Tight Space Self-stabilizing Uniform l-Mutual Exclusion

Maria Gradinariu Sébastien Tixeuil
Laboratoire de Recherche en Informatique, UMR CNRS 8623,
Université de Paris Sud, 91405 Orsay cedex, France
email: ({mariag,tixeuil}@lri.fr)

Abstract

A self-stabilizing algorithm, regardless of the initial system state, converges in finite time to a set of states that satisfy a legitimacy predicate without the need for explicit exception handler of backward recovery. The l-mutual exclusion is a generalization of the fundamental problem of mutual exclusion : the system has to guarantee the fair sharing of a resource that can be used by l processors simultaneously.

We present a space efficient solution to the l-mutual exclusion problem that performs on uniform unidirectional ring networks and that is self-stabilizing. Our solution improves the space complexity of previously known approaches by a factor of $\min(n^2 \times \log(n), \frac{1}{l} \times \log^{l-1}(n))$, while retaining none of their drawbacks in terms of system hypothesis (we support unfair scheduler and ensure strong correctness) or specification verification (we guarantee high level l-mutual exclusion). When l is fixed, the space complexity at each node is constant in average, making our approach suitable for scalable systems.

Extensive proofs can be found in [15].

1. Introduction

Self-stabilization. Robustness is one of the most important requirements of modern distributed systems since various types of (transient) faults are likely to occur as these systems are exposed to constant change of their environment. One of the most inclusive approaches to fault-tolerance in distributed systems is *self-stabilization* [10, 21]. Introduced by Dijkstra [10], this technique guarantees that, regardless of the initial state, the system will eventually converge to the intended behavior.

The scheduling hypothesis is crucial to the self-stabilization property of many distributed systems, because it permits to model the relative speed of the processors in the distributed system. In this paper, we distinguish between two main schedulers : the k-bounded scheduler (between any two actions of an enabled processor, every other processor executes at most k actions) and the unfair scheduler (between any two actions of an enabled processor, every other processor may execute an infinite number of actions).

l-Mutual Exclusion. The mutual exclusion problem — one of the most fundamental problems in distributed computing — consists in achieving access serialization to a common shared resource. A processor accessing this shared resource is *privileged*(or holds a privilege). This problem was generalized by Fisher, Lynch, Burns, and Borodin in [12] to the l-mutual exclusion problem. This extended problem can be viewed in two distinct ways : *(i)* there is a single shared resource that must not be used by more that l users at the same time or *(ii)* there are l shared resources and the system must achieve access serialization to each of them. The l-mutual exclusion problem was studied later by Shavit in [24]. Both [11] and [19] present algorithms for the l mutual exclusion problem. In [2], a practical application of l-mutual exclusion is presented. None of the aforementioned works is self-stabilizing.

More formally, a distributed system satisfies the l-mutual exclusion specification if and only if it satisfies each of the two following predicates : *(1)* **l-safety** : in any configuration, there are exactly l ($1 \leq l \leq n$, where n is the number of processors in the distributed system) privileges in the system, *(2)* **fairness** : in any computation, any processor is privileged infinitely often. This specification is referred as *low-level* l-mutual exclusion in the literature, and will be denoted by $\mathcal{SL}_{lME}$ in the following.

The fairness predicate can be generalized as **l-fairness** (formally, in any computation, and for any $x \in [1, l]$, each processor holds x privileges infinitely often). The specification involving l-safety and l-fairness is referred as *high-level* l-mutual exclusion, and will be denoted by $\mathcal{SH}_{lME}$ in the following.

Related Work. Self-stabilizing solutions to the l-mutual exclusion problem can be found in [13] (on unidirectional rings), [1] (on complete networks), [25] (on chain and bidirectional rings), [16] (on trees and bidirectional rings). In

particular, [13] is a generalization of Dijkstra's first mutual exclusion algorithm (see [10]). It should be noted that all these solutions only satisfy the low-level l-mutual exclusion specification ($\mathcal{SL}_{lME}$). In addition, [25] and [16] satisfy a less restrictive form of l-liveness (in any configuration, there are *at most* l privileged processors in the system).

It is well-known (see [3]) that there exists no deterministic algorithm which solves the mutual exclusion problem in uniform networks. In order to break this symmetry the solutions presented in [13, 16, 25] use a distinguished processor which performs an algorithm that is different from the other processors and [1] makes use of unique identifiers at each processor in the system.

In the context of unidirectional rings, the space complexity (in number of states per processor) of [13] is $O(l \times n^2)$ (where n is the size of the ring). A natural way to use this algorithm in uniform unidirectional rings is to compose it with a leader election algorithm for the same topology. [5] presented a space-optimal leader election algorithm for uniform unidirectional ring networks. This algorithm uses $O(snd^2(n))$ states per processor (where n denotes the size of the ring network, and $snd(n)$ denotes the smallest non divisor of n) under a k-bounded scheduler and $O(snd^3(n))$ states per processor under an unfair scheduler. Then the composition of the two algorithms would result in $O(l \times n^2 \times snd^2(n))$ and $O(l \times n^2 \times snd^3(n))$ states per processor, respectively.

In [9], randomization is used to break symmetry and a uniform solution to the l-mutual exclusion problem is presented in unidirectional rings. Basically, this solution is by executing simultaneously l instances of the space-optimal mutual exclusion algorithm presented in [4]. A drawback of this solution is that it performs correctly only under a k-bounded scheduler. This solution also suffers from its *weak correctness* property : there exists system computations (even though those computation have probability 0) such that at least one processor is never privileged.

