $|stab_{\mathcal{X}}| \geq 2$. In addition, the function φ increments r_p for each processes p within $\{0, ..., K-1\}$. □

5.2 Self-Stabilization wrt. *ME*

We show that Algorithm 1 is self-stabilizing wrt. *ME* in $D-1$ steps, and each of the algorithms 2 and 3 is self-stabilizing wrt. *ME* in $\lceil D/2 \rceil - 1$ steps.

LEMMA 6. Given any of the algorithms 1-3, if *SU* holds in a configuration γ_0, then *ME* holds in γ_0.

PROOF: If *SU* holds in γ_0, then by Def. 4, $\forall\, p \in P, q \in \mathcal{N}_p \bullet r_p^0 = r_q^0$. A process p is privileged only if $r_p = id_p(+\epsilon)$, by definition of $privileged_p$, which implies that at most one process is privileged if *SU* holds (safety). By Lemma 2, *SU* holds in all configurations $\gamma_1, \gamma_2, ...$ By definition of the commands, the possibly enabled commands for any process in any of the configurations $\gamma_0, \gamma_1, ...$ are *CA* and *NA*. Each of the commands increments r_p of each process p by 1 in each step. By construction of φ, after finite steps, the values of r_p are incremented within $\{0, ..., K-1\}$. Since $\{0, ..., K-1\}$ reflects all processes' ids, it follows that the predicate $privileged_p$ holds infinitely often for each process p (Liveness). □

THEOREM 1. Algorithm 1 is self-stabilizing wrt. *ME* in $D-1$ steps.

Proof Sketch (cf. Appendix-2): First, by Lemma 5, *SU* holds in $2D$ steps, and therefore, by Lemma 6, the safety and liveness properties of *ME* hold. Second, Algorithm 1 guarantees that in D steps, for each process p and q, if $r_p, r_q \geq \min(stab_{\mathcal{X}})$, then $r_p = r_q$ holds. The second argument implies that in $D-1$ steps, at most one process is privileged. □

To show that each of the algorithms 2 and 3 is self-stabilizing wrt. *ME* in $\lceil D/2 \rceil - 1$ steps, we follow an approach inspired from [3]. Basically, we extend the term *island* borrowed from [3] to use it in the proofs. Note that we do not necessarily use it the same way as in [3].

An island denotes a maximal strict subset of processes, whose values of r are equal, and are in $stab_{\mathcal{X}}$. By definition of *privileged* in the algorithms 1-3, at most one process may be privileged in any island.

DEFINITION 6 (ISLAND). Given a configuration γ_i of a topology $G = (P, E)$, an *island* is a maximal (wrt. inclusion) non-empty strict subset $I \subsetneq P$, such that $\forall\, p, q \in I \bullet r_p^i = r_q^i \wedge r_p^i \in stab_{\mathcal{X}}$. ♢

DEFINITION 7. Let I be an Island in a topology G.

- I is said to be an *init-island* iff $\forall p \in I \bullet r_p = \min(stab_{\mathcal{X}})$, and a *non-init-island* otherwise.
- The *border of* I ($Border(I)$) is defined as follows: $Border(I) = \{p \in I | \exists\, q \in P \backslash I \bullet q \in \mathcal{N}_p\}$. The *depth* of I ($depth(I)$) is defined as follows: $depth(I) = \max\{\min\{dist(G, p, q) | q \in Border(I)\} | p \in I\}$ ♢

LEMMA 7. Given an execution step (γ_i, γ_{i+1}) of Algorithm 2, if a process p belongs to a non-init-island I in γ_i, then p belongs to an island of depth $depth(I) + 1$ in γ_{i-1}.

PROOF: Let p be a process belonging to a non-init-island I in γ_i. Assume, by contradiction, that p does not belong to any island in γ_{i-1}. This implies that $r_p^{i-1} \in tail_{\mathcal{X}}^*$ holds, implying that p may execute only CA in (γ_{i-1}, γ_i), and then $r_p^i \in tail_{\mathcal{X}}$ must hold. By Def. 6, this implies that p either belongs to an init-island or to no island in γ_i. This contradicts that p belongs to a non-init-island in γ_i.

