Analyze of randomized self-stabilizing algorithms under non-deterministic scheduler classes

Joffroy Beauquier, Colette Johnen

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

abstract We present a formal model for randomized distributed algorithms that includes the scheduler (also called adversary or demon).
Through an example related to stabilization, we show that a formal proof that does not use a formal definition of a scheduler is pointless. As a matter of fact, we show that the same algorithm, according to different schedulers, can be either correct or incorrect and in the cases where it is correct, can have different complexities.
The paper is an attempt to better understand what proving a randomized self-stabilizing algorithm in a non-deterministic environmentmeans.

key words: randomized algorithms, distributed algorithm, self-stabilizing system, scheduler, analyze of randomized algorithms.

résumé Nous présentons un cadre formel pour analyser les algorithmes répartis et probabilistes. La spécificité de ce modèle est la définition formelle et simple de la notion d'ordonnanceur (aussi appelé démon).
Via un exemple d'algorithme auto-stabilisant, nous montrons qu'une preuve dans un cadre où la notion d'ordonnanceur n'est pas clairement définis est sans intérêt. Un même algorithme, selon l'ordonnancement des processeurs, peut fonctionner correctement ou non. La complexité en temps peut varier grandement (par exemple, de borné à non borné) selon l'ordonnancement des processeurs.
Un des objectif de cette étude est d'apporter un éclairage nouveau sur l'analyse des algorithmes probabilistes dans un environnement non-déterministe.

mots clés: algorithmes probabilistes, algorithme répartis, système auto-stabilisant, ordonnanceur, analyse d'algorithmes probabilistes.

1 Introduction

Perhaps the main reason for the difficulty of understanding the behavior of distributed programs is their inherent non-determinism. Distributed systems are loosely coupled in the sense that the relative speeds of their local activity is usually not known in advance and in most cases totally unpredictable. Execution times and message delays may vary substantially for several repetitions of the same algorithm. But a distributed program has to be correct despite this uncertainty. In

other words, the proof of a distributed program must take into account the non-determinism created by the environment. That is the reason why the notion of adversary, also called demon or scheduler, has been introduced. The words adversary or demon refer to a hostile environment, that would try to disturb program executions, the idea being that to prove the correction of a program in the "worst" environmental situation, guarantees its validity in a less severe surrounding. Scheduler refers to the fact that independent events must appear in some given order. In the sequel, we will use equivalently demon or scheduler. Among the "tasks" devoted to the scheduler appear the choice of the speeds of processes, the delays for message transmission and the scheduling of tasks. For instance, if several processes are able to take a step, the scheduler is the entity that chooses which ones will be activated. In the context of fault tolerance, the scheduler chooses which process will fail and at what time, or which messages will be lost or delayed. Several impossibility results like in [11] or [1] use in their proofs such schedulers. As we will show it in this paper, the correction of a distributed program is meaningless if the external non-determinism is not considered. As a matter of fact some published algorithms appeared later to be incorrect for having underestimated the potential behavior of the environment. A consequence is that if the program is proved to be correct in an abstract model, the scheduler has to be part of this model. A way to deal with an unpredictable environment is to make it more predictable. That is the case when it is assumed that the scheduler acts probabilistically [5]. For instance when several processes can take a step simultaneously, there is some fixed probability for each one to take effectively the step. Or each time a message is sent, there is a small probability that it becomes lost. Some could argue that such a scheduler model is sufficient for most practical situations and that in real life environment follows statistical laws, but we think that, for several reasons, such a point of view is unsatisfactory.
The first reason is that a probabilistic scheduler cannot be invoked for proving that a deterministic program satisfies a deterministic specification. Introducing a probabilistic scheduler allows only to check a probabilistic specification derived from the original one. Take the consensus problem and one of its specifications given in [14]. Suppose that each message can be lost with some probability. Bracha and Toueg developed a solution under this assumption but what they solved in [5] is a probabilistic specification of the asynchronous consensus, not the original specification (which is impossible to solve). This remark is quite general and involves that all that can be proved with a probabilistic scheduler are programs solving probabilistic specifications.
The second reason is purely theoretical. A probabilistic scheduler is a very poor adversary, that can be deceived so easily. Some program are able to resist to very powerful adversaries and it is a challenge to always get a result as strong as possible.
The third reason is that we think that, in real life, some schedulers are not probabilistic at all, in the sense that the coming out of events does not obey simple independent probabilistic laws. The point is obvious if malicious adversaries are considered. A software error will always cause the same failure each time the bad portion of code is executed. and, if at some point of an execution, some messages become very slow, the chance that following messages be also very slow will increase. In fact that some bad events are correlated, rather than follow independent probability laws is expressed in [14] : "However we should keep in mind that this assumption (the bounded number of failures) is somewhat problematic : in most practical situations, if the number of failures is already large, then it is likely that more failures will occur."