Since [8] presents a uniform self-stabilizing mutual exclusion that behaves correctly under an unfair scheduler and that exhibits the *strong correctness* property, the same method as [9] on [4] could be applied on [8] to obtain a uniform self-stabilizing l-mutual exclusion algorithms on unidirectional rings that performs under the same hypothesis.

In these two cases, the space complexity of [9] is $O(snd^l(n))$ states per processor (where n denotes the size of the ring network, and $snd(n)$ denotes the smallest non divisor of n), and that of [9]+[8] is $O(n^l \times snd^{2\times l}(n))$.

Our contribution. In this paper, we concentrate on solving l-mutual exclusion on uniform unidirectional rings in a self-stabilizing way. We present two solutions that both guarantee high level fairness, perform under k-bounded and unfair scheduler respectively, satisfy weak and strong correctness respectively, and use $O(l \times T(n,l))$ and $O(l \times T(n,l) \times snd(n))$ states per processor respectively, where $T(n,l) = O(snd(n))$ for $l \geq 9$. Note that $snd(n)$ is constant in average and is $O(\log(n))$.

These results are summarized in the following table :

Algorithm	**Space Complexity**
[13]+[5] (k-bounded)	$O(l \times n^2 \times \log^2(n))$
[13]+[5] (unfair)	$O(l \times n^2 \times \log^3(n))$
[9] (k-bounded)	$O(\log^l(n))$
[9]+[8] (unfair)	$O(n^l \times \log^{2\times l}(n))$
This paper (k-bounded)	$O(l \times \log(n))$
This paper (unfair)	$O(l \times \log^2(n))$

Both for the k-bounded and unfair scheduler cases, our solution improves the space complexity of previously known approaches by a factor $n^2 \times \log(n)$ (in the case of [13]+[5]) and by a factor $\frac{1}{l} \times \log^{l-1}(n)$ (in the case of [9] or [9]+[8]). In addition, our unfair solution does not suffer from any of the aforementioned drawbacks in uniform self-stabilizing l-mutual exclusion algorithms.

Outline. The rest of the paper is organized as follows : Section 2 describes the computation model that will be used thereafter, Section 3 presents our algorithms and its correctness proof for the case of a k-bounded scheduler, while Section 4 handles the case of the unfair scheduler. Section 5 gives concluding remarks and hints at further developments.

2. Model

Distributed Systems. A distributed system is a collection of individual computing devices that can communicate with each other. We model a distributed system $S = (C, T, I)$ as a *transition system* where C is the set of system configurations, $T \subseteq C \times C$ is the set of system transitions, and I is the set of initial configurations. A *probabilistic distributed system* is a distributed system where a probabilistic distribution is defined over the system transitions.

We consider ring networks where the processors maintain two types of variables: *local variables* and *field variables*. The local variables of P_i cannot be accessed by any of its neighbors, whereas the field variables are part of the shared register which is used to communicate to P_i's right neighbor. A processor can write only into its own shared register and can read only from the shared registers, owned by its left neighbor or itself. The *state* of a processor is defined by the values of its local and field variables. A processor may change its state by executing its local *algorithm* (defined below). A *configuration* of a distributed system is an instance of the state of its processors.

The algorithm executed by each processor is described by a finite set of guarded actions of the form $\langle \text{guard} \rangle \longrightarrow \langle \text{statement} \rangle$. Each guard of processor P_i is a boolean expression involving P_i's variables and P_i's left neighbor's

field variables. A processor P_i is *enabled* in configuration c if at least one of the guards of the program of P_i is *true* in c. Let c be a configuration and CH be the set of enabled processors in c. We denote by $\{c : CH\}$ the set of configurations that are *reachable* from c if every processor in CH executes an action starting from c. A *computation step* is a tuple (c, CH, c'), where $c' \in \{c : CH\}$. Note that all configurations $\in \{c : CH\}$ are reached from c by executing *exactly one* computation step. In a probabilistic distributed system, every computation step is associated with a probabilistic value (the sum of the probabilities of the computational steps determined by $\{c : CH\}$ is 1). A *computation* of a distributed system is a maximal sequence of computation steps. A *history* is a finite prefix of a computation, while a *factor* is a finite sequence of a computation.

Let h be a history or a factor, then $length(h)$, $last(h)$ and $first(h)$ denote respectively the length of h, the last configuration in h, and the first configuration in h. Let e be a computation, $first(e)$ denotes its initial configuration . Let h be a history, let x be a factor such that $first(x) = last(h)$, and let e be a computation such that $first(e) = last(h)$. Then $[hx]$ denotes a history containing the sequence of computation steps in h followed by the sequence of computation steps in x, while (he) denotes a computation containing the sequence of computation steps in h followed by the sequence of steps in e.

Scheduler. A scheduler can be considered as an adversary. Intuitively the "luck" (i.e., the probabilistic part) of the algorithm and the scheduler play an infinite game. In this game, in any configuration c, during a computation e, the scheduler might use the history of the computation up to configuration c, and then chooses a non-empty subset of the enabled processors in c (according to some internal rules in the scheduler) to execute their enabled action.

A *choice function* for a history h returns a subset of enabled processors CH in $last(h)$. A scheduler can be defined by a collection of choice functions. A computation e *satisfies* a choice function f if and only if for any history h' of e such that $e = (h'(c, CH, c')\ldots)$, $f(h') = CH$.