Let $depth(I) = h$. Assume, by contradiction, that p belongs to an island I' in γ_{i-1}, such that $depth(I') \neq h + 1$. By Def. 7, each $q \in Border(I')$ executes RA or CA in (γ_{i-1}, γ_i). This decreases $depth(I')$ by 1 concerning that each process $g \in I' \backslash Border(I')$ executes NA in (γ_{i-1}, γ_i) - by definition of the guarded commands - implying that $I' \backslash Border(I') = I$. This implies that the depth of I that contains p in γ_i is not equal to $(h + 1) - 1 = h$, which contradicts the assumption that $depth(I) = h$. □

LEMMA 8. In each configuration, there exists at most one island I such that $depth(I) \geq \lceil \frac{D}{2} \rceil$.

PROOF: Assume, by contradiction, that there is a configuration with two different islands I and I', such that $depth(I), depth(I') \geq \lceil \frac{D}{2} \rceil$. By Def. 7, there exists two processes $p \in I$, $p' \in I'$ such that: $\min\{dist(p, q) | q \in Border(I)\} \geq \lceil \frac{D}{2} \rceil$, and $\min\{dist(p', q') | q' \in Border(I')\} \geq \lceil \frac{D}{2} \rceil$. Let $path_1 : p, ..., p_i, ..., p'_i ..., p'$ be the shortest path between p and p', where $p_i \in Border(I)$, and $p'_i \in Border(I')$. By definition of the graph, the length of $path_1$ is less or equal to D. By Def. 6, $dist(G, p_i, p'_i) \geq 1$. By construction, $dist(G, p, p_i), dist(G, p', p'_i) \geq \lceil \frac{D}{2} \rceil$. This implies that the distance between p and p' through this path is greater or equal to $D + 1$. This contradicts that $path_1$ is the shortest one between p and p'. □

THEOREM 2. Algorithm 2 is self-stabilizing wrt. ME in $\lceil D/2 \rceil - 1$ steps.

PROOF: Let γ_0 be an initial configuration. By Lemma 5, SU holds in execution $\gamma_{2D}, \gamma_{2D+1}, ...$, which, by Lemma 6, implies that ME holds in execution $\gamma_{2D}, \gamma_{2D+1}, ...$ This implies that the liveness property of ME holds. It remains to show that for $\lceil \frac{D}{2} \rceil - 1 \leq i < 2D$, the safety property of ME holds in γ_i; i.e. at most one process is privileged in γ_i.

Let $\epsilon = \lceil \frac{D}{2} \rceil - 1$. Assume by contradiction that there exist two processes $p_a, p_b \in P$, such that p_a and p_b are privileged in γ_i. Since: (1) By definition of *privileged*: $r_a^i \neq r_b^i$, and $r_a^i, r_b^i \geq \epsilon$, and (2) by definition, $stab_{\mathcal{X}} = \{0, ..., n + \epsilon - 1\}$, it follows by definition of the guarded commands that r_a and r_b have values greater or equal to 0 in the configurations $\gamma_\epsilon, ..., \gamma_i$, and p_a and p_b execute only NA in each step of execution $(\gamma_{i-\epsilon}, \gamma_{i-\epsilon+1}), ..., (\gamma_{i-1}, \gamma_i)$. By definition of NA:

$r_a^i = \varphi(r_a^{i-1})$, and $r_b^i = \varphi(r_b^{i-1})$
$r_a^i = \varphi^2(r_a^{i-2})$, and $r_b^i = \varphi^2(r_b^{i-2})$
...
$r_a^i = \varphi^\epsilon(r_a^{i-\epsilon})$, and $r_b^i = \varphi^\epsilon(r_b^{i-\epsilon})$

Since $r_a^i \neq r_a^i$, it holds by the above analysis and by the definition of φ that $r_a^{i-\epsilon} \neq r_b^{i-\epsilon}$, which, by Def. 6, implies that p_a and p_b belong to two different islands in $\gamma_{i-\epsilon}$ (*).

