# Randomized Self-stabilizing and Space Optimal Leader Election under Arbitrary Scheduler on Rings*

Joffroy Beauquier, Maria Gradinariu, Colette Johnen

L.R.I./C.N.R.S., Université de Paris-Sud,
bat 490, 91405 Orsay Cedex, France
jb,colette,mariag@lri.fr

**Abstract**

We present a randomized self-stabilizing leader election protocol and a randomized self-stabilizing token circulation protocol under an arbitrary scheduler on anonymous and unidirectional rings of any size. These protocols are space optimal. We also give a formal and complete proof of these protocols.
Therefore, we develop a complete model for probabilistic self-stabilizing distributed systems which clearly separates the non deterministic behavior of the scheduler from the randomized behavior of the protocol. This framework includes all the necessary tools for proving the self-stabilization of a randomized distributed system: definition of a probabilistic space and definition of the self-stabilization of a randomized protocol.
We also propose a new technique of scheduler management through a self-stabilizing protocols composition (*cross-over composition*). Roughly speaking, we force all computations to have some fairness property under any scheduler, even under an unfair one.

**keywords:** self-stabilization, randomized protocol, protocol composition, scheduler, leader election.

## 1 Introduction

Self-stabilization is a framework for dealing with channel or memory failures. After a failure the system is allowed to temporarily exhibit an incorrect behavior, but after a period of time as short as possible, it must behave correctly, without external intervention. For distributed systems, the self-stabilization feature can be viewed as an element of transparency with respect to failures, because the user is not supposed to notice a major change in the quality of service he or she receives, or at least not during a long time.

Another type of transparency for distributed systems is the transparency to the dynamic evolution of the network. It is mandatory that the connection of a new sub-network to the main one does not

*contact author: Colette Johnen, LRI, Université de Paris-Sud, centre d'Orsay - bat 490, 91405 Orsay Cedex, France. fax: +33 1 69 25 65 86, e-mail: colette@lri.fr, web: www.lri.fr/∼colette/

interact with the configuration of a particular user. This type of transparency is not satisfied when the system - i.e. the protocols constituting it - have to be scaled to the size of the network. That is the reason why the study of constant space protocols has received a lot of attention in the past recent years. A protocol uses only constant space if the memory space needed by each processor is constant per link. Then in the case where the network is extended (or reduced) the majority of processors have not to change neither hardware nor software.

In this paper we address a basic task for distributed systems, leader election, having in mind to obtain solutions both self-stabilizing and using constant space memory. When a system is in a symmetrical configuration, no deterministic protocol can break symmetry and elect a leader [2]. This impossibility results applies to self-stabilizing system, then randomization is needed. Some results are known concerning self-stabilizing randomized leader election. In [3] a self-stabilizing leader election protocol on bidirectional id-based networks presented, requiring $\lg^*(N)$ states per process ($N$ being the network size). A basic protocol is given, requiring $N$ states per process, and the result is obtained by using a data structure that stores distributively the variables. In an appendix, [15] uses another data structure based on the Thue-Morse sequence, requiring $O(1)$ bits per edge to store in a distributed manner variables having possibly $N$ values. These two last results require bidirectional networks. When the deadlock freedom property is guaranteed externally, a randomized self-stabilizing, constant space leader election protocol is given in [17] in the message passing model. [4] presents a randomized token circulation protocol on unidirectional rings that stabilizes with some type of scheduler. Its space complexity is $m_N$ states per process, where $m_N$ is the smallest integer that does not divide $N$. In [11], the previous protocol is extended in order to manage any anonymous bidirectional networks. This new protocol requires the same memory space. In [16], the first token circulation protocol that self-stabilizes under unfair schedule is presented. At the end, a randomized compiler that transforms a self-stabilizing protocol on bidirectional, anonymous rings into a a protocol on synchronous unidirectional anonymous rings is presented in [18]; it uses constant space.

In the present paper, we present a space optimal randomized self-stabilizing leader election protocol for anonymous and unidirectional rings of any size, under any scheduler (in particular no fairness assumption is required). Its space complexity is $O(m_N)$. It should be noticed that $m_N$ is constant on average. On odd size rings, 4 bits are sufficient for leader election. The optimality of our protocols is proven in [5].

Proving formally the protocol needs a proper model for randomized self-stabilizing protocols. An important issue is that the model must make a clear distinction between what is non-deterministic (the scheduler/adversary) and what is randomized (the protocol). Some papers consider that the scheduler, when choosing a process to be activated in a given configuration, obeys some probabilistic rule. Although it could be argued that such an hypothesis corresponds generally to the reality, we wont here adopt this approach, mainly because a probabilistic scheduler is a very feeble adversary. Then, in order to obtain the strongest result, we consider the strongest adversary, which is non-deterministic. The deep difference between a probabilistic and a non-deterministic scheduler can be pointed out by a very simple example. Consider a unidirectional ring with two tokens (no matter how tokens are represented). If a process in the ring holds a token and is chosen by the scheduler, with probability 1/2 it passes the token to its successor and with probability 1/2 it keeps the token. It appears (and will be easily provable after we have developed our model) that if the scheduler is non-deterministic, it can avoid forever (with certainty) one of the tokens to catch up with the

other, while if the scheduler is probabilistic (each process holding a token is chosen in a step with probability 1/2), the two tokens catch up with probability 1. Another important issue for the model is to make clear the end part of the previous sentence: What means that the two tokens catch up with probability 1 ? The first step is to relate the probabilistic laws of the random variables that appears in the rules of the randomized protocol, to a probability measure defined on the set of (finite and infinite computations) of this protocol. The second step consists in proving that, for this probability measure and for any "behavior" of the scheduler (obviously what a "behavior" of the scheduler is, must be defined), the set of computations in which one token catches up with the other has probability 1 (or equivalently that the (non empty) set of computations in which the tokens never catch up has probability 0).

At the end it should be mentioned that for the sake of clarity, we have decomposed the leader election protocol into two sub-protocols related by a new composition, that we named cross-over composition. An interesting point with cross-over composition is that, when protocol W is composed with protocol S, the composite has the same properties in term of fairness than the protocol S. In other words, if for some particular reasons, the computations of S under an unfair scheduler are in fact fair, then the computations of the composite under this same unfair scheduler, are also fair. We show how this property of the cross-over composition yields an automatic technique for transforming a protocol, designed and proved for a fair scheduler, into an equivalent protocol for an unfair scheduler, making simpler the task of the designer/prover.

### 1.1 Others related works

Probabilistic I/O automata were presented in [23] and [7]. This work was improved by Wu, Smolka and Stark [25]. In [22], Lynch and Segala introduced a method including the adversary in the probabilistic automaton which models a distributed system. They made a distinction between the protocol which is probabilistic and the adversary which is non deterministic. In [21] Segala considers the model from [25] with the scheduler defined in [22]. In [19], the authors apply the model to verify formally the time properties of randomized distributed protocols. The case study is the randomized dining philosophers protocol of Lehman and Rabin. The main restriction of this model is that it manages only schedulers that have a probabilistic behavior as explained in [19]. Even, if most of the schedulers have a probabilistic behaviors, this model does not always a complete analysis of a randomized protocol. In particular, it does allow to analyze a randomized protocol under the worst conditions: an unfair scheduler.

In the self-stabilization area, the first randomized protocols were proposed by Israeli and Jalfon in [14] and by Anagnostou and El-Yaniv in [1]. The notion of self-stabilization for a randomized protocol is defined without presenting any probabilistic space. Therefore, no formal proof is given.

In [10] Dolev, Moran and Israeli introduced the idea of a two players games (scheduler-luck game) to analyze the performance of randomized self-stabilizing protocol under Read/Write atomicity. The scheduler is an adversary of the protocol that wants to keep the protocol away from legitimate configurations. Sometimes, the luck intervenes to influence the output of the binary randomized variables. Their framework is not designed to establish the self-stabilization of the protocol: in fact they assume that the protocol is self-stabilizing. In contrast, our framework is designed to prove the self-stabilization of a protocol that uses any type of random variables (binary or not).

Hierarchical composition was presented in [12], and in [9]: the $(k+1)$th component stabilizes to the desired behavior after stabilization of first $k$ components. In [12], a selective composition is presented. Composition of independent components interacting with each others was presented in [24]. In [8], the parallel composition is presented, designed to improve the convergence time. Several protocols independently perform the same task, in parallel; a specific protocol selects one output as the composition output (it chooses the output of the fastest protocol).

The paper is organized as follows. The formal model of a distributed system is given in section 2. In section 3, we define the framework used to prove formally randomized self-stabilizing protocols. The cross-over composition is defined in section 4. Space optimal, randomized self-stabilizing token circulation and leader election protocols are presented and proven in section 5. Finally, the paper ends with some concluding remarks.

# 2 Model

First, we give an abstract model of a distributed system; then, we present an interpretation in terms of real systems.

## 2.1 Abstract model

A non deterministic distributed systems is represented in the abstract model of **transition systems**.

**Definition 2.1** *A* **distributed system** *is a tuple* $DS = (\mathcal{C}, T, \mathcal{I})$ *where*

- $C$ *is the set of all system configurations;*
- $T$ *is a transition function of* $\mathcal{C}$ *to the set of* $\mathcal{C}$ *subset;*
- $\mathcal{I}$ *is a subset of the configuration set called the initial configurations.*

In a randomized distributed system, there is a probabilistic law on the output of a transition.

**Definition 2.2** *A* **computation** $e$ *of a distributed system is a sequence of configurations:* $e = [(c_0, c_1), (c_1, c_2) \ldots]$ *where (1)* $c_0 \in \mathcal{I}$*, and (2)* $\forall i :: c_{i+1}$ *is an output of a transition starting to* $c_i$*. The configuration* $c_0$ *is the initial configuration of the computation* $e$*.*

**Notation 2.1** *Let* $c$ *be a initial configuration of a distributed system. The* $c$ **tree** *is the tree composed of all maximal computations whose initial configuration is* $c$*. The computation* **forest** *of a distributed system* $(\mathcal{C}, T, \mathcal{I})$ *is the set of all* $c$ *trees where* $c \in \mathcal{I}$*.*

## 2.2 Interpretation

The abstract model defined above is a mathematical representation of the reality. In fact, the distributed system is the collection of processors ($Proc$) computing protocol $\mathcal{P}$. A protocol has a collection of variables (internal and/or field) and has a code part.

A processor communicates only with its neighbors (a subset of $Proc$). Communication among neighbors is carried out by field variables. A processor can read the field variables of its neighbors; but it cannot read their internal variables. A processor can read and update its own variables.

**Local and global configuration versus state and configuration.** The state of a processor is the collection of values of the processor's variables (internal or field). A configuration of a distributed system is a vector of processor states. A *local configuration* is the part of a configuration that can be "seen" by a processor (i.e. its state and the field variables of its neighbors).

**Actions.** The code is a finite set of guarded actions (i.e. label:: guard $\rightarrow$ statement). The guard of an action on $p$ is a boolean expression involving $p$ local configuration. The statement of a $P$ action updates the $p$ state. If the action is randomized, several statements are possible, and each of them has a probability.

We assume that no processor of a distributed system satisfies the guards of two actions in the same configuration.

A processor $p$ is **enabled** at a configuration $c$, if an action guard of $p$ is satisfied in $c$. The set of enabled processors for $c$ is denoted by $Enabled(c)$.

**Computation step versus transition.** Let $c$ be a configuration, and $CH$ be a subset of enabled processors at $c$. We denote by $< c : CH >$ the set of configurations that are reachable from $c$ after that the processors of $CH$ have performed an action. A computation step has three elements: (1) an initial configuration: $c$, (2) a set of enabled processors: $CH$, and (3) a configuration of $< c : CH >$.

Clearly, the transitions in the abstract model can be interpreted in terms of computation steps.

In the case of a deterministic protocol, a computation step is totally defined by the initial configurations and the set of enabled processors: there is only one final corresponding configuration. But in the case of randomized protocol, the final configuration depends on the output of each processor action. The output of a processor action depends on the value of the processor's random variable. Therefore, in the case of randomized protocols, the computation step has a fourth characteristic element: the probabilistic value associated to the computation step. This value depends on the probabilistic law of the random variable of each processor involved in the computation step.

A computation is **maximal**, if the computation is either infinite, or finite and no processor is enabled in the final configuration. In this case, the configuration is said to be terminal. The set of all maximal computations of the distributed protocol, $\mathcal{P}$, is denoted by $\mathcal{E}_{\mathcal{P}}$. The set of all computations of $\mathcal{P}$ that are not maximal is denoted by $\mathcal{PAR}\_\mathcal{E}_{\mathcal{P}}$; these computations are called **unfinished**. If $e$ is not infinite, we denoted by **last**(e) the final configuration of $e$.

## 2.3 Scheduler

At each step of a computation, a scheduler selects a (non-empty) subset of enabled processors which will be activated during the next computation step. There are two ways to view a scheduler: first as a result, subset of all possible computations of the distributed system; secondly as a dynamic selecting process called at each step of a computation.

**Definition 2.3** *In a distributed system, a* **scheduler** *is a predicate over maximal computations.*

From the dynamic point of view, a scheduler chooses a subset of enabled processors. These processors will be the only ones to perform an action during the next computation step. The choice of the scheduler is done according to the reached configuration but also according to the computation past (the computation steps that have been performed). A scheduler is completely defined if one knows for every sequence of computation steps, all possible scheduler choices (several subsets of enabled processors).