A *strategy* st is a tuple $st = (c, f)$, where c is a configuration of $\mathcal{S}$ and f is a choice function of a scheduler. A computation e of $\mathcal{S}$ is called a st-*computation* where $st = (c, f)$ if and only if *(i)* $first(e) = c$ and *(ii)* if e satisfies the choice function f. In other words, an st-computation e is such that every choice of an enabled processor (by the scheduler) is the result of an application of the choice function of strategy st to the history (of e) at the time the choice was made.

Let st be a strategy. An st-*cone* $\mathcal{C}_h$ corresponding to a history h is the set of all possible st-computations which create the same history h (more details in [22]). The probabilistic value of an st-cone $\mathcal{C}_h$ is the probabilistic value of the history h (i.e., the product of the probability of every computation step in h). An st-cone $\mathcal{C}_{h'}$ is called a *sub-cone* of $\mathcal{C}_h$ if and only if $h' = [hx]$, where x is a computation fragment.

In [6], each strategy is the base for a probabilistic space in which any set of st-computations has an associated probabilistic value.

Probabilistic Self-Stabilizing Systems. A probabilistic self-stabilizing system is a probabilistic distributed system satisfying two important properties: *probabilistic convergence* (the system converges to a configuration satisfying a *legitimacy predicate*) and *correctness* (once the system is in a configuration satisfying a legitimacy predicate, it satisfies the system specification). The literature on self-stabilization discusses two variants of the probabilistic self-stabilizing systems: the systems with *weak correctness*—the system correctness is only probabilistic, and the systems with *strong correctness*—the system correctness is certain.

The problem with systems satisfying the weak correctness is that there is always at least one infinite, incorrect computation. Even though the theoretical probability to obtain an incorrect computation is zero, this is a weakness of the system for any actual applications.

Notation 1 *Let S be a system, D be a scheduler, and $st = (c, f)$ be a strategy such that c is a configuration of S and f is a choice function of D. We use $\diamond PRED_{st}$ to represent the set of st-computations that reach a configuration c' such that c' satisfies the predicate $PRED$ (denoted as $c' \vdash PRED$). The notation $Pr(\diamond PRED_{st})$ is used to express the probabilistic value associated with $\diamond PRED_{st}$.*

Definition 1 (Closed Predicate) *A predicate $PRED$ defined on the system configurations is* closed *if and only if the following condition is true: $PRED$ holds in configuration c implies that $PRED$ also hold in any configuration c' reachable from c.*

The weak self-stabilizing systems is defined below using a special predicate, the *legitimacy predicate*, defined on configurations. A computation e of S satisfying a predicate SP is denoted as $e \vdash SP$.

Definition 2 (Weak Probabilistic Stabilization) *A system S is* weak self-stabilizing *under D for a specification SP if and only if there exists a closed predicate L on configurations such that in any strategy $st = (c, f)$ of S under D the following two conditions are satisfied:*
(i) The probability of the set of st-computations, starting from c, reaching a configuration c', such that c' satisfies L (the legitimacy predicate), is 1 (probabilistic convergence); (Formally, $\forall st, Pr(\diamond L_{st}) = 1$) and
(ii) The probability of the set of computations, starting from a configuration c' such that c' satisfies L, and satisfying

SP, *is* 1 *(weak correctness). (Formally,* $\forall st, Pr(\{e \in st : e = (e'e''), last(e') \vdash L$ *and* $e'' \vdash SP\}) = 1$.*)*

The Weak probabilistic stabilizing systems only guarantee the *weak correctness* (once the system is in a configuration satisfying the legitimacy predicate, the system satisfies the specification with probability 1). A desirable property for the probabilistic distributed self-stabilizing systems would be a strong correctness. This would ensure that starting from a configuration satisfying the legitimacy predicate, the system unconditionally satisfies its specification. Such systems are called *strong stabilizing systems*.

Definition 3 (Strong Probabilistic Stabilization) *A system* S *is* strong self-stabilizing *under* D *for a specification* SP *if and only if there exists a closed predicate* L *on configurations such that in any strategy* $st = (c, f)$ *of* S *under* D*, the two following conditions hold:*
(i) The probabilistic convergence property is satisfied; (Formally, $\forall st, Pr(\diamond L_{st}) = 1$*)*
(ii) All computations, starting from a configuration c' *such that* c' *satisfies* L*, satisfy* SP *(strong correctness). (Formally,* $\forall st, \forall e \in st : e = (e'e'')$ *such that* $last(e') \vdash L$ *then* $e'' \vdash SP$*).*

Note that Definition 3 is stronger than Definition 2 (that was used in [17, 4])

Convergence of Probabilistic Stabilizing Systems Building on previous works on probabilistic automata (see [20, 22, 23, 26]), [6] presented a framework for proving self-stabilization of probabilistic distributed systems. In the following we recall the main results of [6], which are based on a key property of the system called *local convergence* and denoted by LC. This LC property is a progress statement that has positive probability as those presented in [7] (for the case of deterministic systems) and [22] (for the case of probabilistic systems).