For all these reasons, we present in this paper a model of distributed programs and non-deterministic schedulers. The model is quite general and can take into account whatever scheduler you could think of. In particular, probabilistic schedulers appear as special cases of the general notion. In the model, the scheduler can also be responsible for the granularity of the atomic events. Then it covers the classical schedulers, like the centralized or distributed schedulers or the read/write atomicity

introduced in the self-stabilization framework ([7], [6], [12], [13], and [8]). Our first aim is to show how this model can be used for proving the correctness or the incorrectness of distributed programs according to a given specification. In particular, as already mentioned before, we want to show that a same program can be correct or incorrect for the same specification, according to the choice of the scheduler. Our second aim is to demonstrate how the presented framework allows a precise computation of complexity measures in term of expectation. For the sake of clarity, we chose a very simple example, with a probabilistic specification, to make clear that a powerful non-deterministic scheduler can indefinitely postpone the realization of the specification, while a feeble one cannot. This example is closely related to a self-stabilization problem, namely the token circulation [7], and can be considered as a typical example of how to prove and compute the complexity of self-stabilizing distributed algorithms. The example is a random walk of two tokens on an unidirectional ring. In the initial configuration, there are two tokens on the ring, held by two different processes. There is no assumption on the processes that initially hold the tokens. Only a process holding a token can take a step. A step consists in tossing a coin (probability 1/2 for head and tail) and if head to transmit the token. For taking a step, a process must be chosen by the scheduler. The scheduler must choose processes among those holding a token. Finally, the specification is that one of the initial tokens catches up with the other one with probability one. To the question whether the above system meets the specification, most people would answer positively. The idea is that if one of the token does not move because it always "draw" tail and if the other always (or at least a number of time at most the size of the ring) "draw" head, the second token will catch up with the first. Because this scenario can appear at any time with a fixed probability, we are done. As you have noticed, no scheduler appears in this reasoning, and it is why it is incorrect. The sequel (we hope) will make this point clearer.

Related works. In [9], Dolev, Israeli and Moran introduced the idea of a two players game between the scheduler and what they call luck, i.e. the random values, without defining formally the probabilistic space of computations The structure (informally presented) behind a sl-game is a strategy (formally defined in this paper) where some branches have being cut. In [18], [15], and [16], Lynch, Pogosyants and Segala present a formal method for analyzing probabilistic I/O automata which modelize distributed systems. A clear distinction between the protocol, which is probabilistic, and the scheduler, which is non-deterministic, is made. The notion of cone, that is at the basis of the probabilistic space, is also used. These works do not consider self-stabilization. In these two approaches, the analyze of a probabilistic algorithm under a class of schedulers is not defined. Mainly because the notion of scheduler is not presented.
On the other hand there are numerous proofs of self-stabilizing algorithms under some particular schedulers.

2 Model

Abstract model. A non deterministic distributed system is represented in the abstract model of *transition systems*. A *distributed system* is a tuple $DS = (\mathcal{C}, T, \mathcal{I})$ where $\mathcal{C}$ is the set of all system configurations; T is a transition function of $\mathcal{C}$ to the set of $\mathcal{C}$ subsets; and $\mathcal{I}$ 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. A *computation step* is a pair of configurations (c_i, c_j) where c_j is an output of a transition starting to c_i. A *computation* e of DS is a sequence of consecutive computation steps $e = (c_0, c_1), (c_1, c_2) \ldots$ where $c_0 \in \mathcal{I}$.

Let c be an 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 $c \in \mathcal{I}$.