**Notation 2.2** *$Proc_s$ is the set of subsets of $Proc$.*

**Definition 2.4** *Let $\mathcal{P}$ be a protocol. A function of* **Choice** *is a function from $\mathcal{PAR\_E}_{\mathcal{P}}$ to $Proc_s$ such that $f(e) \subset Enabled(last(e))$ and $f(e) \neq \emptyset$.*

*Let $f$ be function of* Choice*. Let $e$ be a maximal computation. If for any prefix of $e$ defined as $[e1, (c, CH, c')]$, we have $CH = f(e1)$, we said that $e$ adheres to $f$.*

*$f$ adheres to $D$ if and only if all computations that adhere to $f$ satisfy the predicate $D$.*

*The dynamic aspect of a scheduler $D$ on $\mathcal{P}$ is the collection of* Choice *functions that adhere to $D$. We call $Choice_{PD}$ this set of* Choice *functions.*

Now, we define the computation forest of a distributed system under a scheduler.

**Definition 2.5** *Let $DS$ be a distributed system. Let $T$ be a tree of $DS$. The $T$ tree under $D$ is the subtree of $T$ that contains only the computations verifying $D$. The computation* **forest** *of $DS$ under $D$ is the set of all $c$ trees of $DS$ under $D$ where $c \in \mathcal{I}$.*

## 2.4 Strategy

One can imagine schedulers having no regular behavior. For instance, a scheduler that is well-disposed to the processor $p$: as soon as $p$ is enabled then it chooses $p$. Or a scheduler that avoids to select $q$ - it chooses $q$ only if $q$ is the only enabled processor -. To analyse a protocol, the scheduler has to be considered as an adversary: the protocol must work properly in spite of the "bad behavior" of the scheduler. The interaction between the scheduler and the protocol can be seen as a game. In this game, the goal of the scheduler is to prevent the protocol doing its job.

In this game the scheduler has different strategies to win according to some "rules". The figure 1 presents a strategy. Initially, the scheduler chooses some enabled processors in the initial configuration. This first step in the scheduler strategy gives all the computation steps that can occur according to the scheduler choice (in a randomized protocol, the choice of the scheduler determines several computation steps). For each obtained computation step, the scheduler makes one choice. This second step in the scheduler strategy gives all the sequences of 2 computation steps that can be obtained after the two first choices of the scheduler. For each of these sequences, the scheduler makes one and only one choice, if it can (if the configuration is not terminal), and so on.

**Definition 2.6** *Let $\mathcal{P}$ be a protocol under a scheduler $D$. A* **strategy** *$st$ is defined by the tuple $st = (c, f)$ where $c$ is a configuration, and $f$ is a function of $Choice_{PD}$.*

*Let $st = (c, f)$ be a strategy. A maximal computation belongs to $st$ if and only if (1) $c$ is the initial configuration of $e$; and (2) if $e = [e1(c, CH, c')e2]$ then $CH = f(e1)$.*

**Observation 2.1** *Let $st$ be a strategy of the protocol $P$ under the scheduler $D$. Let $e1$ be a prefix of a computation of $st$ such that $e1(c, CH, c')$ is also a prefix of a computation of $st$. If $c'' \in < c : CH >$ then $e1(c, CH, c'')$ is also a prefix of a computation of $st$.*

**Observation 2.2** *In the case of a deterministic protocol, a strategy is a maximal computation.*

**Observation 2.3** *Let $P$ be a protocol under the scheduler $D$. Each couple $(c, f)$ where $c \in \mathcal{C}$ and $f \in Choice_{PD}$ defines a strategy of $P$ under $D$. Thus the number of strategies of $P$ under $D$ is $|\mathcal{C}|.|Choice_{PD}|$ which is usually infinite.*

In the case of a randomized protocol, we will define a probabilistic space for every strategy of a scheduler $D$.

The abstract side of the strategy is given by the following definition:

**Definition 2.7** *Let $DS$ be a distributed system. Let $T$ be a tree of $DS$ under the scheduler $D$. A strategy is a subtree of $T$ where at a node, there is only one transition outgoing.*

## 2.5 Self-stabilizing distributed systems.

A self-stabilizing distributed system is a particular case of distributed systems where any configuration is an initial configuration.

**Notation 2.3** *Let $\mathcal{X}$ be a set. $x \vdash Pred$ means that the element $x$ of $\mathcal{X}$ satisfies the predicate $Pred$ defined on the set $\mathcal{X}$.*

**Definition 2.8** *A protocol $\mathcal{P}$ is self-stabilizing for a specification $SP$ (predicate over the computations) if and only if there exists a predicate $L$ defined on configurations such that:*

- **convergence** *All computations reach a configuration that satisfies $L$. Formally, $\forall e \in \mathcal{E}_{\mathcal{P}} : e = [(c_0, CH_0, c_1)(c_1, CH_1, c_2)\ldots] :: \exists n \geq 1, c_n \vdash L$.*
- **correctness** *All computations starting in configurations satisfying the predicate $L$ are satisfying the problem specification $SP$. Formally, $\forall e \in \mathcal{E}_{\mathcal{P}} : e = [(c_0, CH_0, c_1)(c_1, CH_1, c_2)\ldots] :: c_0 \vdash L \Rightarrow e \vdash SP$.*

*The predicate $L$ is called the legitimate predicate.*

The attractor technique defined in [13] is a very useful technique to prove the self-stabilization of the distributed systems.

**Definition 2.9 (Attractor)** *Let $\mathcal{P}$ be a protocol. Let $X$ and $Y$ be two predicates defined on configurations. The predicate $Y$ is an attractor for the predicate $X$ ($X \triangleright Y$) if and only if the following condition is true:*

- **convergence** $\forall c_0 \vdash X : \forall e \in \mathcal{E}_{\mathcal{P}} : e = [(c_0, CH_0, c_1)\ldots] :: \exists i \geq 0, c_i \vdash Y$.

Informally, $X \triangleright Y$ means that in any computation of $\mathcal{E}_{\mathcal{P}}$ starting in a configuration satisfying the predicate $X$, the system is guaranteed to reach a configuration satisfying the predicate $Y$.

**Definition 2.10** *Let $\mathcal{P}$ be a protocol under a scheduler $D$. Let $Pr$ be a predicate on configurations. $Pr$ is closed if and only if on any computation step $(c, CH, c')$ such that $c \vdash Pr$, we have $c' \vdash Pr$.*

**Observation 2.4** *Let $L12$ be the predicate on configurations $L1 \wedge L2$. If $L1 \triangleright L2$ and if $L1$ is closed then $L1 \triangleright L12$.*

**Theorem 2.1 (Self-stabilization)** *A protocol $\mathcal{P}$ is self-stabilizing for a specification $SP$, if there exists a sequence of predicates $(true = L_1, L_2, \ldots, L_n)$ (where $L_n$ is the legitimate predicate) such that the following conditions hold:*

- **convergence** $\forall i \in [1, n-1] :: L_i \triangleright L_{i+1}$.
- **correctness** $\forall c_0 \vdash L_n : \forall e \in \mathcal{E}_{\mathcal{P}} : e = [(c_0, CH_0, c_1) \ldots] :: e \vdash SP$.

# 3 Probabilistic model

In a probabilistic distributed system, each processor has a random variable. The fundamental problems solved in this section is the definition of a probability on the system computations related to the random variables (thus to define a probabilistic space). Once this probability defined, we will be able to give a definition of the self-stabilization of a randomized protocol.

## 3.1 Field on a strategy

The basic notion that we will use to define a probabilistic space on the computations of a given strategy is the cone. Cones have been introduced in [21]. We mention here some properties of the union of cones, intersection of cones, and complementary of a cone in a strategy. Finally, we construct a field (a class of subsets of strategy's cones closed by finite unions and by complementary) on strategy computations.

In the sequel of this section, we will always refer to a protocol $\mathcal{P}$ under a scheduler $D$.

**Definition 3.1** *Let $st$ be a strategy. A* **cone** *$C_h$ of $st$ is the set of all $st$'s computations with the common prefix $h$. $h$ is called the history of the cone.*

**Notation 3.1** *Let $st$ be a strategy. Let $C_h$ be a cone of $st$.* **last(h)** *denotes the last configuration of $h$. The number of computation steps in the history $h$ is denoted by $|h|$.*

**Example:** See the figure 1, $h = [(c_0, CH_0, c_1)(c1, CH_1, c_4)]$, the $|h| = 2$. last(h) is the configuration $c_4$.

**Remark 3.1** *Let $st$ be a strategy. $st$ is a particular cone with an empty history. This cone is denoted by $\mathcal{C}^{st}$.*

**Definition 3.2** *Let $st$ be a strategy and let $C_h$ be a cone of $st$. The subcone $C_{h'}$ of the cone $C_h$ is the set of all computations of $C_h$ having $h'$ as a prefix.*

*Let $st$ be a strategy. Let $C_h$ be a cone of $st$. $C_h$ is a* **singular cone**, *if last(h) is a terminal configuration (i.e. no processor is enabled at last(h)).*

**Remark 3.2** *Let $C_{h'}$ be a subcone of the cone $C_h$ then we have $h' = hx$. A singular cone contains only one computation.*

We begin by a sequence of lemma; the three first being straightforward.

**Lemma 3.1** *Let $st$ be a strategy. Let $C_{h1}$ and $C_{h2}$ be two cones of $st$. The intersection of these cones is equal to (i) $C_{h1}$, (ii) $C_{h2}$ or (iii) is empty.*

**Lemma 3.2** *Let $st$ be a strategy. Let $A$ be a countable union of cones of $st$. There is a countable set of pairwise independent cones of $st$ so that their union is $A$.*

**Lemma 3.3** *Let $st$ be a strategy. Let $n \geq 1$ be an integer. Let $M$ be the set containing all cones of $st$ whose history length is $n$ and all singular cones of $st$ whose history length is $m$ where $m \leq n$. We have $\bigcup_{C \in M} C = st$.*

**Lemma 3.4** *Let $st$ be a strategy and let $C_h$ be a cone of $st$. Let $\overline{C_h}$ be the complementary of $C_h$ in $st$. $\overline{C_h}$ is a finite union of pairwise independent cones.*

**Proof:** Let $M1$ be the set of pairwise independent singular cones whose history length is lesser than $|h|$. Let $M2$ be the set of cones whose history length is equal to $|h|$. Let $M2' = \{C :: C \in M2 \text{ and } C \cap C_h = \oslash\}$. Let $M$ be $M1 \cup M2'$.
From the construction of the set $M$ and from the lemma 3.3, we have $\overline{C_h} = \bigcup_{C \in M} C$. Then, $\overline{C_h}$ is a finite union of cones. From lemma 3.2, $\overline{C_h}$ is a finite union of pairwise independent cones. □

**Lemma 3.5** *In a strategy, the intersection of two finite unions of pairwise independent cones is a finite union of pairwise independent cones.*

**Proof:** Let $A = \bigcup_{1 \leq i \leq n} C_{h_i}$ and $B = \bigcup_{1 \leq j \leq m} C_{t_j}$ be two finite unions of cones.
We have $A \cap B = \bigcup_{1 \leq i \leq n}(C_{h_i} \cap B) = \bigcup_{1 \leq i \leq n}(\bigcup_{1 \leq j \leq m}(C_{h_i} \cap C_{t_j}))$.
As the intersection of two cones is a cone or the empty set, $A \cap B$ is a finite union of cones. Then $A \cap B$ is a finite union of pairwise independent cones (lemma 3.2). □

**Corollary 3.1** *In a strategy, the finite intersection of finite unions of pairwise independent cones of a strategy is a finite union of pairwise independent cones.*

**Lemma 3.6** *In a strategy, the complementary of a finite union of pairwise independent cones is a finite union of pairwise independent cones.*

**Proof:** The complementary of a finite union of pairwise independent cones is the finite intersection of the union's complementaries. □

**Notation 3.2** *Let $st$ be a strategy. We note $\mathcal{F}_{st}$ the set of all finite unions of pairwise independent cones of the strategy $st$.*

A field of a strategy is a class of subsets of strategy's cones closed by finite unions and by complementary. Formally:

**Definition 3.3** *Let $st$ be a strategy. Let $\mathcal{S}$ be a class of subsets of $st$. $\mathcal{S}$ is a field of $st$ if and only if (1) $st \in \mathcal{S}$, (2) $A \in \mathcal{S}$ implies $\overline{A} \in \mathcal{S}$, and (3) $\forall i \in [1,n] :: A_i \in \mathcal{S}$ implies $\bigcup_{1 \leq i \leq n} A_i \in \mathcal{S}$.*

**Theorem 3.1** *Let $st$ be a strategy. $\mathcal{F}_{st}$ is a field.*

**Proof:** As $C^{st} = st$, $st$ is an element of $\mathcal{F}_{st}$, The finite union of finite unions of pairwise independent cones is a finite union of cones of $st$. This union can be expressed as a finite union of pairwise independent cones of $st$ (lemma 3.2). $\mathcal{F}_{st}$ is closed by finite unions. $\mathcal{F}_{st}$ is closed by complementary (lemma 3.6). □

## 3.2 Strategy - probabilistic space

A probabilistic space is by definition a triple $(\Omega, \mathcal{F}, P)$ where $\Omega$ is a set, $\mathcal{F}$ is a $\sigma$-field of $\Omega$ and $P$ is a probabilistic measure. A $\sigma$-field is a class of subsets of $\Omega$ containing $\Omega$, closed by complementary and by countable unions. A probabilistic measure is defined from $\mathcal{F}$ to $[0,1]$ and verifies the following properties: P($\Omega$)=1 and if $A_1, A_2, \ldots,$ is a disjoint sequence of $\mathcal{F}$ sets then $P(\bigcup_{k=1}^{\infty} A_i) = \sum_{k=1}^{\infty} P(A_k)$. According to the properties of $P$, if $A$ is an element of $\mathcal{F}$, then we have $P(\overline{A}) = 1 - P(A)$.