By definition of $privileged_{p_a}$, for each $q \in \mathcal{N}_{p_a}$, $r_q^i = r_a^i$ holds. The same holds for p_b. By Def. 7, this implies that each of the processes p_a, p_b belongs to an

island of depth greater or equal to 1. Let $h_a \geq 1$ (resp. $h_b \geq 1$) be the depth of the island to which p_a (resp. p_b) belongs in γ_i. Since $r_a^i, r_b^i \geq \epsilon$, it follows by Lemma 7 that:
in γ_{i-1}, p_a (resp. p_b) belongs to a non-init-island of depth $h_a + 1$ (resp. $h_b + 1$),
in γ_{i-2}, p_a (resp. p_b) belongs to a non-init-island of depth $h_a + 2$ (resp. $h_b + 2$),
...
in $\gamma_{i-\epsilon}$, p_a (resp. p_b) belongs to an island of depth $h_a + \epsilon$ (resp. $h_b + \epsilon$).
Since $h_a, h_b \geq 1$, and $\epsilon = \lceil \frac{D}{2} \rceil - 1$ by construction, it follows that $(h_a + \epsilon), (h_b + \epsilon) \geq \lceil \frac{D}{2} \rceil$. By Lemma 8, there exists at most one island I'', such that $depth(I'') \geq \lceil \frac{D}{2} \rceil$. This implies that p_a and p_b belong to the same island in $\gamma_{i-\epsilon}$. This is a contradiction to (*). □

THEOREM 3. Algorithm 3 is self-stabilizing wrt. *ME* in $\lceil D/2 \rceil - 1$ steps.
Proof Sketch: Similar to the proof argument for Theorem 2, with a difference that $stab_{\mathcal{X}}$ is equal to $\{-\epsilon, ..., n-1\}$ instead of $\{0, ..., n+\epsilon-1\}$, and $tail_{\mathcal{X}}$ is equal to $\{-(D+\epsilon), ..., -\epsilon\}$ instead of $\{-D, ..., 0\}$. □

5.3 Self-Stabilization wrt. Δ Recurrence of *priv*

In this section, we show that the algorithms 1-3 stabilize to Δ recurrence of granting unique privilege to any process p; i.e. Δ recurrence of $privileged_p$ for any p, where Δ equals 1.0 for each of the algorithms 1 and 3, and $n/(n + \lceil D/2 \rceil - 1)$ for Algorithm 2. We use *priv* to denote the condition that there exists a process p, such that predicate $privileged_p$ holds.

DEFINITION 8 (*priv*). A condition *priv* is said to be satisfied in a configuration γ of a topology $G = (P, E)$ iff there exists a process $p \in P$ such that $privileged_p$ holds in γ. ♢

First we consider recurrence in the algorithms 1 and 3.

LEMMA 9. For each configuration γ_0 of a topology $G = (P, E)$, if *SU* holds, and for each process $p \in P$, $r_p^0 \in \{0, ..., K-1\}$, then for each execution $\gamma_0, \gamma_1, ...$ of any of the algorithms 1 and 3, the condition $priv_{1.0}$ holds.
PROOF: By Def. 2, $priv_{1.0}$ holds in $\gamma_0, \gamma_1, ...$ iff *priv* holds in each configuration $\gamma_0, \gamma_1, ...$ We show that this holds by induction: (1) For the base case γ_0, by hypothesis, *SU* holds, and for each process p, $r_p^i \in \{0, ..., K-1\}$ holds. Given that each element in $\{0, ..., K-1\}$ belongs to a process *id*, by definition of *privileged*, one process is privileged in γ_i, implying that *priv* holds in γ_i. (2) The inductive step: let γ_i be a configuration, in which *SU* and $r_p^i \in \{0, ..., K-1\}$ hold. We show that *priv* holds in the following configuration γ_{i+1}. By Lemma 2, *SU* holds in γ_{i+1}. By definition of the guarded commands, only command *NA* is enabled, whose result keeps $r_p^{i+1} \in \{0, ..., K-1\}$ for each process p. This implies that *priv* holds in γ_{i+1}. □

THEOREM 4. Each of the algorithms 1 and 3 is self-stabilizing wrt. $priv_{1.0}$ in $\alpha + D$ steps.