Interpretation. In fact, the distributed system is the collection of processes ($Proc$) computing protocol P. A protocol has a collection of variables (internal and/or field) and has a code part. A process communicates only with its neighbors (a subset of $Proc$). Communication among neighbors is carried out by field variables.
The state of a process is the collection of values of the process's variables (internal or field). A configuration of a distributed system is a vector of process states. A *local configuration* is the part of a configuration that can be "seen" by a process (i.e. its state and the field variables of its neighbors). 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 some probability. A process p is *enabled* at a configuration c, if an action guard of p is satisfied in c. The set of enabled processes for c is denoted by $Enabled(c)$.

Computation step versus transition. Let c be a configuration, and CH be a subset of enabled processes at c. We denote by $< c : CH >$ the set of configurations that are reachable from c after that the processes of CH have performed an action. A computation step has three elements: (1) an initial configuration: c, (2) a set of enabled processes: 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 processes. But in the case of randomized protocol, the final configuration depends on the output of each process action. 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 process involved in the computation step.
A computation is *maximal*, if the computation is either infinite, or finite and no process is enabled in the final configuration. In this case, the configuration is said to be terminal.

Strategy. Basically, a scheduler is intended to be an abstraction of the external non-determinism. Because the effect of the environment is unknown in advance, the scheduler notion must be able to formalize any external behavior. At the extremity, one should consider that the environment is possibly depending of a malicious adversary, trying to prevent the algorithm for performing its tasks. This powerful adversary should be assumed to have a global knowledge of the system, of its past, but also of its possible futures. At the contrary, it seems a valid model to assume that this adversary is unable to know in advance the result of a random experience. Defining a scheduler in some operational way - at that point of the computation the scheduler has such or such choice - raises the problem to define exactly in function of what the choice is made. If the choice depends of the actual configuration and of the history, the generality of the scheduler is restricted. How, for instance, express that the scheduler must be fair. That is the reason why we define a scheduler as a property (subset) on infinite runs. In our approach the key notion is the strategy.

Definition 2.1 *Let DS be a distributed system. Let T be a tree of DS. A strategy is a subtree of T where at a node, there is only one outgoing transition.*

Figure 1 presents a strategy. Formally, a *scheduler* is completely defined by the set of strategies which it may "produce". For instance, a pseudo-fair scheduler is the set of strategies st such that in st, the set of fair computations has probability 1 (we associate a probabilistic space to each strategy).

In the case of a deterministic protocol, a strategy is a maximal computation; and a scheduler is a subset of maximal computations. In the case of a randomized protocol, we will define a probabilistic space for every strategy of a scheduler.

3 Probabilistic model

In a randomized distributed system, each process has a random variable. The fundamental problem solved in this section is the definition of a probability measure 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.

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 [17].

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. 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. **last(h)** denotes the last configuration of h. The number of computation steps in the history h is denoted by $|h|$. Let st be a strategy. st is a particular cone with an empty history. This cone is denoted by $\mathcal{C}^{st}$.

Theorem 3.1 *Let st be a strategy. We note $\mathcal{F}_{st}$ the set of all finite unions of pairwise independent cones of the strategy st. $\mathcal{F}_{st}$ is a field.*

In a randomized protocol, each process has a random variable. The output of an action of a process 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 process. 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 3.1 *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 process p, val_p is a value of X_p, and c' is the obtained configuration after that all processes of CH have set their X_p variable to val_p and have performed an action.*