### 3.2.1 Probability of a computation step

In a randomized protocol, each processor has a random variable. The output of an action of a processor $p$ depends on the value of $p$ random variable. The random variables are independent, thus the output of a $p$ action is independent of the output of an action of another processor. The probability of a computation step is the product of probability of every output of actions that have been performed during the computation step.

**Definition 3.4** *The probabilistic value associated to a computation step, $pr(c, CH, c')$ is defined by: $pr(c, CH, c') = \prod_{p \in CH} pr(X_p = val_p)$, where $X_p$ is the random variable of the processor $p$, $val_p$ is a value of $X_p$, and $c'$ is the obtained configuration after that all processors of $CH$ have set their $X_p$ variable to $val_p$ and have performed an action.*

**Observation 3.1** *It is easy to prove that $\sum_{c' \in <c:CH>} pr(c, CH, c') = 1$.*

### 3.2.2 Probabilistic space on a strategy

In this section, we equip a strategy $st$ with a probability space. The construction of the probability measure will be made hierarchically using results of the classical theory of probabilities. We will define a probabilistic measure on $\mathcal{F}_{st}$ (a field of $st$). Well-known results of probabilistic theory establish that a field can be extended by closure to a $\sigma$-field. The probabilistic measure defined on the field can be also extended to the $\sigma$-field. Also, we will construct a probabilistic space on top of $st$.

Let $st$ be a strategy. We associate to the cone $C_h$ a value, function of its history, by extending the probability $pr$ defined on computation steps. The value of $C_h$ is the product of the probabilities of each computation step of $h$ (the computation step probabilities are independent). From these values, we will build a probability measure on $\mathcal{F}_{st}$.

**Definition 3.5** *Let st be a strategy. The value attached to the cone $C_h$ in st is:*

$P_{st}(C_h) = \prod_{k=0}^{j} pr(c_k, CH_k, c_{k+1})$, *where* $h = [(c_0, CH_0, c_1)(c_1, CH_1, c_2) \ldots (c_j, CH_j, c_{j+1})]$.

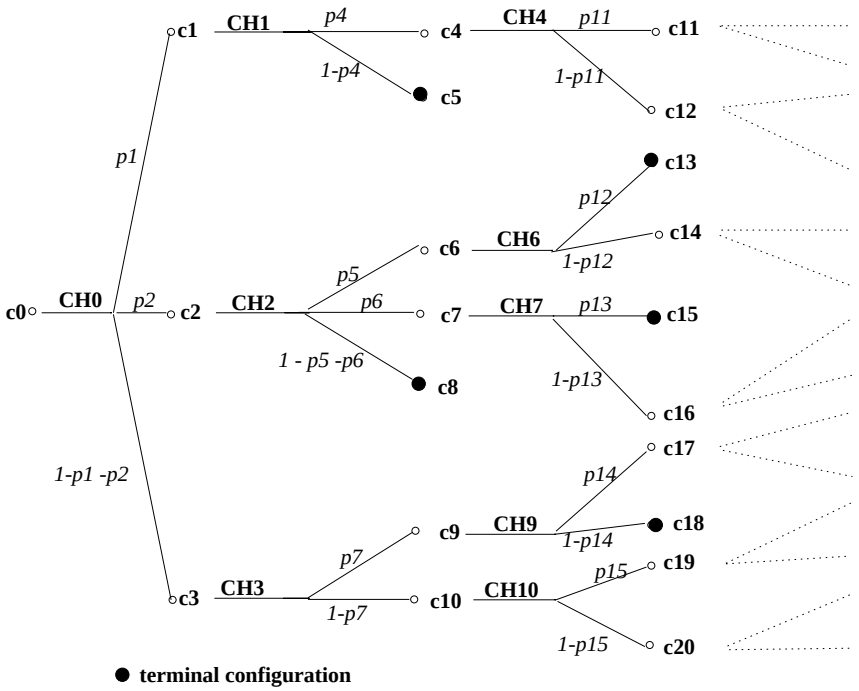


Figure 1: **A strategy with the probabilities of the computation steps**

**Example:** See the figure 1, $h1 = [(c_0, CH_0, c_1)(c_1, CH_1, c_5)]$, $P_{st}(C_{h1}) = p1.(1 - p4)$. $C_{h1}$ is a singular cone. $h2 = [(c_0, CH_0, c_2)(c_2, CH_2, c_7)(c_7, CH_7, c_{16})]$, $P_{st}(C_{h2}) = p2.p6.(1 - p13)$.

**Lemma 3.7** *Let st be a strategy. Let $C_h$ be a non singular cone of st. Let $M$ be the set of subcones of $C_h$ whose history length is $|h| + 1$. We have $P_{st}(C_h) = \sum_{C \in M} P_{st}(C)$*

**Proof:** Let $h'$ be a computation step defined as $(c, CH, c')$. $C_{hh'} \in M$ if and only if (1) $c =$ last(h), (2) $CH$ is the choice of $st$ at $h$ ($f_{st}(h) = CH$), and (3) $c_h \in < c : CH >$. By definition of $P_{st}$, if $h' = (c, CH, c)$ then $P_{st}(C_{hh'}) = P_{st}(C_h).pr(c, CH, c_h)$. We have $\sum_{C_{hh'} \in M} P_{st}(C_{hh'}) = \sum_{c_h \in <c:CH>} (P_{st}(C_h).pr(c, CH, c_h)) = P_{st}(C_h)$. □

**Corollary 3.2** *Let st be a strategy. We have $P_{st}(C^{st}) = 1$*

**Proof:** Let $M$ be the set containing all distinct cones of $st$ whose history length is 1. We have $\bigcup_{C \in M} C = st = C^{st}$ (lemma 3.3). According to 3.7, we have $P_{st}(\bigcup_{C \in M} C) = 1$ □

**Lemma 3.8** *Let st be a strategy. Let $C_h$ be a cone of st. Let $C_{h1}, C_{h2}, \ldots$ be a series of pairwise independent cones of st such that $C_h = \bigcup_{1 \leq i \leq n} C_{hi}$. We have $P_{st}(C_h) = \sum_{1 \leq i \leq n} P_{st}(C_{hi})$.*

**Proof:** We prove this lemma by induction on the length of the series $C_{h1}, C_{h2}, \ldots$ . If the series has one element, the lemma is verified. Assume that the lemma is verified when the series contains less than $n$ cones.

Let $C_{h1}, C_{h2}, \ldots$ be a series of $n$ pairwise independent cones of $st$ such that $C_h = \bigcup_{1 \leq i \leq n} C_{hi}$. $C_h$ is not singular. Let $M1$ be the set of independent subcones of $C_h$ whose history length is $|h| + 1$. We have $P_{st}(C_h) = \sum_{C \in M1} P_{st}(C)$ (lemma 3.7).

Let $C_{h'}$ be a cone of $M1$. We define $M_{h'}$ as $M_{h'} = \{C_{h1} \cap C_{h'}, C_{h2} \cap C_{h'}, \ldots\}$. According to the property of cone intersection we have $\forall i \in [1, n], C_{h'} \cap C_{hi} = \oslash$ or $C_{hi} \subset C_{h'}$; thus, $M_{h'} = \{C_{hi} :: C_{hi} \subset C_{h'}\}$. As $C_h = \bigcup_{1 \leq i \leq n} C_{hi}$, we have $C_{h'} = \bigcup_{c_{hi} \in M_{h'}} C_{hi}$ and $\bigcup_{C_{h'} \in M1} M_{h'} = \bigcup_{1 \leq i \leq n} C_{hi}$. $M_{h'}$ contains at most $n - 1$ cones; thus $P_{st}(C'_h) = \sum_{c_{hi} \in M_{h'}} P_{st}(C_{hi})$.

$P_{st}(C_h) = \sum_{C_{h'} \in M1} P_{st}(C_{h'}) = \sum_{C_{h'} \in M1} (\sum_{c_{hi} \in M_{h'}} P_{st}(C_{hi})) = \sum_{1 \leq i \leq n} P_{st}(C_{hi})$. □

**Lemma 3.9** *Let $st$ be a strategy. Let $A = \bigcup_{1 \leq i \leq n} C_i^a$ be a finite union of pairwise independent cones of $st$. Let $B = \bigcup_{1 \leq i \leq m} C_i^b$ be a finite union of pairwise independent cones of $st$. If $A = B$ then $\sum_{1 \leq i \leq n} P_{st}(C_i^a) = \sum_{1 \leq i \leq m} P_{st}(C_i^b)$.*

**Proof:** We split the cones of $A$ into three sets: (1) a cone of $A1$ is a subcone of a cone of $B$; (2) a cone of $A2$ contains a cone of $B$; and (3) a cone of $A3$ is also a cone of $B$. Formally, we have $A1 = \{C_i^a :: \exists C_j^b : C_i^a \subset C_j^b \text{ and } C_i^b \neq C_j^a\}$; $A2 = \{C_i^a :: \exists C_j^b : C_i^b \subset C_j^a \text{ and } C_i^b \neq C_j^a\}$; $A3 = \{C_i^a :: \exists C_j^b : C_i^b = C_j^a\}$. These three sets are disjoint because the cones $C_j^b$ are disjoint. Similarly, we split the cone of $B$ into three disjoint sets. For each cone of $A2$, $C_i^a$, we build $M_i^a$ the set of $B1$ cones that are subcones of $C_i^a$. Formally, we have: $M_i^a = \{C_j^b \in B1 :: C_j^b \subset C_i^a\}$.

Let $C_i^a$ be a cone of $A2$. Let $C_j^b$ be a cone of $B2 \cup B3$. We have $C_i^a \cap C_j^b = \oslash$; because the cone of $A$ and $B$ are pairwise independent. Moreover $A = B$; thus we have $\bigcup_{C_j^b \in M_i^a} C_j^b = C_i^a$. We also have (lemma 3.8) $\sum_{C_j^b \in M_i^a} P_{st}(C_j^b) = P_{st}(C_i^a)$.

A cone of $B1$ cannot be included in a cone of $A1$ or $A3$ because the cones of $B$ are pairwise independent; thus $\bigcup_{C_i^a \in A2} M_i^a = B1$.
$\sum_{C_i^a \in A2} (\sum_{C_j^b \in M_i^a} P_{st}(C_j^b)) = \sum_{C_j^b \in B1} P_{st}(C_j^b) = \sum_{C_i^a \in A2} P_{st}(C_i^a)$.
Similarly, we prove that $\sum_{C_j^b \in B2} P_{st}(C_j^b) = \sum_{C_i^a \in A1} P_{st}(C_i^a)$.
Clearly, we have $\sum_{C_j^b \in B3} P_{st}(C_j^b) = \sum_{C_i^a \in A3} P_{st}(C_i^a)$. $\sum_{1 \leq i \leq n} P_{st}(C_i^a) = \sum_{1 \leq i \leq m} P_{st}(C_i^b)$. □

**Definition 3.6** *Let $st$ be a strategy. Let $P_{st}$ be the function defined as follow:*

$$P_{st}(C_h) = \prod_{k=0}^{j} pr(c_k, CH_k, c_{k+1}), \text{ where } h = [(c_0, CH_0, c_1)(c_1, CH_1, c_2) \ldots (c_j, CH_j, c_{j+1})].$$

$$P_{st}(A) = \sum_{i=1}^{n} P_{st}(C_i) \text{ where } A = \bigcup_{i=1}^{n} C_i \text{ and the cones } C_i \text{ are pairwise independent.}$$

**Observation 3.2** *$P_{st}$ is effectively a function, because the image of an element of $\mathcal{F}_{st}$ by $P_{st}$ is unique (lemma 3.9).*

**Lemma 3.10** *Let $st$ be a strategy. Let $A$ be an element of $\mathcal{F}_{st}$. We have $P_{st}(A) + P_{st}(\overline{A}) = 1$*

**Proof:** Let $M$ be the set of of pairwise independent cones whose history length is equal to 1. According to the corollary 3.2, we prove that $P_{st}(\bigcup_{C\in M} C) = 1$. We have $\bigcup_{C\in M} C = st = A \cup \overline{A}$. As $A$ and $\overline{A}$ are pairwise independent, $1 = P_{st}(A \cup \overline{A}) = P_{st}(A) + P_{st}(\overline{A})$. □

**Theorem 3.2** *Let st be a strategy. The function $P_{st}$ is a probabilistic measure defined on the field $\mathcal{F}_{st}$.*

**Proof:** $P_{st}$ is a probabilistic measure because the following properties are verified:

- According to the corollary 3.2, $P_{st}(st) = 1$.
- Let $A$ be an element of $\mathcal{F}_{st}$. we have $P_{st}(A) + P_{st}(\overline{A}) = 1$ (lemma 3.10). By definition of $P_{st}$, $P_{st}(A) \geq 0$ and $P_{st}(\overline{A}) \geq 0$. Thus $P_{st}(A) \in [0, 1]$. □

**Definition 3.7** *Let st be a strategy. Let $\mathcal{S}$ be a class of subsets of st. $\mathcal{S}$ is a $\sigma$-field of st if and only if (1) $C^{st} \in \mathcal{S}$, (2) $A \in \mathcal{S}$ implies $\overline{A} \in \mathcal{S}$, and (3) $\forall i :: A_i \in \mathcal{S}$ implies $\bigcup_{i=0}^{\infty} A_i \in \mathcal{S}$.*

**Notation 3.3** *Let $\sigma(\mathcal{F}_{st})$ be the $\sigma$-field generated by $\mathcal{F}_{st}$.*

**Theorem 3.3** *There is a unique extension, $P^*_{st}$, of the probabilistic measure $P_{st}$ to $\sigma(\mathcal{F}_{st})$.*

**Proof:** The extension is made according to the classical theory of probabilities. This function is a probabilistic measure on the $\sigma(\mathcal{F}_{st})$. For more details see chapter 1 of [6]. □

Let $st$ be a strategy. The triple $(st, \sigma(\mathcal{F}_{st}), P^*_{st})$ defines a probabilistic space on $st$.