PROOF: Let γ_0 be a configuration. By Lemma 4, for each execution $\gamma_0, \gamma_1, ...$ of any of the algorithms 1 and 3, no process executes RA after γ_D. Let p be a process, such that r_p is the minimum among all processes in γ_D. If $r_p^D \in stab_{\mathcal{X}}$, then SU holds, because no process runs RA after γ_D. If $r_p^D \in tail_{\mathcal{X}}^*$, in the step (γ_D, γ_{D+1}), $r_p^{D+1} = r_p^D + 1$, and since no process executes RA, then by definition of the guarded commands, for each process $q \in P$, $r_q^{D+1} \geq r_p^{D+1}$ holds. Inductively, in $\gamma_{\alpha+D}$, it holds that $r_q^{\alpha+D} \geq 0$. By Lemma 5, each of the algorithms 1 and 3 is self-stabilizing wrt. SU in $2D$ steps. Since $\alpha \geq D$, it follows that SU holds in $\gamma_{\alpha+D}$, by Lemma 9, $priv_{1.0}$ holds in $\gamma_{\alpha+D}, \gamma_{\alpha+D+1}, ...$ □

Note that $\alpha = D$ in Algorithm 1, and $\alpha = \lceil 1.5D \rceil - 1$ in Algorithm 3. This implies that $priv_{1.0}$ holds in $2D$ steps for Algorithm 1, and $\lceil 2.5D \rceil - 1$ steps for Algorithm 3. In the following, we consider recurrence in Algorithm 2.

LEMMA 10. Let $\Delta = n/(n+\epsilon)$. In each execution $\gamma_0, \gamma_1, ...$ of Algorithm 2 over a topology $G = (P, E)$, if for each process $p \in P$, $r_p^0 = \epsilon$ holds, then the condition $priv_{\Delta}$ holds.
Proof Sketch (cf. Appendix-3): If for each process $p \in P \bullet r_p^0 \in \epsilon$ holds, then NA is executed by all processes in all following steps, and one process is privileged in each of $\gamma_0, ..., \gamma_{n-1}$, yielding n privileges in n configurations. This implies that in any sequence $\gamma_0, ..., \gamma_j$, where $j \in \{n, ..., n+\epsilon-1\}$, the recurrence of $privileged_p$ is greater or equal to $n/(n+\epsilon) = \Delta$. Now the value of r_p in $\gamma_{n+\epsilon-1}$ equals $r_p^0 - 1$. This implies that the sequence $\gamma_0, ..., \gamma_{n+\epsilon-1}$ is repeated infinitely, implying that for all $i \geq 0$, the recurrence of the finite sequence $\gamma_0, ..., \gamma_i$ is greater or equal to Δ. This implies that $priv_{\Delta}$ holds in $\gamma_0, \gamma_1, ...$ □

THEOREM 5. Let $\Delta = n/(n+\epsilon)$. Algorithm 2 is self-stabilizing wrt. $priv_{\Delta}$ in $\max\{(\lceil 2.5D \rceil - 1), (n + \lceil D/2 \rceil - 2)\}$ steps.
PROOF: Let γ_0 be an initial configuration. We distinguish two cases: (1) SU holds in γ_0. In this case, for all processes p, q, $r_p^0 = r_q^0$, and in any following configuration, only NA or CA may be enabled for any process. If $r_p^0 \in \{-D, ..., \epsilon\}$, then in at most $D + \epsilon$ $(= \lceil 1.5D \rceil - 1)$ steps, a configuration is reached where $r_p = \epsilon$. Otherwise, if $r_p^0 \in \{\epsilon + 1, ..., n+\epsilon-1\}$, then a configuration, where $r_p = \epsilon$ is reached in st most $n+\epsilon-1$ $(= n + \lceil D/2 \rceil - 2)$ steps. (2) SU does not hold in γ_0. In this case, for any process p with the minimum value of r in γ_0, if $r_p^0 \in stab_{\mathcal{X}}$, then within D steps, at least one process executes RA. Otherwise, $r_p^0 \in tail_{\mathcal{X}}^*$. In both situations, by Lemmas 3 and 4, after at most $D + \alpha = 2D$ steps, a configuration γ_j is reached, in which SU holds and for each process p, $r_p^j = 0$. From γ_j, the processes execute NA, and after ϵ steps, $r_p^{j+\epsilon} = \epsilon$. This sums up to $\lceil 2.5D \rceil - 1$ steps. Considering both cases, by Lemma 10 the convergence time to achieve $priv_{\Delta}$ is $\max\{(\lceil 2.5D \rceil - 1), (n + \lceil D/2 \rceil - 2)\}$ □

6 Discussion

Sections 4 and 5 presented the algorithms and their time complexities. Table 2 summarizes the time and space complexities of the algorithms 1-3.