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

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

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

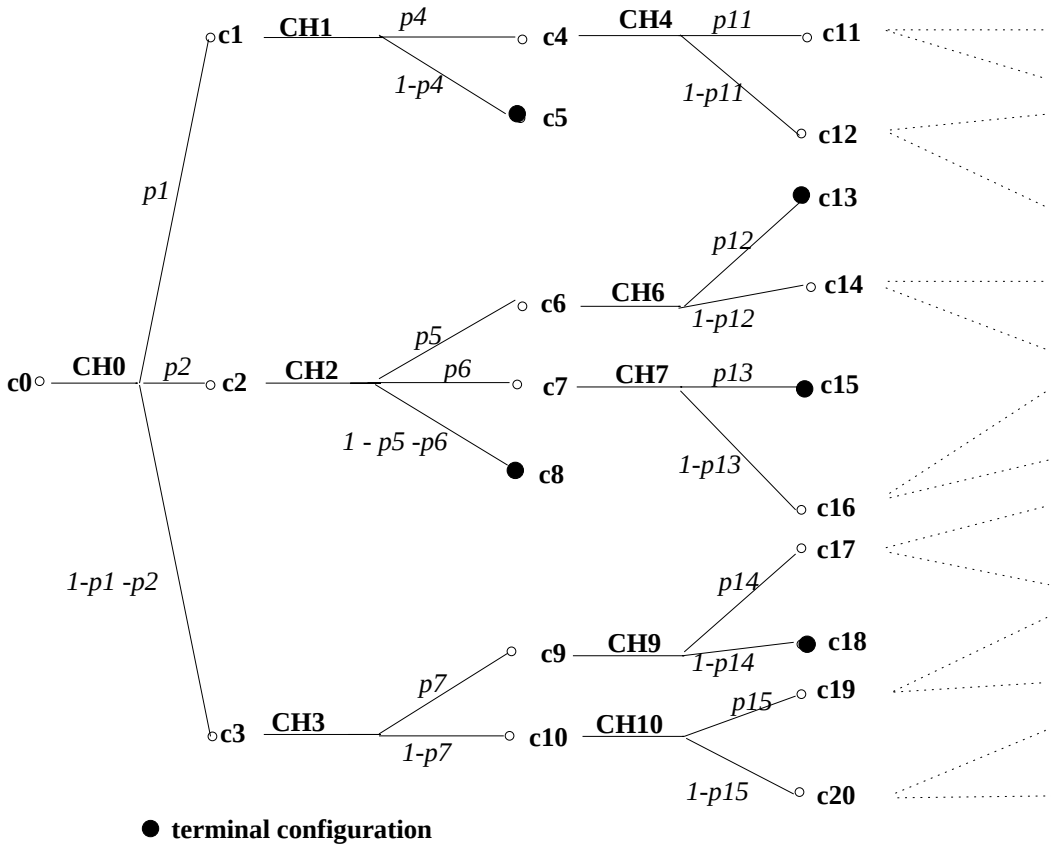

Figure 1: **A strategy with the probabilities of the computation steps**

Example 3.1 *See the figure 1,* $h1 = [(c_0, CH_0, c_1)(c_1, CH_1, c_5)]$, $P_{st}(C_{h1}) = p1.(1 - p4)$. $h2 = [(c_0, CH_0, c_2)(c_2, CH_2, c_7)(c_7, CH_7, c_{16})]$, $P_{st}(C_{h2}) = p2.p6.(1 - p13)$.

Let st be a strategy. The following properties are verified :

- $P_{st}(C^{st}) = 1$
- 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 $C_h = \bigcup_{1 \leq i \leq n} C_{hi}$. Then $P_{st}(C_h) = \sum_{1 \leq i \leq n} P_{st}(C_{hi})$.
- 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 $\sum_{1 \leq i \leq n} P_{st}(C_i^a) = \sum_{1 \leq i \leq m} P_{st}(C_i^b)$.

Definition 3.3 *Let st be a strategy. Let* P_{st} *be the function defined as follow:*

- $P_{st}(C_h) = \prod_{k=0}^{j} pr(c_k, CH_k, c_{k+1})$, *where* $h = [(c_0, CH_0, c_1)(c_1, CH_1, c_2) \ldots (c_j, CH_j, c_{j+1})]$.
- $P_{st}(A) = \sum_{i=1}^{n} P_{st}(C_i)$ *where* $A = \bigcup_{i=1}^{n} C_i$ *and the cones* C_i *are pairwise independent.*

P_{st} is a function, because the image of an element of $\mathcal{F}_{st}$ by P_{st} is unique. Let A be an element of $\mathcal{F}_{st}$; we have $P_{st}(A) + P_{st}(\overline{A}) = 1$.

Theorem 3.2 *Let st be a strategy. The function* P_{st} *is a probabilistic measure defined on the field* $\mathcal{F}_{st}$. *Let* $\sigma(\mathcal{F}_{st})$ *be the* σ*-field generated by* $\mathcal{F}_{st}$. *There is an 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 $\sigma(\mathcal{F}_{st})$ [4]. □

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 $\sigma(\mathcal{F}_{st})$-.