An st-cone $\mathcal{C}_h$ satisfies the LC property, denoted as $LC(PR1, PR2, \delta, n)$, where $PR1$ and $PR2$ are two predicates defined on configurations and $PR1$ is a closed predicate, if the following two conditions hold: *(i)* $last(h)$ satisfies $PR1$; and *(ii)* The probability of all st-computations in $\mathcal{C}_h$ reaching a configuration c', such that c' satisfies $PR2$, in at most n computation steps, is at least δ. Informally, the LC property characterizes a probabilistic self-stabilizing system in the following way: The system reaches a configuration which satisfies a particular predicate, in a bounded number of computation steps, with a positive probability. We now formally capture the notion of LC below:

Definition 4 (Local Convergence) *Let* st *be a strategy,* $PR1$ *and* $PR2$ *be two predicates on configurations, where* $PR1$ *is a closed predicate. Let* $\mathcal{C}_h$ *be a* st*-cone with* $last(h) \vdash PR1$ *and let* M *denote the set of sub-cones* $\mathcal{C}_{h'}$ *of the cone* $\mathcal{C}_h$ *such that the following is true for every sub-cone* $\mathcal{C}_{h'}$*:* $last(h') \vdash PR2$ *and* $\exists N > 0$*,* $length(h') - length(h) \leq N$*. The cone* $\mathcal{C}_h$ *satisfies* LC $(PR1, PR2, \delta, N)$ *if and only if* $\exists \delta > 0$*,* $Pr(\bigcup_{\mathcal{C}_{h'} \in M} \mathcal{C}_{h'}) \geq \delta$.

Now, if in strategy st, there exist $\delta_{st} > 0$ and $N_{st} \geq 1$ such that any st-cone, $\mathcal{C}_h$ with $last(h) \vdash PR1$, satisfies $LC(PR1, PR2, \delta_{st}, N_{st})$, then the main theorem of the framework presented in [6] states that the probability of the set of st-computations reaching configurations satisfying $PR1 \wedge PR2$ is 1. Formally:

Theorem 1 ([6]) *Let* st *be a strategy. Let* $PR1$ *and* $PR2$ *be closed predicates on configurations such that* $Pr(\mathcal{EPR}1_{st}) = 1$*. If* $\exists \delta_{st} > 0$ *and* $\exists N_{st} \geq 1$ *such that any* st*-cone* $\mathcal{C}_h$ *with* $last(h) \vdash PR_1$*, satisfies the* LC $(PR1, PR2, \delta_{st}, N_{st})$ *property, then* $Pr(\diamond PR12) = 1$*, where* $PR12 = PR1 \wedge PR2$.

Remark 1 *If any strategy of a distributed system verifies Theorem 1 with* $PR1$ *being the true predicate (that is trivially verified in any system configuration) and* $PR2$ *being the legitimacy predicate then the system satisfies the* probabilistic convergence *as defined in Definition 3.*

Note 1 *Note that the previous result can easily be extended to histories. Details on this extension can be found in [14].*

3. Weak self-stabilizing l-mutual exclusion under k-bounded scheduler

In the following, we present a self-stabilizing algorithm that satisfy specification $\mathcal{SH}_{lME}$. In Algorithm 3.1, we provide a solution that performs under a k-bounded scheduler, and ensures weak correctness.

Informal Description of Algorithm 3.1 Algorithm 3.1 performs on n-sized unidirectional ring networks. It uses a generalization of the token management system presented in [18, 4], and ensures the fair circulation of exactly l ($l \geq 1$) tokens (or privileges) among system processors, provided that the system scheduler is k-bounded: between any two actions of an enabled processor, any other processor executes at most k actions. Randomization is used for two purposes: *(i)* to break the system symmetry and *(ii)* to ensure the high level fairness property.

In more details, a processor holding $x > 1$ tokens (*i.e.* a multi-privileged processor), randomly chooses the amount of tokens to be released but one. If the result of coin tossing is 0, then all tokens remain at the processor. Similarly, a processor holding a single token (*i.e.* a single-privileged processor) tosses a coin to decide between blocking and releasing its token.

Algorithm 3.1 Self-stabilizing l-Mutual exclusion under a k-bounded scheduler (for p)

Field: $v_p \in [0, l \times T(n,l) - 1]$ (*the privilege*)

Variables: $rand_p$ holds any value in {0, l-1}.

Functions:

$Gap(p) = (v_p - v_{p-1}) \bmod (l \times T(n,l))$

$$Nb\_privileges(p) = \begin{cases} 0 & \text{, if } Gap(p) = l \\ l & \text{, if } Gap(p) \neq l \\ & \text{ and } Gap(p) \bmod l = 0 \\ Gap(p) \bmod l & \text{, if } Gap(p) \bmod l \neq 0 \end{cases}$$

Macros:

$Pass\_ex\_privileges(p) \equiv v_p := (v_{p-1} - l + rand_p) \bmod (l \times T(n,l))$

$Pass\_privilege(p) \equiv v_p := (v_{p-1} + l) \bmod (l \times T(n,l))$

$Toss\_privileges(p) \equiv rand_p := random(0, \ldots, Nb\_tokens(p) - 1)$

Actions:

$\mathcal{A}_1$:: $Nb\_privileges(p) > 1 \longrightarrow$

$Toss\_privileges(p)$; if ($rand_p \neq 0$) then $Pass\_ex\_privileges(p)$;

$\mathcal{A}_2$:: $Nb\_privileges(p) = 1 \longrightarrow$

$Toss\_privileges(p)$; if ($rand_p$ = 1) then $Pass\_privilege(p)$;

Detailed Description of Algorithm 3.1 Every processor executing Algorithm 3.1 maintains a integer field variable. As stated in Section 2, this field variable can be read and wrote by its owner, and read by the right neighbor processor of its owner.