Alg.	*ME*-Convergence Time	Recurrence Δ	$priv_{\Delta}$-Convergence Time	Space
1	$D-1$	1.0	$2D$	$n+D$
2	$\lceil D/2 \rceil -1$	$n/(n+\lceil D/2 \rceil -1)$	$\max\{(\lceil 2.5D \rceil -1), (n+\lceil D/2 \rceil -2)\}$	$n+\lceil 1.5D \rceil -1$
3	$\lceil D/2 \rceil -1$	1.0	$\lceil 2.5D \rceil -1$	$n+\lceil 1.5D \rceil -1$

Table 2. Time and Space Complexities for Algorithms 1-3

As observed, reducing the convergence time or space of a self-stabilizing algorithm may affect the performance of the algorithm: a fast recovery to a legitimate configuration does not imply a fast installation of the desired recurrence. This tradeoff is obvious concerning the algorithms 1 and 2, where both are based on the finite incrementing system presented in Section 4.1. On one hand, Algorithm 1 guarantees that in $2D$ steps, both *ME* and $priv_{1.0}$ are achieved, while Algorithm 2 does not. On the other hand, Algorithm 2 converges to *ME* faster than Algorithm 1 does. Algorithm 3 is an enhanced version of Algorithm 2: it can still achieve 1.0 recurrence of *priv*. However, it may not converge to $priv_{1.0}$ in $2D$ steps as Algorithm 1 guarantees.

Recurrence can be extended to capture other aspects of the performance of self-stabilizing systems. For example, in practice, some recurrence of a condition may be achieved in any execution starting from particular configurations, while it may not be achieved starting from others. This raises the need to control the stabilization path, not only to achieve a safety, but also to achieve the desired recurrence. As another example, recurrence can be extended to evaluate the number of particular actions in one step. A good candidate of such an extension is the class of local mutual exclusion algorithms, e.g. [16], in which multiple processes are safely granted privileges in the same configuration.

7 Optimality of $\lceil D/2 \rceil -1$ *ME*-Convergence Time for Synchronous Executions

We refine the proof of the lower bound *ME*-convergence time complexity for synchronous executions, given in [3]. We show that $\lceil D/2 \rceil -1$ is optimal.

We borrow two definitions and one Lemma from [3].

Definition 9 (Local State [3]). Given a configuration γ, a process p and an integer $0 \leq k \leq D$. The *k-local* state of p in γ (denoted by $\gamma_{p,k}$) is the configuration of the communication subgraph $G' = (P', E')$ induced by $P' = \{p' \in P | dist(G, p, p') \leq k\}$ defined by $\forall\, p' \in P' \bullet \gamma_{p,k}(p') = \gamma(p')$.[1] $\diamondsuit$

Definition 10 (Restrictions of an Execution [3]). Given an execution $\Xi = (\gamma_0, \gamma_1), (\gamma_1, \gamma_2), ...$ and a process p, the *restriction of Ξ to p* (denoted by Ξ_p) is defined by: $\Xi_p = (\gamma_0(p), \gamma_1(p)), (\gamma_1(p), \gamma_2(p)), ...$ $\diamondsuit$

[1] By definition, $\gamma_{p,0} = \gamma(p)$.

LEMMA 11 ([3]). Let γ, γ' be two configurations such that there exists a process p and an integer $1 \leq k \leq D$ satisfying $\gamma_{p,k} = \gamma'_{p,k}$. Let A be a self-stabilizing algorithm wrt. *ME* for synchronous executions. The restrictions to p of the prefixes of length k of the synchronous executions of A starting respectively from γ and γ' are equal.

In the following, we refine [3, Theorem 4], to show that $\lceil D/2 \rceil - 1$ is optimal for synchronous executions. Next, we point out a flaw in [3, Theorem 4].

THEOREM 6. The *ME*-convergence time of any self-stabilizing distributed algorithm wrt. *ME* is greater or equal to $\lceil D/2 \rceil - 1$ (if $D > 0$) for synchronous executions.

PROOF: The claim holds trivially for $D = 1$. In the following, we consider the cases where $D \geq 2$. Let A be a self-stabilizing distributed algorithm wrt. *ME*, and let t be the *ME*-convergence time of A. Assume, by contradiction, that $t < \lceil \frac{D}{2} \rceil - 1$ (Note that $\lceil \frac{D}{2} \rceil - 1 = \lceil \frac{D-2}{2} \rceil$).