3.1 Self-Stabilization of a randomized protocol

In this section, we define the self-stabilization for randomized protocols with respect to the probabilistic model defined in the previous section. This section introduces also the probabilistic version of the attractor. The main idea behind these definitions is simple : To analyze a self-stabilizing algorithm under a scheduler, one has to analyze every strategy of the scheduler.

Notation 3.1 *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 3.1 *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 $\mathcal{EPR}_{st} = A$.*

Definition 3.4 (Probabilistic convergence) *Let L be a predicate defined on configurations. A probabilistic distributed protocol P under a scheduler D converges to L iff : In any strategy st of P under D the probability of the set of computations reaching a L is equal to 1. Formally, $\forall st, P_{st}(\mathcal{EL}_{st}) = 1$.*

Definition 3.5 (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: 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, $(\forall st, P_{st}(\mathcal{EL}_1) = 1) \Rightarrow (P_{st}(\mathcal{EL}_2) = 1)$.*

Definition 3.6 (Probabilistic Self-stabilization) *A randomized protocol P is self-stabilizing for a specification SP (predicate on the computations), if there exists a predicate on configuration L such that P converges to L and P verifies the following property*

- **correctness** *$\forall st$ of P under D, $\forall e \in st$, if $e \in \mathcal{EL}$ then e has a suffix that verifies SP.*

Definition 3.7 (Weak Probabilistic Self-stabilization) *A randomized protocol P is weakly self-stabilizing for a specification SP (predicate on the computations), if there exists a predicate on configuration L such that P converges to L and P verifies the following property*

- **probabilistic correctness** *$\forall st$ of P under D, $P_{st}(\{e \in \mathcal{EL}$ and e has a suffix that verifies $SP\}) = 1$.*

Observation 3.1 *We note $L12$ the following predicate on configurations $L1 \wedge L2$. If $L1 \triangleright_{prob} L2$ and if $L1$ is closed then $L1 \triangleright_{prob} L12$.*

In the case, of memoryless scheduler [10] (i.e. the choice of the subset of enabled processes only depends on the current configuration), each strategy is a Markov chain and can be analyzed using classical Markov properties as it is shown in [10]. Nevertheless, each strategy has to be analyzed. For instance, in some memoryless strategies of the protocol 4.1 the probability to converge to a legitimate configuration is equal to 1 and in some others this probability is 0.

3.2 Proving the convergence of protocols

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

Definition 3.8 *(Local convergence) Let st be a strategy. Let C_h be a cone in the strategy st. The cone C_h satisfies 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 there exist $\delta_{st} > 0$ and $n_{st} > 1$ such that any cone of st satisfies $Local\_Convergence$ ($PR1$, $PR2$, δ_{st}, n_{st}) then we say that st verifies the $Convergence$ ($PR1$, $PR2, \delta_{st}, n_{st}$) property or $Convergence$ ($PR1$, $PR2$) property.

3.3 Theorem 3.3

The following lemma is straightforward.

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.*

Theorem 3.3 *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: theorem 3.3 Let EL_k be the set of computations reaching a configuration satisfying $PR1$ and, after that, in at most $k.n_{st}$ steps reaching 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 3.1). $\mathcal{EPR}1 = \bigcup_{C \in M_1} C$ 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 hypotheses, $\overline{EL_{k-1}} \cap \mathcal{EPR}1$ is a countable union of pairwise independent cones. We call M_k the set of independent cones whose 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 computation 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 like 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 3.1 *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 3.2 *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 $\forall st, P_{st}(\mathcal{EPR}2) = 1$.*

Using a technique similar to the one presented in [9], one can prove that in a strategy st where the property $Convergence(PR1, PR2, \delta_{st}, n_{st})$ is satisfied the expectation time is bounded by $\frac{n_{st}}{\delta_{st}}$.