The difference between the field variable of a processor and the field variable of its left neighbor is called a *gap*. We assume that these field variables are bounded and that all operations on them are made modulo $l \times T(n,l)$, with

$$T(n,l) = \min_{i \in \{0,\ldots,l\} \setminus \{1\}} (T_i(n)) \quad (1)$$

where $T_i(n)$ is the smallest non divisor of n that also divides $(n-1+i)$. Note that when n is odd, $T(n,l) = 2$.

Depending on l, $T(n,l)$ and its gap value, a processor computes its number of privileges through a local function: $Nb\_privileges$. The result of this function indicates which rule should be executed by the processor: *(1)* A multi-privileged processor may execute rule $\mathcal{A}_1$ and randomly chooses the number of privileges to be hold (and implicitly the number of privileges that will be passed to its right neighbor). This multi-privileged processor has to keep at least one privilege. *(2)* A single-privileged processor may execute rule $\mathcal{A}_2$ and randomly choose to hold or pass its privilege.

3.1. Analysis of Algorithm 3.1

In this section we prove that Algorithm 3.1 is self-stabilizing for the $\mathcal{SH}_{lME}$ specification. The l-liveness property is guaranteed by the following properties:*(1)* there exists no deadlock configuration (*i.e.* a configuration where no rule is enabled at any processor), *(2)* the system holds at least l tokens at any time, *(3)* in any computation, the total number of privileges never increases.

Lemma 1 *In any configuration of Algorithm 3.1, the sum of gaps is 0 and there exists no configuration where all gaps equal l.*

Proof: Let *c* be an arbitrary configuration of Algorithm 3.1. Let $v_1, \ldots, v_n$ be the field variables of the processors in *c*. The sum of the gaps is determined by those variables: $(v_1 - v_n) + (v_2 - v_1) + (v_3 - v_2) + \ldots + (v_n - v_{n-1}) = 0$.

Let *e* be an arbitrary computation of Algorithm 3.1. Suppose that there is a configuration in *e* where all gaps equal l. The number of gaps is n, hence by Lemma 1 we obtain: $0 = l \times n (\bmod l \times T(n,l))$. This implies that $T(n,l)$ divides n, which is in contradiction with definition of $T(n,l)$ (see equation (1)). □

As an immediate corollary, there exists no configuration where the number of privileges is 0, and thus there exists no deadlock configuration.

Lemma 2 *There exists a configuration of Algorithm 3.1 with exactly l privileges.*

Proof: There exists a configuration *c* such that $(n-1)$ gaps equal l and one gap equals d (*i.e.* the number of privileges is deduced from *d*). Summing up the gaps in configuration *c* leads to: $(n-1) \times l + d = 0 (\bmod\ l \times T(n,l))$. The solutions of the previous equation are $d_i = i \times l, i \in \{0, \ldots l\} \setminus \{1\}$. By $Nb\_privileges$ definition, in any case the corresponding number of privileges is l. □

Lemma 3 *In any computation of Algorithm 3.1, there exists no configuration where there are less than l privileges.*

Proof: Let *e* be an arbitrary computation of Algorithm 3.1. Let us assume that there exists a configuration *c* in *e* where the sum of the privileges is less than l. If n_1 denotes the number of processors without privileges, there are $n - n_1$ processors holding privileges, yet the total number of privileges is less than l. Since n_1 processors have no privilege, there are n_1 gaps equal to l. By Lemma 1, we obtain:

$$n_1 \times l + g = 0 (\bmod l \times T(n,l)) \quad (2)$$

where g is the sum of the gaps at processors holding at least one privilege. By our assumption, the number of privileges is lower than l, hence g can be written as:

$$g = g_1 \times l + g_2 \quad (3)$$

where $g_2 < l$ and $g_2 \neq 0$. Using (3), equation (2) becomes:

$$(n_1 + g_1) \times l + g_2 = 0 (\bmod l \times T(n,l)) \quad (4)$$

Equation (4) implies that l divides g_2, which is impossible because $g_2 < l$ and $g_2 \neq 0$. By contradiction, configuration *c* with less than l privileges is impossible. □

Lemma 4 *In any computation of Algorithm 3.1, the number of privileges never increases.*

Corollary 1 *In any computation of Algorithm 3.1, any configuration reachable from a configuration with exactly l privileges has exactly l privileges.*

Definition 5 *A* legitimate configuration *is a configuration with exactly l privileges.*

Definition 6 *Let e be a computation of Algorithm 3.1. A* round *in e is the smallest factor of e in which every system processor held at least one privilege.*

3.1.1 Proof of Correctness

In the following, we prove that Algorithm 3.1 satisfies specification $\mathcal{SH}_{lME}$ with probability 1. In more details, we show that for any strategy, the set of computations reaching a legitimate configuration and satisfying $\mathcal{SH}_{lME}$ has probability 1. From previous definitions, it is sufficient to prove that for any strategy and any cone of computations $\mathcal{C}_h$, such that $last(h)$ is a legitimate configuration, $\mathcal{C}_h$ has a sub-cone $\mathcal{C}_{hh'}$ (where h' is a round) of positive probability.