Let $G = (P, E)$ be an arbitrary graph, and let p, q be two processes in P such that $dist(G, p, q) = D$. Let $\Xi = (\gamma_0, \gamma_1), (\gamma_1, \gamma_2), ...$ be an execution starting from a configuration γ_0.

By the liveness property of *ME*, Ξ contains an infinite suffix in which p (resp. q) is privileged infinitely often. Hence, there exists a configuration γ_i (resp. γ_j) such that p (resp. q) is privileged in γ_i (resp. γ_j) and $i > t$ (resp. $j > t$).

By construction, since $dist(G, p, q) = D$, and $D \geq 2$, it follows that there is a path $p, p', ..., q', q$, such that $dist(G, p', q') = D-2$ (p' and q' might be identical).

Since $t < \lceil \frac{D-2}{2} \rceil$, there exists at least one configuration γ'_0, such that $(\gamma'_0)_{p',t} = (\gamma_{i-t})_{p',t}$ and $(\gamma'_0)_{q',t} = (\gamma_{j-t})_{q',t}$. Let $\Xi' = (\gamma'_0, \gamma'_1), (\gamma'_1, \gamma'_2), ...$ be the synchronous execution of A starting from γ'_0.

By Lemma 11, one can deduce that the restriction to p' of the prefix of length t of Ξ' is the same as the one of the suffix of Ξ starting from γ_{i-t}. In addition, by definition of the graph, for all $g \in \mathcal{N}_p^*$, $dist(G, g, q) \geq dist(G, p', q)$, because the path $p, p', ..., q, q'$ is the shortest between p and q. This, analogously, implies that the restriction to g of the prefix of length t of Ξ' is the same as the one of the suffix of Ξ starting from γ_{i-t}.

By definition of a process, a process's variables are visible only to the process itself and its neighbors. Hence, the privilege condition can be defined only over the variables of the process and all its neighbors.

This, by construction of γ_i, implies that p is privileged in γ'_t. By the same deduction, q is also privileged in γ'_t. This contradiction leads to the result. □

In [3, Theorem 4], the issue, that the privilege condition of a process p may also cover the states of the neighbors of p, is missed, i.e. the privilege condition was considered as if it covers only the local state of p. With this consideration, [3, Theorem 4] concluded that the *ME*-convergence time is lower bounded by $\lceil D/2 \rceil$. This consideration is not necessarily required in the shared memory model.

8 Conclusion

We presented recurrence in self-stabilization, which addresses the ratio of satisfying a condition in an execution, and the convergence time to achieve this frequency. This notion explores the contrast of the convergence time to a legitimate behavior and the convergence time to a useful behavior. We designed and analyzed three self-stabilizing mutual exclusion algorithms with different time and space requirements in synchronous environments. In particular, we presented an algorithm that achieves $\lceil D/2 \rceil - 1$ convergence time complexity to mutual exclusion for synchronous executions and any topology. We have shown that this time complexity is optimal, by refining an earlier lower bound proof.

References

[1] E. W. Dijkstra. Self-Stabilizing Systems in Spite of Distributed Control. *Communi. of the ACM*, 17(11), 1974.
[2] S. Dolev. *Self-Stabilization*. The MIT Press, 2000.
[3] S. Dubois and R. Guerraoui. Introducing Speculation in Self-Stabilization - An Application to Mutual Exclusion. *CoRR*, abs/1302.2217, 2013.
[4] A. Cournier, A. K. Datta, f. Petit, and V. Villain. Optimal Snap-Stabilizing PIF Algorithms in Un-Oriented Trees. *J. High Speed Networks*, 14(2), 2005.
[5] A. Kravchik and S. Kutten. Time Optimal Synchronous Self Stabilizing Spanning Tree. In *DISC*. Springer, 2013.
[6] A. Bui, A. K. Datta, F. Petit, and V. Villain. Space Optimal PIF Algorithm: Self-Stabilized with no Extra Space. In *IPCCC*. IEEE, 1999.
[7] A. Bui, A. K. Datta, F. Petit, and V. Villain. State-Optimal Snap-Stabilizing PIF in Tree Networks. In *WSS*, 1999.
[8] C. Boulinier, F. Petit, and V. Villain. When Graph Theory Helps Self-Stabilization. In *PODC*. ACM, 2004.
[9] C. Boulinier, F. Petit, and V. Villain. Synchronous vs. Asynchronous Unison. *Algorithmica*, 51(1), 2008.
[10] A. Cournier, F. Petit, V. Villain, and A. K. Datta. Self-Stabilizing PIF Algorithm in Arbitrary Rooted Networks. In *ICDCS*, 2001.
[11] N. Fallahi, B. Bonakdarpour, and S. Tixeuil. Rigorous Performance Evaluation of Self-Stabilization Using Probabilistic Model Checking. In *SRDS*. IEEE, 2013.
[12] Y Nakaminami, H. Kakugawa, and T. Masuzawa. An Advanced Performance Analysis of Self-Stabilizing Protocols: Stabilization Time with Transient Faults During Convergence. In *IPDPS*. IEEE, 2006.
[13] A. Dhama, O. E. Theel, and T. Warns. Reliability and Availability Analysis of Self-stabilizing Systems. In *SSS*, volume 4280. Springer, 2006.
[14] S. Dolev, A. Israeli, and S. Moran. Self-Stabilization of Dynamic Systems Assuming Only Read/Write Atomicity. *Distributed Computing*, 7(1), 1993.
[15] R. Hadid and V. Villain. A New Efficient Tool for the Design of Self-Stabilizing ℓ-Exclusion Algorithms: The Controller. In *WSS*, 2001.
[16] A. K. Datta, S. Devismes, F. Horn, and L. L. Larmore. Self-Stabilizing k-out-of-ℓ Exclusion on Tree Networks. In *IPDPS*. IEEE, 2009.