4 Example

Now we present a very simple protocol, that will allow us to exemplify all the previous notions : the randomized token circulation protocol (CTC) on unidirectional anonymous rings designed by Beauquier, Cordier and Delaët in [2]

For the sake of clarity we will not start the protocol from any configuration, but only from the configurations in which there are exactly two tokens in the ring, the distances between them being strictly greater than 1. The question that we are interested in, is to prove whether or not this (restricted) protocol has the following property : one of the tokens catches up with the other with probability 1 (that exactly means that the set of runs of the protocol in which one of the tokens catches up with the other has a probability 1). The correct answer is that the question is insufficiently specified and that satisfying or not the property depends on the scheduler. Thus, our aim will be to determine exactly where is the borderline between the class of schedulers that prevent the convergence and the class of those that do not. In the case of convergence, we want also have some information about the expectation time (in fact number of steps) before convergence. First, note that each of the above classes is non empty. Among the "bad" schedulers there is the following unfair scheduler, that can be informally described in the following way (but precisely defined using

our model). This bad scheduler chooses the process having a token until this token moves; then it chooses the process having the other token until this token moves and so one. If initially the tokens are at distance 2 or more, no one will ever catch up with the other. At the opposite it has been proved in [3] that the protocol converges (in fact from any initial configuration) with probability 1 with the k-bounded scheduler (k is an integer, see below). In the following sections, we will analyze this protocol under a large class of strategies to determine when this protocol converges and when it does not, when the expectation time is bounded and when it is not.

Protocol 4.1 token circulation on anonymous and unidirectional rings: CTC

Field variables on p:

v_p is a variable taking value in [0, m_N -1].

Random Variables on p:

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

Action on p:

$\mathcal{A}$:: $v_p - v_{lp} \neq 1 \bmod m_N \longrightarrow$ if $(rand\_bool_p = 0)$ then $v_p := (v_{lp} + 1) \bmod m_N$;

A process holds a token iff it is enabled (i.e. $v_p - v_{lp} \neq 1 \bmod m_N$) There is always a token in the ring. The number of tokens cannot increase whatever may happen. Let L be the following predicate over configurations: "there is only one token in the system". Once the convergence of the algorithm is proven to L, one can easily prove that the protocol is a weak self-stabilizing protocol for the specification : *one token fairly circulates in the ring.*

In an unidirectional ring, The *distance* from processor p to processor q is measured starting from p and moving to the right until q is reached. In particular, the distance from p to q is not the distance from q to p.

5 Rotating strategies

Because there are only two tokens (but a more general treatment would be feasible) a class of strategies can be described as what we call rotating strategies. Roughly speaking, in a rotating strategy the scheduler chooses to try to move one of the token tr_1 times, then the other tr_2 times, then the first token tr_3 times and so one. Because the scheduler is an adversary and tries to avoid the token merging, as soon as one of the token moves it deals with the other. Then, a rotating strategy is completely defined via an infinite sequence of positive integers $tr_1, tr_2, tr_3...$ where tr_i is the maximal duration (i.e. the maximal number of computation steps) of the ith turn. At each computation step of the 2i+1th (resp. 2i+2th) turn, only the process having the token $T1$ (resp $T2$) performs its action; thus a 2i+1th (resp. 2i+2th) turn is said to be a $T1$ (resp. $T2$) turn. The ith turn (a Tx turn) stops and the i+1th turn begins (i) after that the Tx token has moved or (ii) after tr_i consecutive actions of the process having the Tx token where the token does not move. A strategy of a k-bounded scheduler [3] is a rotating strategy in which the duration of each turn is inferior to $k + 1$.

Let st be a rotating strategy. We denote by ϵ_i the probability in st that a token catches up with the other token during the ith turn. A rotating strategy can be viewed as a game between the algorithm (called the player) and the scheduler. The game may be described by a series of independent random variables $X_1, X_2, X_3, ...$ such that $P_{st}(X_i = \omega) = \epsilon_i$. X_i represents the ith

trial of a random game (convergence game). The player (algorithm) wins when the two tokens merge (called the ω events). The player/algorithm may perform as many trials he wants. We have $P_{st}(\mathcal{EL}_{st}) = \sum_{i=1}^{\infty} \epsilon_i \times \prod_{j=1}^{i-1}(1-\epsilon_j)$.