Lemma 5 *Let st be an arbitrary strategy of Algorithm 3.1 under a k-bounded scheduler. Let $\mathcal{C}_h$ be a cone such that $last(h)$ is a legitimate configuration. There exists a positive probability ϵ and a positive number N such that $\mathcal{C}_h$ has a sub-cone $\mathcal{C}_{hh'}$ of probability at least ϵ, with h' a round and $length(h') < N$.*

Corollary 2 *Any strategy of Algorithm 3.1 under a k-bounded scheduler satisfies $\mathcal{SH}_{lME}$ with probability 1.*

3.1.2 Proof of Convergence

In order to prove the convergence of Algorithm 3.1 we prove that in any strategy of the algorithm under a k-bounded scheduler the local convergence property (see Definition 4) hold in any cone $\mathcal{C}_h$, where $last(h)$ is a non-legitimate configuration (it contains more than l privileges).

Lemma 6 *Let st be a strategy of Algorithm 3.1 under a k-bounded scheduler. Let $\mathcal{C}_h$ be a cone such that $last(h)$ is not a legitimate configuration. There exists a positive probability ϵ and a positive number N such that $\mathcal{C}_h$ has a sub-cone $\mathcal{C}_{hh'}$ of probability ϵ, and such that $last(h')$ is a legitimate configuration and $length(h') < N$.*

Proof: Let $\mathcal{C}_h$ be a cone such that $c = last(h)$ is not a legitimate configuration. Let $(p_i)_{i=1\ldots m}$ be the privileged processors (ordered clockwise), and let $(nbp_i)_{i=1\ldots m}$ be their respective number of privileges. We assume that d_i denotes the distance between the p_i and p_{i+1}. Since c is non-legitimate, the total number of privileges in c is $nbp > l$. From Lemma 4, nbp never increases. In the following we prove that nbp decreases with positive probability until it reaches l, by showing that there exist a positive probability that a processor p_m eventually holds all privileges. In such a configuration, by Lemmas 2 and 3, p_m holds exactly l privileges.

Assume that each time p_1 is chosen by the scheduler, it releases at most one token. Assume also that each time a processor different from p_1 executes its action, it does not release any privilege while p_1 still holds its privilege. Then, there is a cone $\mathcal{C}_{h_1}$ of positive probability such that in $last(h_1)$, all privileges held by p_1 in c were transfered to p_2. By using the same reasoning, there exists a cone $\mathcal{C}_{h_{m-1}}$ of positive probability such that in $last(h_{m-1})$, all privileges are blocked at p_m, and $nbp_m = l$. Let us denote by $\mathcal{C}_{hh'}$ the cone $\mathcal{C}_{h_{m-1}}$; this cone has probability:

$$\epsilon \geq \prod_{i=1}^{m} \left(\left(\frac{1}{\sum_{j=1}^{i} nbp_i (\mathrm{mod} l)} \right)^{(\sum_{i=1}^{m} nbp_i - 1)} \prod_{t=i+1}^{m} \left(\frac{1}{nbp_t} \right)^{k} \right)^{d_i}$$

and

$$lenght(h) \leq \sum_{i=1}^{m} \left(\left(\sum_{j=1}^{i} nbp_i (\mathrm{mod} l) \right) + k \times m \right) d_i$$

□

Corollary 3 *In any strategy the probability of the set of computations reaching legitimate configuration is 1.*

Proof: From Theorem 1 and Lemma 6 the corollary is proven. Note that the hypothesis of Theorem 1 holds since our bound was shown for the worst possible scenario. □

Theorem 2 *Algorithm 3.1 self-stabilizes to $\mathcal{SH}_{lME}$ under a k-bounded scheduler.*

3.2. Space Complexity

The two following lemmas prove that $snd(n)$ is $O(\log(n))$ and is constant in average.

Lemma 7 $snd(n) = O(\log(n))$.

Proof: When n is an odd number, $snd(n) = 2$. Let us prove that when $n = t \times k!$, $snd(n) = O(\log(n))$. We use the Stirling formula which gives an approximation for $k!$:

$$k! = \sqrt{2 \times \pi \times k} \left(\frac{k}{e} \right)^{k} \left(1 + \frac{1}{12 \times k} + O\left(\frac{1}{k^2} \right) \right) \quad (5)$$

By Equation 5, we obtain $n \geq t \times 2^k$ when $k \geq 2 \times e$. This implies $snd(n) = O(\log(n))$. □

Lemma 8 *$snd(n)$ is constant in average.*

We now prove that for any $l \geq 9$, $T(n, l) = O(snd(l))$, in the worst case. We then prove that $T(n, l)$ is constant in average.

Lemma 9 *Assume $l \geq 9$. There exists c, ($1 \leq c < 7$) such that for any $n \geq 2$, $T(n,l) \leq (snd(n) + c)$.*

Proof: We study the values of $T(n,l)$ for $n \geq 2$. For any odd number x, we have $T(n,l) = 2$ and $snd(n) = 2$, hence in this case, and for all $c < 7$, we have $T(n,l) \leq snd(n)+c$.

We now study the case when $n = f \times k!$, which implies $snd(n) = O(k)$. We prove that there exists $c \geq 1$ such that $T(n,l) = k + c$ and c does not depend on k.