Appendix 1. Proof of Lemma 5

The results of [9, Algorithm 3] state that: given a finite incrementing system $(\mathcal{X}, \varphi)$, if $\alpha \geq D$, and $K \geq 2$, then $(\mathcal{X}, \varphi)$ is self-stabilizing to SU in $2D$ steps. The algorithms 1 and 2 follow these specifications, implying that they are self-stabilizing wrt. SU in $2D$ steps. It remains to show that for Algorithm 3.

Algorithm 3 differs from [9, Algorithm 3] by shifting the intervals of $tail_{\mathcal{X}}$ and $stab_{\mathcal{X}}$, while keeping a mutual value between them. We show that doing so does not violate the stabilization to SU.

In the design of Algorithm 3, the mutual value between $tail_{\mathcal{X}}$ and $stab_{\mathcal{X}}$ is $-\epsilon$. In addition, $tail^*_{\mathcal{X}} = \{-(D+\epsilon), ..., -\epsilon - 1\}$, and $stab_{\mathcal{X}} = \{-\epsilon, ..., K-1\}$. By construction of Algorithm 3, $|tail^*_{\mathcal{X}}| \geq D$ and $|stab_{\mathcal{X}}| \geq 2$.

Regarding the safety property of SU: By Lemma 4, no process executes the command RA after γ_D. By Lemma 3, since no process executes RA after γ_D, it holds that the safety property of SU is achieved in γ_{2D}.

Regarding the liveness property of SU: in Algorithm 3, $|stab_{\mathcal{X}}| \geq 2$, and the function φ finally increments the values within $\{0, ..., K-1\}$, where $K \geq 2$, even if the initial value is less than 0. By [9] and the previous argument, this implies that the liveness property holds for Algorithm 3. □

Appendix 2. Proof of Theorem 1

We present a Lemma, and then we present the proof of the Theorem.

LEMMA 12. Let γ_0 be a configuration. For each execution $\gamma_0, \gamma_1, ...$ of Algorithm 2, in each of the configurations $\gamma_D, \gamma_{D+1}, ...$, for all $p, q \in P$, if $r_p, r_q \geq 0$, then $r_p = r_q$.

PROOF: If a process p executes RA in a step (γ_{i-1}, γ_i) $(1 \leq i \leq D)$, then $r^i_p = -D$. Since each command may add at most 1 to the value of r_p, and $D-i < D$, It follows that $r^D_p < 0$ holds. In step (γ_i, γ_{i+1}), each $q \in \mathcal{N}_p$ may execute only CA. By definition of CA, $r^{i+1}_q \leq r^{i+1}_p$ must hold. Inductively, for each process $g \in Neigh(p, D-i)$, $r^D_g \leq r^D_p < 0$ holds (*).