In the following sections we denote by $P_{st}$: $P^*_{st}$ -the extension of $P_{st}$ to $\sigma(\mathcal{F}_{st})$-.

### 3.3 Self-Stabilization of a randomized protocol

The self-stabilization for the deterministic protocols was defined in the section 1. In this section we are interested in defining the self-stabilization for the probabilistic protocols with respect to the probabilistic model defined in the previous section. This section introduces also the probabilistic version of the attractor and the definition of probabilistic self-stabilization using probabilistic attractors.

**Notation 3.4** *Let st be a strategy of a protocol under a scheduler D. Let PR be a predicate over configurations. We note by $\mathcal{EPR}_{st}$ the set of st computations reaching a configuration that satisfies the predicate PR.*

**Lemma 3.11** *Let st be a strategy. Let PR be a predicate over configurations. There is a countable union of pairwise independent cones ($A = \bigcup_{i\in N} C_i$) so that $\mathcal{EPR}_{st} = A$.*

**Proof:** Let $M_n$ be defined as $M_n = \{C_h :: |h| = n \text{ and last(h)} \vdash PR\}$. The number of the cones in the set $M_n$ is finite.
Let $M$ be defined as $M = \bigcup_{i\in N} M_i$. By definition of $PR_{st}$, $\mathcal{EPR}_{st} = \bigcup_{C\in M} C$.
$M$ is the countable union of finite sets, thus $M$ is a countable set. Then $\mathcal{EPR}_{st}$ is a countable union of pairwise independent cones (lemma 3.2). □

In the following we will define the self-stabilization for a probabilistic protocol under a scheduler $D$.

**Definition 3.8 (Probabilistic self-stabilization)** *A probabilistic distributed protocol $P$ is self-stabilizing under a scheduler $D$ for a specification $SP$ if and only if there is a predicate $L$ on configurations (defining the legitimate configurations) such that:*

- **probabilistic convergence** *In any strategy $st$ of $P$ under $D$ the probability of the set of computations reaching a legitimate configuration is equal to 1. Formally, $\forall st, P_{st}(\mathcal{EL}_{st}) = 1$.*
- **correctness** *In any strategy $st$, the probability for the set of computations reaching a legitimate configuration and then from this legitimate configuration verifying $SP$ is 1. Formally, $\forall st, P_{st}(\{e \in st : e = [e1, e2],\ last(e1) \vdash L,\ and\ e2 \vdash SP\}) = 1$.*

**Definition 3.9 (Probabilistic Attractor)** *Let $L1$ and $L2$ be two predicates defined on configurations. $L2$ is a probabilistic attractor for $L1$ on a protocol $P$ under a scheduler $D$ ($L1 \triangleright_{prob} L2$) if and only if the following condition holds:*

- **probabilistic convergence** *for all strategies $st$ of $P$ under $D$ such that $P_{st}(\mathcal{EL}_1) = 1$, we have: $P_{st}(\mathcal{EL}_2) = 1$, Formally, $\forall st, P_{st}(\mathcal{EL}_1) = 1 \Rightarrow P_{st}(\mathcal{EL}_2) = 1$.*

**Theorem 3.4 (Probabilistic Self-stabilization)** *A randomized protocol $\mathcal{P}$ is self-stabilizing for a specification $SP$, if there exists a sequence of predicates $(true = L_1, L_2, \ldots, L_n)$ (where $L_n$ is the legitimate predicate) such that the following conditions hold:*

- **probabilistic convergence** $\forall i \in [1, n-1] :: L_i \triangleright_{prob} L_{i+1}$.
- **probabilistic correctness** $\forall st, P_{st}(\{e \in st :: e = [e1, e2],\ last(e1) \vdash L_n,\ and\ e2 \vdash SP\}) = 1$.

**Observation 3.3** *We note $L12$ the following predicate on configurations $L1 \wedge L2$. If $L1 \triangleright_{prob} L2$ and if $L1$ is closed then $L1 \triangleright_{prob} L12$.*

### 3.4 Proving the convergence of self-stabilizing protocols

In this section, we present a theorem that helps to build convergence proofs of randomized protocols. This theorem can be used in case of a proof via attractors, but also in case of a direct proof.

Informally the next definition is introduced for dealing with such a statement: "in a cone where the predicate $PR1$ is satisfied, the probability to reach a configuration satisfying the predicate $PR2$ in less than $n$ steps is greater than $\delta$".

**Definition 3.10** *(Local convergence) Let $st$ be a strategy. Let $C_h$ be a cone in the strategy $st$. The cone $C_h$ holds the property $Local\_Convergence(PR1, PR2, \delta, n)$ if and only if:*

- *last(h) $\vdash PR1$;*
- *$M$ is the set of pairwise independent subcones of $C_h$ ($C_{hh'}$) such that (1) $|h'| \leq n$, and (2) last(hh') $\vdash PR2$;*
- *$P_{st}(\bigcup_{C \in M} C) \geq \delta . P_{st}(C_h)$.*

*On a strategy $st$, if it exists $\delta_{st} > 0$ and $n_{st} > 1$ such that any cone of $st$ satisfies $Local\_Convergence$ ($PR1$, $PR2$, $\delta_{st}$, $n_{st}$) then we said that $st$ verifies the $Convergence$ ($PR1$, $PR2, \delta_{st}, n_{st}$) property or $Convergence$ ($PR1$, $PR2$) property.*

**Theorem 3.5** *Let $st$ be a strategy of the protocol $P$ under a scheduler $D$. Let $PR1$ be a closed predicate on configurations such that $P_{st}(\mathcal{EPR}1) = 1$. Let $PR2$ be a closed predicate on configurations. Let us note $PR12$ the predicate $PR1 \wedge PR2$. If $\exists\ \delta_{st} > 0$ and $\exists\ n_{st} > 1$ such that $st$ verifies the Convergence$(PR1, PR2, \delta_{st}, n_{st})$ property then $P_{st}(\mathcal{EPR}12) = 1$.*

**Proof:** Let $EL_k$ be the set of computations reaching a configuration satisfying $PR1$ and, after that, in at most $k.n_{st}$ steps they reach a configuration satisfying the predicate $PR2$. We prove that $P_{st}(EL_k) \geq 1-(1-\delta)^k$ and $\overline{EL_k} \cap \mathcal{EPR}1$ is a countable union of pairwise independent cones where $P_{st}(\overline{EL_k} \cap \mathcal{EPR}1) = 1 - P_{st}(EL_k)$.
Let $C_h$ be a cone of $st$. We define $M2'_h$ as $M2'_h = \{C_{hh'} :: |h'| \leq n_{st}$, and last(hh') $\vdash PR2\}$. We define $M1'_h$ as $M1'_h = \{C_{hh'} :: |h'| \leq n_{st}$, last(hh') does not verify $PR2$, and either last(hh') is a terminal configuration or $|h'| = n_{st}\}$. $M2'_h$ ($M1'_h$) being a set of cones, according to lemma 3.2, there is a set of pairwise independent cones $M2_h$ ($M1_h$) such that $M2_h = M2'_h$ ($M1_h = M1'_h$).
The cones of $M2_h$ contain all computations of $C_h$ that reach $PR2$ in less than $n_{st}$ steps. The cones of $M1_h$ contain the other computations. By hypothesis $P_{st}(\bigcup_{C \in M2_h} C) \geq P_{st}(C_h).\delta_{st}$.

- **Basic step (n=1).** $\mathcal{EPR}1$ is a countable union of pairwise independent cones (lemma 3.11). $\mathcal{EPR}1 = \bigcup_{C \in M_1} C$ where $M_1$ is a countable set of pairwise independent cones. From the hypothesis, $P_{st}(\bigcup_{C \in M_1} C) = \sum_{C \in M_1} P_{st}(C) = 1$.

  We have $EL_1 = \bigcup_{C_h \in M_1} (\bigcup_{C \in M2_h} C)$; thus $P_{st}(EL_1) \geq \delta_{st}. \sum_{C_h \in M_1} P_{st}(C_h)$.

  So, $P_{st}(EL_1) \geq \delta_{st}.P_{st}(\mathcal{EPR}1) \geq \delta_{st} = 1 - (1 - \delta_{st})$.

  All computations of a cone of $M1_h$ belongs to $\overline{EL_1}$, all computations of $C_h$ that belongs to $\overline{EL_1}$ are in a cone of $M1_h$. Thus, $\overline{EL_1} \cap \mathcal{EPR}1 = \bigcup_{C_h \in M_1} (\cup_{C \in M1_h} C)$. Then $\overline{EL_1} \cap \mathcal{EPR}1$ is a countable union of pairwise independent cones (lemma 3.2). As $P_{st}(\overline{\mathcal{EPR}1}) = 0$, we have $P_{st}(\overline{EL_1} \cap \mathcal{EPR}1) = 1 - P_{st}(EL_1)$

- **Induction step.** We suppose the hypothesis are true for k-1.

  By hypothesis, $\overline{EL_{k-1}} \cap \mathcal{EPR}1$ is a countable union of pairwise independent cones. We call $M_k$ the set of independent cones whose the union is equal to $\overline{EL_{k-1}} \cap \mathcal{EPR}1$. Thus, $\overline{EL_{k-1}} \cap \mathcal{EPR}1 = \bigcup_{C \in M_k} C$, and $P_{st}(\overline{EL_{k-1}} \cap \mathcal{EPR}1) = \sum_{C \in M_k} P_{st}(C)$.

  We name $Diff_k$ the computations set such that (1) $EL_k = EL_{k-1} \cup Diff_k$, and (2) $diff_k \cap EL_{k-1} = \oslash$. $Diff_k = \bigcup_{C \in D_k} C$ where $D_k = \{C_{hh'} :: C_h \in M_k$, and $C_{hh'} \in M2_h\}$.

  We have $Diff_k = \bigcup_{C_h \in M_k} (\bigcup_{C \in M2_h} C)$; thus $P_{st}(Diff_k) \geq \delta_{st}. \sum_{C_h \in M_k} P_{st}(C_h)$.
  So $P_{st}(Diff_k) \geq \delta_{st}.(1 - P_{st}(EL_{k-1}))$. $P_{st}(EL_k) \geq 1 - (1 - \delta_{st})^k$.

  We have $(\overline{EL_k} \cap \mathcal{EPR}1) \subset (\overline{EL_{k-1}} \cap \mathcal{EPR}1)$, thus, we prove as in the basic step that $\overline{EL_k} \cap \mathcal{EPR}1 = \bigcup_{C_h \in M_k} (\cup_{C \in M1_h} C)$. Then, $\overline{EL_1} \cap \mathcal{EPR}1$ is a countable union of pairwise independent cones (lemma 3.2). As $P_{st}(\overline{\mathcal{EPR}1}) = 0$, we have $P_{st}(\overline{EL_k} \cap \mathcal{EPR}1) = 1 - P_{st}(EL_k)$.

$P_{st}(EL_n) \geq 1 - (1 - \delta)^n$. Therefore, $P(\mathcal{EPR}12) = \lim_{n \to \infty} P(EL_n) = 1$. □

**Corollary 3.3** *Let $PR1$ and $PR2$ be two closed predicate on configurations. If each strategy $st$ of a protocol $P$ under a scheduler $D$, verifies Convergence$(PR1, PR2)$ then $(PR1 \triangleright_{prob} PR2)$.*

**Corollary 3.4** *Let $PR2$ be a closed predicate on configurations. If each strategy $st$ of a protocol $P$ under a scheduler $D$ verifies Convergence$(true, PR2)$ then $\forall st, P_{st}(\mathcal{EPR}2) = 1$.*

# 4 Cross-over composition

This composition is designed as a tool for scheduler management. An incipient form of this composition was presented in [5]. Here a formal definition of this composition is provided and an example of scheduler controlling through this quite particular composition is given. In a cross-over composition, the actions of an initial protocol $W$ are synchronized with the actions of a second protocol $S$: the $W$ actions are performed only when a $S$ action is performed too. Thus, the computations of the composite protocol under any scheduler have the same properties as the computations of $S$ in term of fairness.

**Definition 4.1** *Let* **W** *and* **S** *be two arbitrary protocols having no common variable. The* **cross-over composition** *between* **W** *and* **S** *(denoted by* $\mathbf{W}\diamondsuit\mathbf{S}$*) is the protocol defined as:*

- *For every action of* $W$*:* $< l\_W >::< g\_W > \rightarrow < s\_W >$*, and for every action of* $S$ $< l\_S >::< g\_S > \rightarrow < s\_S >$*, the composite protocol contains the following action:*
  $< l_W, l_S >:: < g\_W > \wedge < g\_S > \longrightarrow < s\_W >; < s\_S >$
- *For every action of* $S < l\_S >::< g\_S > \rightarrow < s\_S >$*, the composite protocol contains the following action:*
  $< l\_S >::<$ *no* $W$ *guard holds* $> \wedge < g\_S > \longrightarrow < s\_S >$

*In* $\mathbf{W}\diamondsuit\mathbf{S}$*,* **W** *is called the "weaker" protocol and* **S** *is called the "stronger" .*

$W\diamondsuit S$ has the following properties:

- the actions of composite protocol are constructed according to the actions of its parents;
- an action of $W$ is performed, if an action guard of $S$ is satisfied and if the $W$'s action guard is also satisfied (simultaneously the both actions are performed);
- an action of $S$ is performed even if no action guard of $W$ is satisfied (but the guard action of $S$ is satisfied).