Number n can be written as $n = t \times 10^m + x$, where $x \in \{0, \ldots, 9\}$. We describe using similar formalism numbers $n + i - 1$, with $i \in \{0, \ldots, l\} \setminus \{1\}$, which are used in the calculus of $T(n,l)$:

$$n+i-1 = t \times 10^m + x + i - 1 \ \ (i \in \{0, \ldots l\} \setminus \{1\}) \quad (6)$$

Suppose that there exists no $c < 7$ such that $T(n,l) = k+c$. This means that for any $c < 7$ and for any $i \in \{0, \ldots, l\} \setminus \{1\}$, $k + c$ is not a divisor of $(n + i - 1)$. Formally, this assertion can be written as:

$$t \times 10^m + x - 1 = (k+c) \times q_1 + x_1 \, (x_1 \in \{1, \ldots, 9\}) \quad (7)$$

when $i = 0$ and as:

$$\begin{array}{l} t \times 10^m + x + j = (k + c) \times q_j + x_j \\ (j \in \{1, \ldots, l-1\}, x_j \in \{1, \ldots, 9\}) \end{array} \quad (8)$$

when $2 \leq i \leq l$. We now substract (7) from (8) and obtain:

$$\begin{array}{l} j + 1 = (k + c) \times (q_j - q_1) + (x_j - x_1) \\ (j \in \{1, \ldots, l-1\}, x_j \in \{1, \ldots, 9\}) \end{array} \quad (9)$$

Two cases are to be considered to compute the value of c: *(a.)* $q_j - q_1 = 0$ and then Equation (9) becomes: $j + 1 = x_j - x_1$ ($j \in \{1, \ldots, l-1\}, x_j \in \{1, \ldots, 9\}$). *(b.)* $q_j - q_1 \neq 0$ and then Equation (9) becomes: $c = \frac{(j+1)-(x_j-x_1)}{q_j-q_1} - k = (j + 1) - (x_j - x_1) - k$ (note that by definition of q_j and q_1, if $q_j - q_1 \neq 0$, then $q_j - q_1 = 1$)

In order to prove that our assumption (there exists no $c < 7$ such that $T(n,l) = k + c$) is false, it is sufficient to show that in case *(a.)* there exists j such that for any c, the equation is false; and that in case *(b.)*, there exists j and c such that the equation is false.

In the first case, it is sufficient to take $j = l - 1$. In the second case, we assume l has its minimal hypothesis value, *i.e.* $l = 9$. Exhaustive calculus proves that for any $k \geq 2$, there exists $c < 7$ and $j \leq l - 1$ such that Equation (9) is false. By minimizing terms in the equation, we obtain 7 as a minimal value of k. □

Lemma 10 *Assume $l \geq 9$. $T(n,l) = O(\log(n))$.*

Lemma 11 *Assume $l \geq 9$. $T(n,l)$ is constant in average.*

Lemma 12 *Each processor executing Algorithm 3.1 requires $O(l \times \log(n))$ states, where n is the size of the ring. On average the number of states required by each processor is $O(l)$.*

Proof: Each processor p maintains one field variable v_p, that ranges from 0 to $l \times T(n,l) - 1$. Since its local variable $rand_p$ that ranges from 0 to $l-1$ is used for clarity only and is not conserved between any two rule executions, we do not consider it for space complexity. By Lemma 10, overall each processor needs $O(l \times \log(n))$ states.

From Lemma 8, $T(n,l)$ is constant on average, and each processor variable uses $l \times T(n,l)$ states, hence the number of states required by each processor is $O(l)$. □

4. Strong self-stabilizing l-Mutual exclusion under unfair scheduler

In this section, we overview how to transform Algorithm 3.1 into a strong self-stabilizing algorithms that copes with unfair scheduling. The method is a variation of that presented in [8] that improves memory overhead.

Overview The transformed algorithm consists in two layers: *(i)* the *l-privileges layer* has similar behavior as Algorithm 3.1 but guarantee strong correctness, and *(ii)* the *fair tokens layer*, which permits to cope with unfair scheduling.

The l-privileges layer is obtained from Algorithm 3.1 as follows: each privilege is added a speed that indicates how many processors it should traverse before being stopped. Each processor on the path of a non-zero speed privilege decreases its speed by one. Eventually the privilege speed reaches 0, and the currently holding processor sets a new randomly choosen speed for the privilege.

The fair tokens layer is described in [6] and guarantees that even when the system is running under an unfair scheduler, every processor executes infinitely many actions, and that between any two actions of a given processor any other processor executes a bounded number of its own actions. As such, the fair tokens layer transforms the unfair scheduler into a k-bounded scheduler.

Then, a processor is privileged (in the sense of the l-mutual exclusion algorithm) if it holds both a privilege (from the l-privileges layer) and a fair token (from the fair tokens layer).

The final protocol is obtained by the combination of the two layers using a special technique of composition — the cross-over composition — defined in [6]. Detailed explanations and extensive proofs of the resulting algorithm can be found in [15].

Memory requirements The fair tokens layer of [6] requires $snd(n)$— smallest non divisor of n — states. This

bound was proven optimal in [4] for unidirectionnal rings. The l-privileges layer requires only two different speeds to guarantee strong correctness, so it doubles the number of states at each processor. Overall, the memory requirement for the strong stabilizing protocol under unfair scheduler is $O(l \times \log^2(n))$.