Let p_0 be a process, such that r^0_0 is the minimum among all other processes in P. Let $v = r^0_0$. By definition of the guarded commands, if p_0 executes either CA or NA, then $r^1_{p_0} = \varphi(v)$. If p_0 executes RA, then by (*), $r^D_0 < 0$. Since p_0 has the minimal value in γ_0, by definition of the guarded commands, for each $p_1 \in Neigh(p_0, 1)$, $r^1_1 = \varphi(v)$, or $r^D_1 < 0$ by (*). In the next configuration γ_2, for each process $p_2 \in Neigh(p_0, 2)$, by definition of the commands, either $r^2_2 = \varphi(\varphi(v))$ holds, or $r^D_2 < 0$ holds. Inductively, in configuration γ_D, for each process p, either $r^D_p = \varphi^D(v)$, or $r^D_p < 0$. This implies that if there exists a process p where $r^D_p \geq 0$, then r^D_p is equal to $\varphi^D(v)$, where v is the minimum value of r observed in γ_0. □

Proof of Theorem 1

Since Algorithm 1 is self-stabilizing wrt. SU, and since, by Lemma 6, ME holds if SU holds in any configuration, it follows that the liveness property of ME holds. Let γ_0 be a configuration. We show that for all $i \geq D-1$ steps, the safety property of ME holds in execution $\gamma_i, \gamma_{i+1}, ...$ We distinguish two cases: (1) $i \geq D$. By Lemma 12, for each execution $\gamma_0, \gamma_1, ...$, for each $p, q \in P$, if $r_p^D, r_q^D \geq 0$, it holds that $r_p^D = r_q^D$. By definition of *privileged*, processes p, q can be privileged in γ_D only if $r_p^D \geq 0$ holds. By the uniqueness of the processes' ids, the safety property of ME holds in configurations $\gamma_i, \gamma_{i+1}, ...$ (2) $i = D-1$. Assume by contradiction that there exist processes $p, q \in P$, where both are privileged in configuration γ_i. This implies that $r_p^i \neq r_q^i$, and $r_p^i, r_q^i \geq 0$. By definition of privileged, for all $p' \in \mathcal{N}_p$, $r_{p'}^i = r_p^i$. The same applies to q. By definition of the commands, p and q execute NA in step (γ_i, γ_{i+1}). By definition of NA, $r_p^{i+1} = \varphi(r_p^i)$ holds. The same applies to q. This implies that $r_p^{i+1} \neq r_q^{i+1}$, and $r_p^{i+1}, r_q^{i+1} \geq 0$. By construction, $i+1 = D$. This contradicts Lemma 12. □

Appendix 3. Proof of Lemma 10

Assume that $\forall p \in P \bullet r_p^0 = \epsilon$ holds in γ_0. By Def. 4, SU holds in γ_0, and by Lemma 2, SU holds in $\gamma_1, \gamma_2, ...$ Given that $r_p^0 \in stab_{\mathcal{X}}$, by definition of the guarded commands, only the command NA is executed by all processes in the following steps. This implies that for all $1 \leq i \leq n-1$, for all $p \in P$, $r_p^i = \varphi^i(r_p^0)$. By definition of φ, it follows that $r_p^i \in \{\epsilon, ..., n+\epsilon-1\}$, and $r_p^{n-1} = n+\epsilon-1$. By definition of $privileged_p$, in each configuration γ_i, there exists one process that is privileged, which implies that the number of configurations that satisfy $privileged_p$ in $\gamma_0, ..., \gamma_{n-1}$ is n. This implies that for each execution prefix $\gamma_0, ..., \gamma_i$, the recurrence of $priv$ is 1.0, which is greater than Δ.

Now by definition of NA and by the above analysis, $\varphi^{n+\epsilon-1}(r_p^0) = r_p^0$, and the recurrence of $privileged_p$ in $\gamma_0, ..., \gamma_{n+\epsilon-1}$ is greater or equal to Δ. Now the configuration $\gamma_{n+\epsilon-1}$ is equal to γ_0. This, by definition of NA and φ, implies that the following execution suffix repeats the cycle $\gamma_0, ..., \gamma_{n+\epsilon-1}$. Since this cycle always starts with n configurations satisfying $priv$, and is followed by only ϵ configurations, the recurrence of $priv$ is greater or equals Δ in any $\gamma_0, ..., \gamma_j$ for $j \geq 0$. This implies that $priv_{\Delta}$ holds in $\gamma_0, \gamma_1, ...$ □