For example, the protocol SSCTC (protocol 5.3) is the cross-over composition of the protocol CTC (protocol 5.2) and the protocol DTC (protocol 5.1).

## 4.1 Projection

The influence of each parent on the child can be seen by projecting the child computation on its parents.

Let $c$ be a configuration of $W\diamondsuit S$. The protocols $W$ and $S$ have no common variable; so $c$ is the product of a configuration of $W$ and a configuration of $S$. The projection of $c = c_W c_S$ on $W$ ($S$) is the product of the values of $W$'s variable ($S$'s variable) on each processor in $c$ and is equal to $c_W$ ($c_S$).

Let $(c, CH, c')$ be a computation step of $W\diamondsuit S$. The projection of $(c, CH, c')$ on $S$ is $(c_S, CH, c'_S)$ (all processors of $CH$ perform an $S$'s action in this computation step). We call $CH_W$ the set of $CH$'s processors that perform an $W$'s action in the computation step $(c, CH, c')$. If $CH_W$ is not empty, then the projection of $(c, CH, c')$ on $W$ is $(c_W, CH_W, c'_W)$; otherwise the projection is empty.

Let $\omega$ be a computation of $W\diamondsuit S$. We call $\omega_S$ ($\omega_W$) the projections of $\omega$ on the protocol $S$ ($W$). These projections are obtained by projecting every computation step of $\omega$ on the selected protocol.

**Example:** Let $e$ be the computation $W \diamondsuit S$ defined as $[(c0_w c0_s, CH0, c6_w c1_s)\ (c6_w c1_s, CH6, c6_w c2_s)\ (c6_w c2_s, CH4, c6_w c4_s)\ (c6_w c4_s, CH3, c14_w c5_s)]$ (see the figure 2). The projection of $e$ on $W$ is $[(c0_w, CH0_w, c6_w)(c6_w, CH3_w, c14_w)]$. The projection of $e$ on $S$ is $[(c0_s, CH0, c1_s)(c1_s, CH6, c2_s)\ (c2_s,\ CH4,\ c4_s)(c4_s, CH3, c5_s)]$.

**Observation 4.1** *Every maximal no-empty computation of $W \diamondsuit S$ under the scheduler $D$ has a maximal, non empty projection on $S$ under the scheduler $D$.*

**Definition 4.2 (Total fairness property)** *Let $P$ be a protocol. $P$ is total fair if all $P$ maximal computations under any scheduler contains an infinity of actions of each processor.*

The following theorem introduces a general feature of the cross-over composition - preservation of the stronger properties.

**Theorem 4.1** *Let $W$ and $S$ be two protocols. Let $P$ be a property of maximal computation of $S$. $W \diamondsuit S$ holds the property $P$.*

**Proof:** Let $\omega$ be a maximal computation of $W \diamondsuit S$. From the observation 4.1 any maximal computation of $W \diamondsuit S$ has a maximal projection on $S$. Hence, $\omega_S$ (projection $\omega$ on $S$) is a maximal computation of $S$. $\omega_S$ holds the property $P$. □

**Corollary 4.1** *Let $W$ and $S$ be two protocols. If $S$ is total fair then $W \diamondsuit S$ is total fair.*

The properties of the projection of $W \diamondsuit S$ on $W$ depend on the properties of $S$. We give a necessary condition on $S$ to ensure that the maximality of the projection on $W$ of any computation of $W \diamondsuit S$.

**Lemma 4.1** *Let $W$ and $S$ be two protocols. If $S$ is total fair then every maximal computation of $W \diamondsuit S$ has a maximal projection on $W$.*

**Proof:** Let $\omega$ be a maximal computation of $W \diamondsuit S$. Let $\omega_W$ be the projection of $\omega$ on $W$. Suppose that $\omega_W$ is finite and its final configuration is not terminal. We call $c_W$ the final configuration of $\omega_W$. $c_W$ is the projection of a configuration of $\omega$ that we call $c$. In $\omega$, from $c$ no action of $W$ is performed. In $c$, there is at least one processor having a $W$ guard verified. Let $p$ be such a processor. The processor $p$, due to the total fairness property of the composite (corollary 4.1), will perform an action in $\omega$. According to the cross-over composition definition, when $p$ performs an action of $S$, if $p$ holds an action guard of $W$ then $p$ performs in the same computation step the $W$ action. In this case, $\omega$ contains an action of $W$ after reaching $c$. When $p$ performs an action of $S$, it is possible that the processor $p$ does not hold any action guard of $W$. But in this case, one of $p$'s neighbors has performed an action of $W$. Therefore, $\omega$ contains an action of $W$ after reaching $c$. □

## 4.2 Scheduler managing by cross-over composition

Some protocols are self-stabilizing under some specific schedulers. For instance, under a $k$-bounded scheduler (i.e. selecting computations verifying the following property "until a processor $p$ is enabled another processor can perform at most $k$ actions"). In this section, we study the necessary and

sufficient conditions to transform a self-stabilizing protocol under a $k$-bounded scheduler in a self-stabilizing protocol under an arbitrary scheduler. The idea of the transformation is the cross-over composition between the initial protocol, playing the "weaker" role and a specific protocol. In this paper, we only present this type of transformation but the power of cross-over composition is not limited to this particular case.

Before presenting this transformation in the case of deterministic and probabilistic protocols; we give some basic notations.

**Definition 4.3** *A computation $\omega$ is* **$k$-fair** *if and only if (1) any processor $p$ infinity often performs an action, and (2) between two actions of $p$, a processor performs at most $k$ actions.*

*An arbitrary protocol is $k$-fair, if all its computations under any scheduler are $k$-fair.*

**Remark 4.1** *A $k$-fair protocol is a total fair protocol but the reverse is not true.*

**Lemma 4.2** *Let $W$ and $S$ be two protocols. If $S$ is $k$-fair then $W \diamondsuit S$ is $k$-fair.*

**Proof:** This lemma is a corollary for the theorem 4.1. □

In the following, we consider a special type of scheduler called $k$-bounded scheduler. The scheduler considered in [4] is a $k$-bounded scheduler.

**Definition 4.4** *Let $\omega$ be a computation. $\omega$ is* **$k$-bounded** *if and only if along $\omega$, till a processor $p$ is enabled another processor can perform at most $k$ actions.*

*A scheduler is called $k$-bounded if and only if it selects only $k$-bounded computations.*

**Remark 4.2** *A $k$-fair computationis $k$-bounded; but the converse is not true.*

**Lemma 4.3** *Let $W$ and $S$ be two protocols. Let $\omega$ be a maximal computation of $W \diamondsuit S$ under any scheduler. $\omega_W$ is the projection of $\omega$ on $W$. If $S$ is $k$-fair then $\omega_W$ is a maximal $k$-bounded computation of $W$.*

**Proof:** $\omega_W$ is maximal because $S$ is total fair (lemma 4.1).
Suppose that $\omega_W$ does not satisfy the $k$-bounded predicate. Hence, there is a fragment $F_W$ of $\omega_W$ such that the processor $q$ performs $k+1$ actions, $p$ performs no action and $p$ is always enabled. $F_W$ is the projection of a fragment of $\omega$ called $F$. According to the definition of $W \diamondsuit S$, $F$ has the following property (i) $p$ performs no action in $F$ (ii) $q$ performs at least $k+1$ actions in $F$. $F$ is part of an fragment of $p$ called $F_{pp}$ verifying the following properties: (1) $F_{pp}$ begins and ends by an action of $p$; (2) $F_{pp}$ contains only two actions of $p$; and (3) in $F_{pp}$, $q$ performs at least $k+1$ actions. $F_{pp}$ cannot exist because $W \diamondsuit S$ is $k$-fair (lemma 4.2). □

We prove that $W \diamondsuit S$ is self-stabilizing to $SP$, if $S$ is $k$-fair and $W$ is self-stabilizing to $SP$ under a $k$-bounded scheduler.

**Theorem 4.2** *Let $S$ be a deterministic and $k$-fair protocol. Let $W$ be a deterministic protocol self-stabilizing to $SP$ under a $k$-bounded scheduler. $W \diamondsuit S$ is self-stabilizing for the specification $SP$ under any scheduler.*

**Proof:** Let $\omega$ be a maximal computation of $W \diamondsuit S$. $\omega_W$ is the projection on $W$ of $\omega$. From the lemma 4.3, the computation $\omega_W$ is maximal and it satisfies the $k$-bounded predicate. The weaker protocol is self-stabilizing under $k$-bounded scheduler, then a suffix of $\omega_W$ satisfies $SP$. Thus, the computation $\omega$ has a suffix satisfying the predicate $SP$. □

### 4.2.1 Projection of a strategy

In this section, we present the properties of the projection of a strategy of $W \diamondsuit S$ on $W$ when $S$ is deterministic and $k$-fair. The figure 2 displays an example of such a projection.

Let $S$ be a deterministic and $k$-fair protocol. Let $W$ be a randomized protocol. Let $st$ be a strategy of $W \diamondsuit S$. $st_W$ is the projection of $st$ on $W$. Formally, $st_W = \{e \in \mathcal{E}_W :: \exists e' \in st$ such that the projection of $e'$ on $W$ is $e\}$. As $S$ is $k$-fair, the computations of $st_W$ are maximal and $k$-bounded (lemma 4.3). In what follows, we prove that $st_W$ is a strategy of $W$ under a $k$-bounded scheduler.

**Lemma 4.4** *Let* $e1_W$ *and* $e2_W$ *be two computations of* $st_W$. *If* $e1_W = [e_W(c1_W, CH1_W, c3_W)$ $e12_W]$ *and* $e2_W = [e_W(c2_W, CH2_W, c4_W)e22_W]$ *then* $c1_W = c1_W$ *and* $CH1_W = CH2_W$.

**Proof:** $e1_w$ ($e2_w$) is the projection of a computation of $st$ that we call $e1$ ($e2$). If $e1 = e2$ then by definition of a projection $e1_w = e2_w$.

If $e1 \neq e2$ then it exists $e$ such that $e1 = [e(c1, CH1, c3)e12]$ and $e2 = [e(c2, CH2, c4)e22]$ where $(c1, CH1, c3) \neq (c2, CH2, c4)$. According to the computation definition, $c1 = c2$. According to the strategy definition, $CH1 = CH2$. Thus $c3 \neq c4$, as $S$ is deterministic, we have $c3_W \neq c4_W$. Thus $e1_W = [e_W(c_W, CH_W, c3_W)e12_W]$ and $e2_W = [e_W(c_W, CH_W, c4_W)e22_W]$ where $c3_W \neq c4_W$. □

**Lemma 4.5** *Let* $f_W$ *be the function defined by: if* $e_W \in st_W$ *and if* $e_W = [e1_W(c_W, CH_W, c2_W)$ $e2_W]$ *then* $f_W(e1_W) = CH_W$. $f_W$ *is a function of* Choice

**Proof:** $f_W$ is effectively a *Choice* function: (1)$f_W$ is a function from $\mathcal{PAR}\_\mathcal{E}_W$ to $Proc_s$; (2) the lemma 4.4 proves that $f_W$ is a function; we have (3) $\forall e \in \mathcal{PAR}\_\mathcal{E}_W :: f(e) \in Enabled(\text{last}(e))$ and (4) $f(e) \neq \oslash$. □

**Lemma 4.6** *Let* $st1_W = (c_W, f_W)$ *be a strategy of* $W$ *where (1)* $c_W$ *is the projection on* $W$ *of the initial configuration of* $st$; *and (2)* $f_W$ *is the* Choice *function defined by if* $e_W \in st_W$ *and if* $e_W = [e1_W(c_W, CH_W, c2_W)e2_W]$ *then* $f_W(e1_W) = CH_W$. *Then, we have* $st1_W = st_W$ *and* $st1_W$ *is a strategy.*

**Proof:** According to the lemma 4.5, $st1_W$ is a strategy of $W$. By construction, all computations of $st_W$ belongs to $st1_W$.

We prove that all computations of $st_W$ belongs to $st1_W$ by contradiction. Let $e_W \in st1_W$ and $e_W \notin st_W$. It exists $e1_W$ such that (1) $e_W = [e1_W(c1_W, CH1_W, c3_W)e3_W]$, (2) $e1_W$ is the projection of a prefix of $e$, a computation of $W \diamondsuit S$, and (3) $e1_W(c1_W, CH1_W, c3_W)$ is not the projection of any prefix of any computation of $st$.