5. Concluding Remarks

We presented two probabilistic self-stabilizing algorithms for l-mutual exclusion in unidirectional rings of uniform processors. While the stabilization time of both algorithms may be quite long, they do have nice properties with concern to scheduling assumtions and space requirements. Algorithm 3.1 supports a k-bounded scheduler (between any two actions of a processor, every other processor executes at most k actions), guarantees weak correctness (the *probability* of a processor getting at least one privilege once the system is stabilized is 1) and requires $O(l \times \log(n))$ states per processors. In Section 4 is presented a transformation of Algorithm 3.1 that ensures strong stabilization in spite of unfair scheduling. Overall, the space improvement over previously known approaches is a $\min(n^2 \times \log(n), \frac{1}{l} \times \log^{l-1}(n))$ factor in the worst case. When l is considered constant, then the average number of states (over all possible ring sizes) is also constant for the two algorithms. There remains the open question of state optimality for self-stabilizing uniform l-mutual exclusion algorithms.

References

[1] U. Abraham, S. Dolev, T. Herman, and I. Koll. Self-stabilizing l-exclusion. *In proceedings of third workshop on self-stabilizing systems*, pages 48–63, 1997.

[2] Y. Afek, D. Dolev, E. Gafni, M. Merritt, and N. Shavit. A bounded first-in, first-enabled solution for the l-mutual exclusion problem. *4th International Workshop on Distributed Algorithms, bari, Italy*, LNCS 486:422–431, 1990.

[3] D. Angluin. Local and global properties in networks of processors. In *Proc. Symp. on Theory of Computing(STOC'80)*, pages 82–93, 1980.

[4] J. Beauquier, S. Cordier, and S. Delaët. Optimum probabilistic self-stabilization on uniform rings. In *Proceedings of the Second Workshop on Self-Stabilizing Systems*, pages 15.1–15.15, 1995.

[5] J. Beauquier, M. Gradinariu, and C. Johnen. Memory space requirements for self-stabilizing leader election protocols. In *PODC99*, pages 199–208, 1999.

[6] J. Beauquier, M. Gradinariu, and C. Johnen. Randomized self-stabilizing optimal leader election under arbitrary scheduler on rings. Technical Report 1225, Laboratoire de Recherche en Informatique, September 1999.

[7] K. Chandy and J. Misra. *Parallel Programs Design: A Foundation.* Addison-Wesley, New York, N.Y., 1988.

[8] A. K. Datta, M. Gradinariu, and S. Tixeuil. Self-stabilizing mutual exclusion using unfair distributed scheduler. In *IPDPS'2000*, pages 465–470, 2000.

[9] S. Delaët. *Auto-stabilisation: Modèle et applications à l'exclusion mutuelle*. PhD thesis, LRI, Université de Paris Sud, 1995.

[10] E. Dijkstra. Self stabilizing systems in spite of distributed control. *Communications of the Association of the Computing Machinery*, 17:643–644, 1974.

[11] D. Dolev, E. Gafni, and N. Shavit. Toward a non-atomic era: l-exclusion as test case. *20 Annual ACM Symposium on the Theory of Computing, Chicago, IL, May 2-4*, pages 78–92, 1988.

[12] M. Fischer, N. Lynch, J.E.Burns, and A. Borodin. Resource allocation with immunity to limited process failure. In *20th Annual Symposium on Foundations of Computer Science*, pages 234–254, 1979.

[13] M. Flabeto, A. Datta, and A.A.Schnoone. Self-stabilizing multi-token rings. *Distributed Computing*, 8:133–142, 1994.

[14] M. Gradinariu. *Modelisation, verification et raffinement des algorithmes auto-stabilisants*. PhD thesis, LRI, Université de Paris Sud, 2000.

[15] M. Gradinariu and S. Tixeuil. Tight space self-stabilizing uniform l-mutual exclusion. Technical Report 1249, Laboratoire de Recherche en Informatique, march 2000.

[16] R. Hadid. Space and time efficient self-stabilizing l-exclusion in tree networks. In *IPDPS'2000*, 2000.

[17] T. Herman. Probabilistic self-stabilization. *Information Processing Letters*, 35:63–67, 1990.

[18] A. Israeli and M. Jalfon. Token management schemes and random walks yield self-stabilizing mutual exclusion. In *PODC90 Proceedings of the Ninth Annual ACM Symposium on Principles of Distributed Computing*, pages 119–131, 1990.

[19] G. Peterson. Observation on l-exclusion. *28th Annual Allerton Conference on Communication, Control and Computing, Monticello, IL, October 3-5*, pages 568–577, 1990.

[20] A. Pogosyants, R. Segala, and N. Lynch. Verification of the randomized consensus algorithm of aspen and herlihy: a case study. In *Distributed Computing(13)*, pages 155–186, 2000.

[21] M. Schneider. Self-stabilization. *ACM Computing Surveys*, 25:45–67, 1993.

[22] R. Segala. *Modeling and Verification of Randomized Distributed Real-Time Systems*. PhD thesis, MIT, Departament of Electrical Engineering and Computer Science, 1995.

[23] R. Segala and N. Lynch. Probabilistic simulations for probabilistic processes. In Springer-Verlag, editor, *CONCUR '94, Concurrency Theory, 5th International Conference , LNCS:836*, Uppsala, Sweden, August 1994.

[24] N. Shavit. *Concurrent time stamping*. PhD thesis, Hebrew University, Departament of Computer Science, Israel, 1990.

[25] V. Villain. A key tool for optimality in the state model. In C. U. Press, editor, *Proceedings of the Second Workshop on Distributed Data and Structure (DIMACS'99)*, 1999.

[26] S. H. Wu, S. A. Smolka, and E. W. Stark. Composition and behaviors of probabilistic i/o automata. In *concur94*, pages 513–528, 994.