The projection of $e$ on $W$ is $e1_W(c2_W, CH2_W, c4_W)e4_W$. We have $c1_W = c2_W$ and $CH1_W = CH2_W = f_W(e1_W)$. Call $e1'$ the prefix of $e$ such that (1) its projection on $W$ is $e1_W$, (2) $e1'(c2, CH2,\ c4)$ is a prefix of $e$, and (3) $e1_W(c2_W, CH2_W, c4_W)$ is the projection of $e1'(c2, CH2,$ $c4)$ on $W$. There is a configuration $c3$ ($c3 = c3_W c4_S$) such that (1) $c3 \in < c2 : CH2 >$, and (2) the projection of $c3$ on $W$ is $c3_W$. By the observation 2.1, $e1(c2, CH2, c3)$ is the prefix of a computation of $st$. Thus $e1_W(c1_W, CH1_W, c3_W)$ is the projection of a prefix of a $st$'s computation. □

**Theorem 4.3** *Let $S$ be a deterministic and $k$-fair protocol. Let $W$ be a randomized protocol. Let $st$ be a strategy of $W \diamondsuit S$. We name $st_W$ the projection of $st$ on $W$. Let $A_W$ be a predicate on computations of $W$. $st_W$ is a strategy of $W$ and we have $P_{st}(\{e \in st :: e_W \vdash A_W\}) = P_{st_W}(\{e' \in st_W :: e' \vdash A_W\})$.*

**Proof:** The lemma 4.6 proves that $st_W$ is a strategy of $W$. Let $C_{h_W}$ be a cone of $st_W$. As $S$ is deterministic, $h_W$ is the projection of only one computation of $st$, called $h$. Therefore, $C_{h_W}$ is the projection of only one cone of $st$ called $C_h$. And, we have $P_{st}(C_h) = P_{st_W}(C_{h_W})$. By extension, we have $P_{st}(\{e \in st :: e_W \vdash A\}) = P_{st_W}(\{e' \in st_W :: e' \vdash A\})$. □

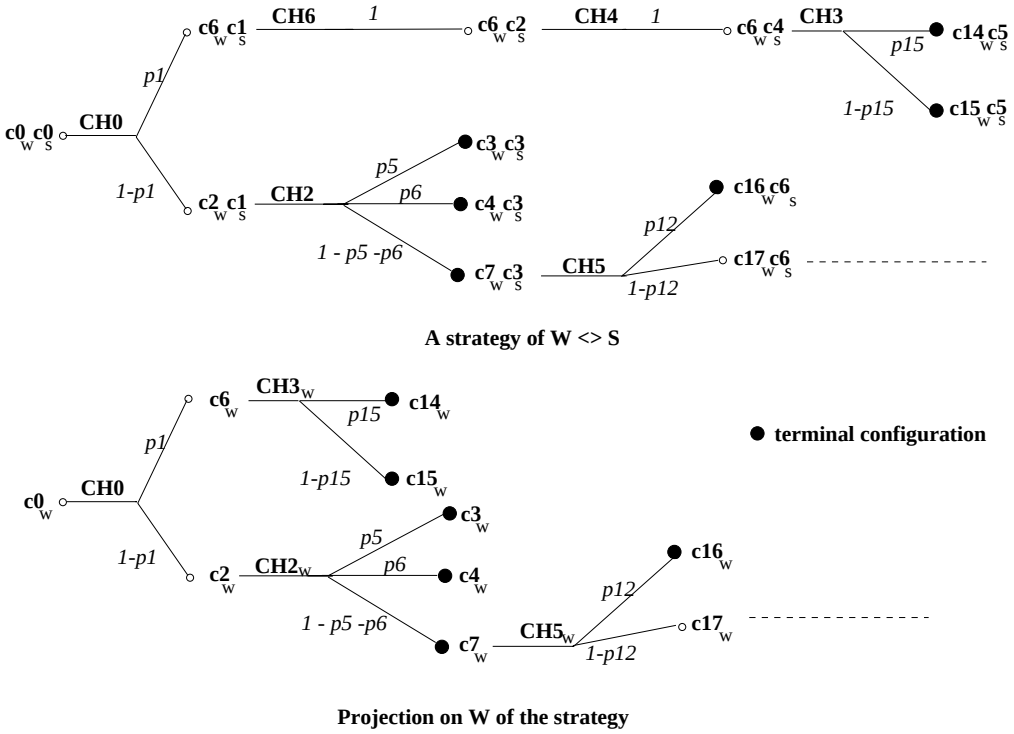


Figure 2: **The projection of a strategy of W ◇ S on W (S being deterministic and $k$-fair)**

**Theorem 4.4** *Let $S$ be a deterministic and $k$-fair protocol. Let $W$ be a probabilistic protocol self-stabilizing for the specification $SP$ under a $k$-bounded scheduler. $W \diamondsuit S$ is self-stabilizing for the specification $SP$ under any scheduler.*

**Proof:** Let $st$ be a strategy of $W \diamondsuit S$. Let $st_W$ be the projection of this strategy on $W$. $st_W$ is a strategy of $W$ (see theorem 4.3). As $S$ is $k$-fair, the computations of $st_W$ are $k$-bounded (lemma 4.3). The probability of the set of computations of $st_W$ that will eventually satisfy $SP$ is 1. According to theorem 4.3, the probability of the set of computations of $st$ that will eventually satisfy $SP$ is 1. □

The cross-over composition is used in this paper to transform a protocol self-stabilizing under a quite particular scheduler (the $k$-bounded scheduler) into a protocol self-stabilizing under an

arbitrary scheduler. This technique can be also used to transform a protocol self-stabilizing under a central scheduler into a protocol self-stabilizing under an arbitrary scheduler. We can imagine other applications for our technique in the field of scheduler management but the result depends always on the properties of the stronger protocol.

# 5 Randomized self-stabilizing token circulation and leader election

In section 5.1, we present a protocol that is (*N-1*)-fair ($N$ being the size of the ring) on anonymous, unidirectional rings. In section 5.2, we present a space optimal randomized token circulation protocol under any scheduler on anonymous and unidirectional rings. The protocol is obtained by cross-over composition. Finally, in section 5.3, a space optimal randomized self-stabilizing leader election protocol under any scheduler for anonymous and unidirectional rings of any size is presented.
The space complexity of our token circulation and leader election protocols are $O(\lg m_N)$ bits per processor where $m_N$ is the smallest integer not dividing $N$ ($N$ being the ring size). Notice that the value of $m_N$ is constant on average. For example, on odd size rings, 4 (2) bits per processor are necessary and sufficient for leader election (token circulation). The optimality of our protocols was proven in [5].

These protocols are self-stabilizing under any scheduler. There is no restriction on the scheduler except that it has to choose enabled processors. But the scheduler may be unfair by avoiding to choose some specific processors.

## 5.1 Deterministic token circulation

**Remark 5.1** *All operations are made modulo $m_N$.*

**Notation 5.1** *We said that a processor has a token if and only if it satisfies the Deterministic_token predicate (defined in the protocol 5.1).*

**Lemma 5.1** *In a ring, there is always a token.*

**Proof:** Assume there is a terminal configuration. Call $c$ such a configuration where no processor has a token. Let $p_0, p_1, p_2 \dots p_{N-1}$ be the processors of the ring. On $c$, $\forall i \in [0, N-1]$ $dt_{i+1} = (dt_i + 1) \bmod m_N$ On $c$, $dt_{N-1} = (dt_0 + N - 1) \bmod m_N$ or $dt_0 = (dt_{N-1} + 1) \bmod m_N = (dt_0 + N) \bmod m_N$. It is not possible because $N \bmod m_N \neq 0$. □

When a processor performs an action, it loses its token. It will perform again an action only after receiving a token (after that its left neighbor has performed an action).

**Corollary 5.1** *The protocol DTC is without termination under any type of scheduler.*

**Notation 5.2** *The distance between two processors $i$ and $j$ is denoted by dist(i, j). We have $dist(j, i) = N - dist(i, j)$.*

**Theorem 5.1** *The protocol DTC is (*N-1*)-fair.*

**Protocol 5.1** Deterministic token circulation on anonymous and unidirectional rings: DTC

**Field variables on** $p$:

$dt_p$ is a variable taking value in [0, $m_N$ - 1]. (*the variable represents the deterministic token*)

**Predicate**:

$Deterministic\_token(p) \equiv dt_p - dt_{lp} \neq 1 \bmod m_N$

**Macro**:

$Pass\_Deterministic\_token(p) : dt_p := (dt_{lp} + 1) \bmod m_N$

**Action on** $p$:

$\mathcal{A}$:: $Deterministic\_token(p) \longrightarrow Pass\_Deterministic\_token(p)$

**Proof:** Let $p$ be a processor. We name $p_k$ the processor such that $dist(p, p_k) = k$. We prove by induction that the processor $p_k$ ($k \in [1, N-1]$) will perform at most $k$ actions before that $p$ performs an action.

**Basic step.** The processor $p_1$ may perform only one action. After losing its token, it cannot perform any action before an action of $p$.

**Induction step.** Suppose that the processor $p_{k-1}$ performs at most $k-1$ actions before an action of $p$. $p_k$ will get at most $k-1$ tokens from $p_{k-1}$. The processor $p_k$ will perform at most $k$ actions ($k-1$ actions, if it gets $k-1$ tokens; plus one action, if it has initially a token).

Between two actions of $p$, another processor can perform at most *N-1* actions. Therefore $p$ performs an infinite of actions along a maximal computation (each maximal computation is infinite). □

## 5.2 Token circulation under an arbitrary scheduler

The randomized token circulation protocol (CTC) presented by Beauquier, Cordier and Delaët in [4] is self-stabilizing under a $k$-bounded scheduler on unidirectional and anonymous rings. We compose this protocol with $DTC$, the obtained protocol is self-stabilizing token circulation protocol under an arbitrary scheduler.

A processor is privileged, if its verifies the *privilege* predicate (defined in the protocol. 5.2). A round for a privilege in the protocol $CTC$ is a fragment computation starting in a configuration $c$ and ending in a configuration $c'$, having the following properties: (1) in $c$ and $c'$ , only $p$ holds a privilege; and (2) in the fragment each processors holds the privilege one and only one time. Let $SP_{ME}$ be the following predicate over computations: "in each configuration, there is only one privileged processor and the computation contains an infinity of rounds". Let $L_{ME}$ be the following predicate over configurations: " there is only one privileged processor in the system". A legitimate configuration for the protocol $CTC$ is a configuration which satisfies the predicate $L_{ME}$.

**Lemma 5.2** *In the protocol CTC under any scheduler the predicate $L_{ME}$ is closed.*

**Proof:** During a computation step, either the processor holding the privilege does not pass the privilege, hence it stays the only privileged processor. Or this processor passes its privilege; then its neighbor becomes the new privileged processor. □

**Protocol 5.2** token circulation on anonymous and unidirectional rings: CTC

**Field variables on** $p$:

$t_p$ is a variable taking value in [0, $m_N$ -1]. (*the variable represents the privilege*)

**Random Variables on** $p$:

$rand\_bool_p$ taking value in {1, 0}. Each value has a probability 1/2.

**Predicate**:

$Privilege(p) \equiv t_p - t_{lp} \neq 1 \bmod m_N$

**Macro**:

$Pass\_privilege(p) : t_p := (t_{lp} + 1) \bmod m_N$

**Action on** $p$:

$\mathcal{A}$:: $Privilege(p) \longrightarrow$ if $(rand\_bool_p = 0)$ then $Pass\_privilege(p)$;

For proving the convergence of the protocol $CTC$ we use the direct verification of the self-stabilization definition.

**Lemma 5.3** *Let $k$ be an integer. Let $st$ be a strategy of the protocol $CTC$ under a $k$-bounded scheduler. There exist $\epsilon_{st} > 0$ and $N_{st} \geq 1$ such that any cone of $st$ satisfies Local_Convergence(true, $L_{ME}$, $\delta_{st}$, $n_{st}$).*

**Proof:** Let $C_h$ be a cone of $st$. Assume that in last(h), there are several privileged processors. Let $p_1, p_2, \ldots p_m$ be the privileged processors in last(h). Let $d_1$ be the distance between $p_1$ and $p_2$.
We exhibit a history where (1) the privileged processors $(p_3, \ldots, p_m)$ stay privileged; and (2) other processors are not privileged (except may be $p_2$). At that point, at least one privilege has disappeared. By extension, we present a history, where all privileges except one are similarly removed.
Let $C_{hh'}$ be a subcone of $C_h$ where $h'$ verifies the following properties: (1) $h'$ ends with an action of $p_1$ where it passes its privilege; (2) $h'$ contains one action of $p_1$; and (3) no other processor has passed its privilege. As the scheduler is $k$-bounded, h' exists and other privileged processors perform at most k actions in h'. Each time that a privileged processor performs an action in $h'$ its $rand\_bool$ variable takes the value 1, when $p_1$ performs its action, its $rand\_bool$ variable takes the value 0. We have $|h'| \leq k(m-1)+1$ and $P_{st}(C_{hh'}) \geq P_{st}(C_h).\frac{1}{2}^{k(m-1)+1}$.
According to the definition of $h'$, in last(hh'), the right neighbor of $p_1$ (p') is privileged and $p_1$ is no more privileged (the privilege of $p_1$ has reached $p'$).
Step by step, we build a subcone of $C_{h_1}$, $C_{hh_1}$ where $h_1$ verifies the following properties: (1) $h_1$ ends with an action of the left neighbor of $p_2$ where it passes its privilege to $p_2$ (after this action, $p_2$ may or may not be privileged); (2) $p_2$, $p_3$, ..., $p_m$ have kept their privilege; (3) in last($hh_1$), the privilege of $p1$ has reached $p_2$. In last($h_1$), there are at most $m-1$ privileges. We have $|h_1| \leq d_1(k(m-1)+1) \leq kN^2 + N$, and
$P_{st}(C_{hh_1}) \geq P_{st}(C_h).\frac{1}{2}^{d_1(k(m-1)+1)} \geq P_{st}(C_h).\frac{1}{2}^{kN^2+N}$.
Finally, we build a subcone of $C_h$, $C_{hH}$ where in last($hH$), there is only one privileged processor. We have $|H| \leq (m-1)(kN^2+N) \leq kN^3 + N^2$, and $P_{st}(C_{hH}) \geq P_{st}(C_h).\frac{1}{2}^{(m-1)(kN^2+N)} \geq$

**Protocol 5.3** Randomized token circulation protocol under any scheduler: SSCTC

**Field variables on** $p$:

$dt_p$ is a variable taking value in [0, $m_N$ -1]. (*the variable represents the deterministic token*)

$t_p$ is a variable taking value in [0, $m_N$-1]. (*the variable represents the privilege*)

**Random Variables on** $p$:

$rand\_bool_p$ taking value in {1, 0}. Each value has a probability 1/2.

**Predicate**:

$Deterministic\_token(p) \equiv dt_p - dt_{lp} \neq 1 \bmod m_N$

$Privilege(p) \equiv t_p - t_{lp} \neq 1 \bmod m_N$

**Macro**:

$Pass\_Deterministic\_token(p) : dt_p := (dt_{lp} + 1) \bmod m_N$

$Pass\_privilege(p) : t_p := (t_{lp} + 1) \bmod m_N$

**Action on** $p$:

$\mathcal{A}_1$:: $Deterministic\_token(p) \wedge \neg Privilege(p) \longrightarrow$
$Pass\_Deterministic\_token(p)$

$\mathcal{A}_2$:: $Deterministic\_token(p) \wedge Privilege(p) \longrightarrow$
$Pass\_Deterministic\_token(p)$; if $(rand\_bool_p = 0)$ then $Pass\_Privilege(p)$

$P_{st}(C_h).\frac{1}{2}^{kN^3+N^2}$. $\epsilon_{st} = \frac{1}{2}^{kN^3+N^2}$, $N_{st} = kN^3 + N^2$. □

**Lemma 5.4** *Let st be a strategy of the protocol CTC. Let $\mathcal{C}_h$ be a cone of st so that in last(h), there is only one privilege. The probability of the subcone $\mathcal{C}_{hh1}$ of the cone $\mathcal{C}_h$ where $h1$ is a round and $|h1| = N$ is $P_{st}(\mathcal{C}_h).\frac{1}{2}^n$.*

**Proof:** Let $p$ be the processor holding the privilege in the configuration last(h). There is a probability 1/2 that the privilege passes to the $p$'s right neighbor, in one computation step. The previous reasoning is repeated until the privilege goes back to $p$. Let $\mathcal{C}_{hh1}$ be the subcone of $\mathcal{C}_h$ where in last($hh1$), $p$ holds again the privilege after that all other processors have held the privilege in $h1$. The probability of this cone is $P(\mathcal{C}_{hh1}) = P(\mathcal{C}_h).\frac{1}{2}^N$ and $|h1| = N$.

**Lemma 5.5** *The protocol CTC is correct.*

**Proof:** Let $st$ be a strategy. Let $C_h$ be a cone of $st$ ending by a legitimate configuration. We call $\epsilon'_k$ the probability of the subcones of $C_h$, $C_{hh'}$, where $|h'| \leq kN$ and during the history $h'$ the privilege completes at least one round. Applying the lemma 5.4, we found that $\epsilon'_1 \geq P_{st}(C_h).(1 - (1 - \frac{1}{2}^N))$. Set $\epsilon_k = 1 - (1 - \frac{1}{2}^N)^k$. Suppose that $\epsilon'_k \geq P_{st}(C_h).\epsilon_k$. In the same way as the proof of theorem 3.5, we prove that $\epsilon'_{k+1} \geq P_{st}(C_h).(\epsilon_k + (1-\epsilon_k).\frac{1}{2}^N)$. Hence, $\epsilon'_{k+1} \geq P_{st}(C_h).\epsilon_{k+1}$. The probability of the set of computations of $C_h$ in which the privilege completes at least one round is 1. By induction, we prove that for any integer $n$, the probability of the set of computations of $C_h$ in which the privilege completes at least $n$ rounds is 1. □

**Theorem 5.2** *The protocol CTC is self-stabilizing for the specification $SP_{ME}$ under a k-bounded scheduler.*

**Proof:** The convergence is proven by lemma 5.3 and corollary 3.4. The correctness is proven by the lemma 5.5. □

The cross-over composition between the protocol $CTC$ and the protocol $DTC$ is called $SSCTC$ and presented in the protocol 5.3. $SSCTC = CTC \diamondsuit DTC$.

**Theorem 5.3** *The protocol SSCTC is self-stabilizing for the specification $SP_{ME}$ under an arbitrary scheduler.*

**Proof:** The protocol $SSCTC$ is the result of the cross-over composition between the protocol $CTC$ which is self-stabilizing under a (*N-1*)-bounded scheduler for the specification $SP_{ME}$ (theorem 5.2) and the protocol $DTC$ which is a (*N-1*)-fair protocol (theorem 5.1). Using the theorem 4.4, we prove that the protocol $SSCTC$ is self-stabilizing for the specification $SP_{ME}$ under an arbitrary scheduler. □

## 5.3 Leader election under an arbitrary scheduler

### 5.3.1 Leader election under a $k$-bounded scheduler

We present the randomized protocol $LE$ (see the protocol 5.4). We prove, using the attractor technique, that the protocol $LE$ is self-stabilizing for the leader election specification under a $k$-bounded scheduler.

**Notation 5.3** *A processor has a privilege, if it verifies the Privilege predicate. A processor is a leader, if it verifies the Leader_mark predicate. These predicates are defined in the protocol 5.4.*

The goal of the color is to freeze the leader when it is unique, but also to ensure the circulation of leader when the ring contains several ones. When a processor is privileged and leader, it randomly selects a color. During the circulation of the privilege this color will be communicated to every processor of the ring ($\mathcal{A}3$). The leader waits until becoming privileged again. At that time, if the color of its left neighbor is the same as its color, then it stays leader and starts the checking again by randomly selecting a new color (action $\mathcal{A}_2$). In this case, it "assumes" that it is the only leader in the ring.

Since the color is randomly selected, when there are several leaders in the ring, a leader will eventually become privileged when its left neighbor does not have the right color (i.e. its color, called *co*). In this case, the leader passes its leadership (action $\mathcal{A}_1$) to its right neighbor. It "assumes" that several leaders coexist in the ring. The leadership do several moves in the rings up to catch the next leader. More precisely, the leadership moves until it reaches a processor that does not have the *co* color (usually this processor is a leader). As after an $\mathcal{A}1$ move, a processor takes the *co* colors; in all cases, the leadership will do at most $N$ moves.

Once the ring is stabilized, there is one frozen leader and one privilege that circulates. We prove that $LE$ is a self-stabilizing leader election protocol under a $k$-bounded scheduler.

***Protocol 5.4*** *Randomized leader election on anonymous and unidirectional rings: LE*

**Field variables on** $p$:

$lm_p$ is a variable taking value in [0, $m_N$ -1]. (*the variable represents the leader mark*)
$t_p$ is a variable taking value in [0, $m_N$-1]. (*the variable represents the privilege*)
$c_p$ is a boolean. (*0 = blue and 1 = green)*

**Random Variables on** $p$

*(The two independent variables, are the two fields of the unique processor random variable.)*
$rand\_bool_p$ *taking value in* {*1, 0*}. *Each value has a probability 1/2.*
$rand\_color_p$ *taking value in* {*blue, green*}. *Each value has a probability 1/2.*

**Predicate***:*

$Leader\_mark(p) \equiv lm_p - lm_{lp} \neq 1 \bmod m_N$
$Privilege(p) \equiv t_p - t_{lp} \neq 1 \bmod m_N$

**Macro***:*

$Pass\_Leader\_mark(p)$ *:* $lm_p := (lm_{lp} + 1) \bmod m_N$
$Pass\_privilege(p)$ *:* $t_p := (t_{lp} + 1) \bmod m_N$

**Action on** $p$*:*

$\mathcal{A}_1$*::* $Leader\_mark(p) \wedge (c_p \neq c_{lp}) \wedge Privilege(p) \longrightarrow$
*if* $(rand\_bool_p = 0)$ *then* $\{c_p := c_{lp}; Pass\_Leader\_mark(p); Pass\_privilege(p)\}$
$\mathcal{A}_2$*::* $Leader\_mark(p) \wedge (c_p = c_{lp}) \wedge Privilege(p) \longrightarrow$
*if* $(rand\_bool_p = 0)$ *then* $\{c_p := rand\_color_p; Pass\_privilege(p)\}$
$\mathcal{A}_3$*::* $\neg Leader\_mark(p) \wedge Privilege(p) \longrightarrow$
*if* $(rand\_bool_p = 0)$ *then* $\{c_p := c_{lp}; Pass\_privilege(p)\}$

**Theorem 5.4** *The predicate* $L_{ME}$ *is a probabilistic attractor of true on the protocol LE under a k-bounded scheduler.*

**Proof:** Similarly to the proof of the lemmas 5.3, we prove that the predicate $L_{ME}$ is a probabilistic attractor of *true* on the protocol $LE$ under a $k$-bounded scheduler. □

Once, a computation of $LE$ has reached a configuration satisfying $L_{ME}$; only one processor is enabled (the privileged one) at each computation step, whatever the computation performed. Thus the scheduler has no choice: it must select the enabled processor. After the next computation step, according to *rand_bool* value, either this processor still has the privilege or the privilege has moved to its right neighbor. In all cases, the scheduler has no choice. Therefore, all computations have the same pattern: a processor perform several actions till it has the privilege then its right neighbor gets the privilege and performs several actions, and so on. There is another pattern: "the privileged processor stays privileged forever"; but on any strategy, the probability of this pattern is zero.

Let $coherent\_color(p)$ be the following predicate over configurations: " $p$ verifies $coherent\_color(p)$ predicate if (1) all processors between $p$ and $q$ have the p's color ($q$ being the first privileged

processor at $p$ right), or (2) all processors between $p$ and the first leader to the $p$ right have the $p$'s color". Let *coherence*$(k)$ be the following predicate over configurations: " there are at least $k$ *coherent_color* processors in the ring".

The specification for the leader election problem is the following predicate $SP_{LE}$: "there is only one leader and its stays leader forever". Let $L_{LE}$ be the following predicate over configurations: "(1) *coherence*$(N)$ is verified, (2) there is only one leader, and (3) there is only one privileged processor". A legitimate configuration for the protocol $LE$ is a configuration satisfying the predicate $L_{LE}$.

**Observation 5.1** *Coherence*$(0)$ *is equal to the true predicate.*

The proof for self-stabilization of the protocol $LE$ will be made using the attractor technique. In fact, we prove that the predicate $L_{LE}$ is a probabilistic attractor for the predicate $L_{ME}$.

**Lemma 5.6** *For any processor p,* $coherent\_color(p)$ *is a closed predicate.*

**Proof:** Let $p$ be a *coherent_color* processor. We name $q$ the first privileged processor at $p$'s right. If there is a leader $q'$ on the path from $p$ to $q$ where $q' \neq q$ and $q' \neq p$ then after any action, $p$ still verifies the *coherent_color* predicate (only privileged processors may perform an action). Otherwise, all processors between $p$ and $q$ have the $p$'s color and are not leaders (i.e. the left neighbor of $q$ has the $p$'s color). After any action of a privileged processor other than $q$, $p$ still verifies the *coherent_color* predicate. We study the action of $q$. First case, $q$ is a leader that has the $p$'s color: to pass the privilege, it performs the action $\mathcal{A}2$, thus it stays leader. Second case, $q$ is a leader that does not have the $p$'s color: when it passes its privilege, it passes the leadership and takes $p$'s color (action $\mathcal{A}1$). Third case, $q$ is not a leader, after passing its privilege by the action $\mathcal{A}3$, $q$ gets the $p$'s color. In all cases, $p$ still verifies the predicate $coherent\_color(p)$, after the $q$'s action. □

**Corollary 5.2** *For all* $k \in [0, N]$, *coherence*$(k)$ *is a closed predicate.*

**Corollary 5.3** *On the protocol LE under any scheduler, the predicate* $L_{ME}$ *and* $L_{LE}$ *are closed (see the proof of lemma 5.2).*

**Lemma 5.7** *Let st be a strategy of the protocol LE under a k-bounded scheduler. Let i be an integer less than N. There exist* $\epsilon_{st} > 0$ *and* $N_{st} \geq 1$ *such that any cone of st satisfies Local_Convergence* $(L_{ME} \wedge coherence(i), L_{ME} \wedge coherence(i+1), \delta_{st}, n_{st})$.

**Proof:** Let $C_h$ be a cone of $st$ where last(h) $\vdash L_{ME}$. Assume that there are exactly $i$ processors that verify the predicate *coherent_color* on last$(h)$. We name $q$ the privileged processor. We name $p$ the first processor at $q$'s right that does not verify the *coherent_color* predicate.

We will build a subcone of $C_h$, $C_{hh'}$ such that at last$(hh')$, $p$ verifies the *coherent_color* predicate. As the *coherent_color* predicate is closed then last$(hh')$ verifies the *coherence*$(k+1)$ predicate.

We exhibit a scenario where (1) the *rand_bool* variable of all processors between $q$ and $p$ takes the value 0 (i.e they pass the privilege in one computation step); (2) the *rand_color* variable of all processors between $q$ and $p$ takes the value *blue*; and (3) the scenario ends with an action of $p$.

The leaders between $q$ and $p$ perform either the action $\mathcal{A}1$ or $\mathcal{A}2$, but in all cases, they pass the privilege in one computation step.
Thus, we build a subcone of $C_h, C_{hh'}$ where on last$(H1H)$, $p$'s right neighbor is privileged, Moreover, we have $|h'| \leq N$, and $P_{st}(C_{hh'}) \geq P_{st}(C_h).\frac{1}{2}^{2N}$. On last$(h)$, the privilege is held by the right neighbor of $p$. Thus $p$ is a *coherent_color* processor.

$N_{st} = N$ and $\epsilon_{st} = \frac{1}{2}^{2N}$. □

**Lemma 5.8** *Let st be a strategy of the protocol LE under a k-bounded scheduler. There exist* $\epsilon_{st} > 0$ *and* $N_{st} \geq 1$ *such that any cone of st satisfies Local_Convergence*$(L_{ME} \wedge coherence(N), L_{LE}, \delta_{st}, n_{st})$.

---

**Protocol 5.5** Randomized leader election protocol under any scheduler: SSLE

---

**Field variables on** $p$:

$dt_p$ is a variable taking value in [0, $m_N$-1]. (*the variable represents the deterministic token*)
$lm_p$ is a variable taking value in [0, $m_N$-1]. (*the variable represents the leader mark*)
$t_p$ is a variable taking value in [0, $m_N$-1]. (*the variable represents the privilege*)
$c_p$ is a boolean. (*0 = blue and 1 = green*)

**Random Variables on** $p$:

$rand\_bool_p$ taking value in {1, 0}. Each value has a probability 1/2.
$rand\_color_p$ taking value in {blue, green}. Each value has a probability 1/2.

**Predicate**:

$Deterministic\_token(p) \equiv dt_p - dt_{lp} \neq 1 \bmod m_N$
$Leader\_mark(p) \equiv lm_p - lm_{lp} \neq 1 \bmod m_N$
$Privilege(p) \equiv t_p - t_{lp} \neq 1 \bmod m_N$

**Macro**:

$Pass\_Deterministic\_token(p) : dt_p := (dt_{lp} + 1) \bmod m_N$
$Pass\_Leader\_mark(p) : lm_p := (lm_{lp} + 1) \bmod m_N$
$Pass\_privilege(p) : t_p := (t_{lp} + 1) \bmod m_N$

**Action on** $p$:

$\mathcal{B}_1$:: $Deterministic\_token(p) \wedge Leader\_mark(p) \wedge (c_p \neq c_{lp}) \wedge Privilege(p) \longrightarrow$
$Pass\_Deterministic\_token(p)$;
if $(rand\_bool_p = 0)$ then { $c_p := c_{lp}$; $Pass\_Leader\_mark(p)$; $Pass\_privilege(p)$}

$\mathcal{B}_2$:: $Deterministic\_token(p) \wedge Leader\_mark(p) \wedge (c_p = c_{lp}) \wedge Privilege(p) \longrightarrow$
$Pass\_Deterministic\_token(p)$;
if $(rand\_bool_p = 0)$ then {$c_p := rand\_color_p$; $Pass\_Privilege(p)$}

$\mathcal{B}_3$:: $Deterministic\_token(p) \wedge \neg Leader\_mark(p) \wedge Privilege(p) \longrightarrow$
$Pass\_Deterministic\_token(p)$;
if $(rand\_bool_p = 0)$ then { $c_p := c_{lp}$; $Pass\_Privilege(p)$ }

$\mathcal{B}_4$:: $Deterministic\_token(p) \wedge \neg Privilege(p) \longrightarrow Pass\_Deterministic\_token(p)$

---

**Proof:** Let $C_h$ be a cone of $st$ where last(h) $\vdash L_{ME} \wedge coherence(N)$. If there is one leader in last(h), then last(h) $\vdash L_{LE}$. Assume that there are several leaders in last(h). Let us denote by $q$ the privileged processor. We name $p$ the first leader at $q$'s left.

**1. on last(h), the privilege processor is not a leader.** The privilege reaches $p$, after that each processor between $q$ and $p$ passed the privilege in one computation step (i.e. the *rand_bool* variable of all processors between $q$ and $p$ takes the value 0) In the cone $C_h$, we exhibit the subcone, $C_{hh'}$, where in last($hh'$), $p$ is privileged; and, we have $|h'| \leq N$, $P_{st}(C_{hh'}) \geq P_{st}(C_h).\frac{1}{2}^N$.

We name H1 the history $hh'$.On last(H1), (1) the privileged leader has the same color as its left neighbor, or (2) the privileged leader has a different color from its left neighbor. In the last case, we call $H2$ the history $H1$.

**2. on last(H1), a leader is privileged, and has the same color as its neighbor.** We name $p$ the privileged leader and $s$ the next leader at $p$'s right. Let us study the scenerio where (1) $p$ performs the action $\mathcal{A}2$ and does not get the $s$' color; and (2) processors between $p$ and $s$ pass the privilege in one computation step. Thus, we build a subcone of $C_{H1}, C_{H1H'}$ where on last($H1H'$), $s$ is a privileged leader and the color of $s$ is not equal to the color of its left neighbor ($p$'s color). Moreover, we have $|H'| \leq N$, and $P_{st}(C_{H1H'}) \geq P_{st}(C_{H1}).\frac{1}{2}^{N+1}$. Now, we call $H2$ the history $H1H'$.

**3. On last(H2), a leader is privileged and has different color from its left neighbor.** We name $t$ the privileged leader, and $r$ the first leader at $t$ right. As $t$ verifies the *coherent_color* predicate, all processors between $t$ and $r$ have the $p$'s color. There is a scenario where the privilege and the leadership reach $r$, after that each processor between $p$ and $r$ passed the privilege and the leadership in one computation step by the action $\mathcal{A}1$ (their *rand_bool* variable has taken the value 0). The leadership that was on $t$ is now on $r$; In the cone $C_{H2}$, there is a subcone, $C_{H2H''}$, where $|H''| \leq N$ and $P_{st}(C_{H2H''}) \geq P_{st}(C_{H2}).\frac{1}{2}^N$. On last($H2H''$), thus there are less leaders than on last($H2$) (the leadership on $t$ is now amalgamated with the leadership in $s$).

By repeating the step 1, 2 and 3 at most $m-1$ times (where $m$ is the number of leaders in last($h$)); we get a configuration where there is one leader. There is a subcone of $C_h$, $C_{hH}$ where (1) there is only one leader, (2) $|H| \leq 3N^2$, and (3) $P_{st}(C_{hH}) \geq P_{st}(C_h).\frac{1}{2}^{N(3N+1)}$. □

**Theorem 5.5** *The protocol LE is self-stabilizing for the $SP_{LE}$ specification under a k-bounded scheduler.*

**Proof:**

- Convergence: In the protocol $LE$ under a $k$-bounded scheduler, the predicate $L_{ME}$ is a probabilistic attractor of $true$ (theorem 5.4). The predicate $L_{LE}$ is a probabilistic attractor for the predicate $L_{ME}$: the hypothesis of the corollary 3.3 are proven by the lemmas 5.7 and 5.8.
- Correctness: As the predicate $L_{LE}$ is closed, when $L_{LE}$ is verified on a configuration, there is only one leader. Whatever the computation performed, the leader stays the leader: the action $\mathcal{A}_1$ is never performed.

The condition of the theorem 3.4 are verified; The protocol $LE$ is self-stabilizing for specification $SP_{ME}$ under a $k$-bounded scheduler. □

### 5.3.2 Leader election under an arbitrary scheduler

Let $SSLE$ be the result of the cross-over composition between the protocols $LE$ and $DTC$ ($SSLE = LE \diamondsuit DTC$). The code of $SSLE$ is given in the protocol 5.5.

**Theorem 5.6** *The protocol $SSLE$ is stabilizing for the leader election specification under an arbitrary scheduler.*

**Proof:** The protocol $SSLE$ is a protocol obtained by the cross-over composition of $LE$ and $DTC$. The protocol $LE$ is self-stabilizing under a (*N-1*)-bounded scheduler (theorem 5.5) and the protocol $DTC$ is (*N-1*)-fair (theorem 5.1). The protocol $SSLE$ is self-stabilizing under an arbitrary scheduler according to the theorem 4.4. □

The randomized $SSLE$ protocol uses $2 \cdot m_N^3$ states per processor; each processor needs $O(\lg m_N)$ bits. Note that the value of $m_N$ is constant on average.

# 6 Conclusion

In this paper, we presented and proved a new randomized leader election protocol, that needs the minimal amount of space. Introducing explicitely the scheduler in the model allowed us to exhibit the notion of strategy, which is the key element to be equipped with a probability. It should be noticed that this model is quite general and does not depend neither on the ring structure nor on stabilization, and could be used for proving other randomized protocols under a non-deterministic scheduler. It should be also noticed that the probabilistic proof techniques could possibly apply to other types of protocols. For instance, the proof (informal) of Rabin's randomized Byzantine protocol (in [20]) uses the fact that at each round there is a positive probability to reach agreement (analogous to probabilistic convergence for self-stabilizing systems) and the fact that once reached, agreement persists (anologous to closure). We think that the tools that we presented allow to give a precise proof of Rabin's protocol, in which the behaviour of "Byzantine Generals" is made explicit in the notion of strategy.

We also introduced a new protocol composition, the cross-over product. We showed how the cross-over composition yields an automatic technique for transforming a protocol designed and proved for a fair scheduler, into an equivalent protocol for an unfair ones. This technique can be used as soon as a fair token circulation (albeit the scheduler is unfair) is available on the network structure. We gave such a fair token circulation on rings and we are presently working to extend it to general networks.

# 7 Acknowledgments

The authors thank P. Faith Fich for all the suggestions which clearly have improved our paper. We thank also Laurent Rosaz who has an important contribution in simplifying the presentation of the probabilistic framework.

## References


[1] E. Anagnostou and R. El-Yaniv. More on the power of random walks - uniform self-stabilizing randomized algorithms. In *WDAG91 Distributed Algorithms 5th International Workshop Proceedings, Springer-Verlag LNCS:579*, pages 31–51, 1991.

[2] D. Angluin. Local and global properties in networks of processors. In *12th Symposium on the Theory of Computing*, pages 82–93. ACM, 1980.

[3] B. Awerbuch and R. Ostrovsky. Memory-efficient and self-stabilizing network reset. In *PODC94 Proceedings of the Thirteenth Annual ACM Symposium on Principles of Distributed Computing*, pages 254–263, 1994.

[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 Proceedings of the Eighteenth Annual ACM Symposium on Principles of Distributed Computing*, pages 199–208, 1999.

[6] P. Billingsley. *Probability and Measure*. John Wiley & Sons, 1986.

[7] I. Christoff. Testing equivalences and fully abstract models for probabilistic processes. In *CONCUR'90 First International Conference on Concurrency Theory, Springer-Verlag LNCS:458*, pages 126–140, 1990.

[8] S. Dolev and T. Herman. Parallel composition of stabilizing algorithms. In *Proceedings of the fourth Workshop on Self-Stabilizing Systems*, pages 25–32, 1999.

[9] S. Dolev, A. Israeli, and S. Moran. Self-stabilization of dynamic systems assuming only read/write atomicity. *Distributed Computing*, 7:3–16, 1993.

[10] S. Dolev, A. Israeli, and S. Moran. Analyzing expected time by scheduler-luck games. *IEEE Transactions on Software Engineering*, 21:429–439, 1995.

[11] J. Durand-Lose. Random uniform self-stabilizing mutual exclusion. In *Distributed computing : OPODIS'98*, pages 89–97. Hermes, 1998.

[12] M.G. Gouda and T. Herman. Adaptive programming. *IEEE Transactions on Software Engineering*, 17:911–921, 1991.

[13] M.G. Gouda and N. Multari. Stabilizing communication protocols. *IEEE Transactions on Computers*, 40:448–458, 1991.

[14] 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.

[15] G. Itkis and L. Levin. Fast and lean self-stabilizing asynchronous protocols. In *FOCS94 Proceedings of the 34th Annual IEEE Symposium on Foundations of Computer Science*, pages 226–239, 1994.

[16] H. Kakugawa and M. Yamashita. Uniform and self-stabilizing token rings allowing unfair daemon. *IEEE Transactions on Parallel and Distributed Systems*, 8:154–162, 1997.

[17] A. Mayer, Y. Ofek, R. Ostrovsky, and M. Yung. Self-stabilizing symmetry breaking in constant-space. In *STOC92 Proceedings of the 24th Annual ACM Symposium on Theory of Computing*, pages 667–678, 1992.

[18] A. Mayer, R. Ostrovsky, and M. Yung. Self-stabilizing algorithms for synchronous unidirectional rings. In *SODA96 Proceedings of the Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 564–573, 1996.

[19] A. Pogosyants and R. Segala. Formal verification of timed properties of randomized distributed algorithms. In *PODC95 Proceedings of the Fourteenth Annual ACM Symposium on Principles of Distributed Computing*, pages 174–183, 1995.

[20] M. O. Rabin. Randomized byzantine generals. In *FOCS84 Proceedings of the 24st Annual IEEE Symposium on Foundations of Computer Science*, pages 403–409, 1984.

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

[22] R. Segala and N. Lynch. Probabilistic simulations for probabilistic processes. In *CONCUR'94 Fifth International Conference Concurrency theory, Springer-Verlag LNCS:836*, pages 481–496, 1994.

[23] R.J. van Glabbeek, S.A. Smolka, B. Steffen, and C.M.N. Toft. Reactive, generative and stratified models of probabilistic processes. In *Proceedings 5th Annual Symposium on Logic in Computer Science*, Philadelphia, USA, 1990.

[24] G Varghese. Compositional proofs of self-stabilizing protocols. In *Proceedings of the Third Workshop on Self-Stabilizing Systems*, pages 80–94. Carleton University Press, 1997.

[25] S.H. Wu, S. A. Smolka, and E.W. Stark. Composition and behaviors of probabilistic i/o automata. In *CONCUR'94 Fifth International Conference Concurrency theory, Springer-Verlag LNCS:836*, pages 513–528, 1994.