5.1 Rotating strategies that do not converge

Theorem 5.1 *In the initial configuration c0, there are two tokens in the ring and the token distances are strictly greater than 1. Let st be a rotating strategy from c0 defined by the sequence of positive integers $tr_1, tr_2, tr_3...$ such that $\sum_{i=1}^{\infty} \frac{1}{2^{tr_i}} \leq \epsilon$. We have $P_{st}(\mathcal{EL}_{st}) \leq \epsilon$.*

Proof: Let i be an integer strictly greater than 1. ith turn is a Tx ($x = 1$ or $x = 2$) turn. If the token Ty ($y \neq x$) has moved during the i-1th turn then at the end of this turn the distance from Tx to Ty is strictly greater than 1. During the following turn, the Tx token cannot catch up with Ty. Thus, Tx catches up with Ty during the ith turn only if during the previous turn Ty has not moved. The probability in st that Ty does not move during the i-1th turn is $1/2^{tr_i-1}$. Therefore $\forall i > 1$, we have $\epsilon_i \leq 1/2^{tr_i-1}$. As initially, the distance from $T1$ to $T2$ is strictly greater than 1, $\epsilon_1 = 0$. $P_{st}(\mathcal{EL}_{st}) \leq \sum_{i=1}^{\infty} \frac{1}{2^{tr_i}} \leq \epsilon$. □

Notation 5.1 *We name $C_{1+\alpha}$ the value $\sum_{i=1}^{\infty}(\log_2(i+1))^{1+\alpha}$ (the sum $\sum_{i=1}^{\infty}(\log_2(i+1))^{1+\alpha}$ is bounded).*

Corollary 5.1 *In the initial configuration c0, there are two tokens in the ring and the token distances are strictly greater than 1. Let st be a rotating strategy from c0 defined by the sequence of positive integers $tr_1, tr_2, tr_3...$ such that $\forall i, tr_i > log_2(C_{1+\alpha}(i+1)^{1+\alpha})$ where α is a positive real. We have $P_{st}(\mathcal{EL}_{st}) < 1$.*

5.2 Rotating strategies that do converge

We will study a specific scenario $conv^{n-1}$ that drives the ring to a legitimate configuration. We will prove that the probability of this scenario is 1.

Definition 5.1 *We say that the token distances stay "quasi-invariable" between the turns j-1 and j+1 if their distances after the j+1th turn are the same as before the jth turn. A token may move during the jth turn; but in this case the other token must move during the next turn to ensure that the token distances go back to the initial values.*

The following scenario called $conv_m^1$ is defined as : after the m+1th turn the distance from Tx to Ty has decreased. (assuming that the mth turn is a Ty turn).

We define the scenario called $conv_m^{j+1}$ recursively. There exists k such that (i) the scenario $conv^j$ is realized at the end of the turn m-2k+1; (ii) the token distances stay "quasi-invariable" between the turns m-2k+1 and m-1; and (iii) after the m+1th turn, the distance from Tx to Ty has decreased.

Informally, there exist j integers $0 = k_1 < k_2 < k_3 < ... < k_j < m/2$ such that $\forall i \in [1, j]$ (i) the Tx token is at distance $d+i$ of Ty before the $m-2k_i$ turn; (ii) the distance from Tx to Ty is $d+i-1$ at the end of the $m-2k_i+1$ turn; and (iii) the distance from Tx to Ty stays quasi-invariable from the $m-2k_i+1$ turn to the $m-2k_{i+1}-1$ turn. Thus, after the m+1th turn, Tx is at distance d of Ty if it was at distance $d+j$ before the $m-2k_j$ turn.

Observation 5.1 *Let st be a rotating strategy. We denote by β_j the probability in st that the distance from Tx to Ty has decreased between the turn j-1 and j+1. We have $\beta_j = \frac{1}{2^{tr_j}} \times (1 - \frac{1}{2^{tr_{j+1}}})$ ($\frac{1}{2^{tr_j}} \geq \beta_j \geq \frac{1}{2^{tr_{j+1}}}$).*
We denote by $1 - \mu_j\beta_j$ the probability in st that the token distances stay quasi-invariable between the turns j-1 and j+1.

The probability of $conv_m^1$ in st (called $P_{st}(conv_m^1)$) is β_m. The probability of $conv_m^{j+1}$ in st (called $P_{st}(conv_m^{j+1})$) is $\beta_m \sum_{i=1}^{m/2} \times (P_{st}(conv_{m-2i}^j) \prod_{l=1}^{i-1}(1 - \mu_{m-2l}\beta_{m-2l}))$.

We will give in the following theorem a lower bound to the probability of the scenario $conv_m^{n-1}$ for a large class of rotating strategies. In any strategy, the probability of $conv_m^{n-1}$ is a lower bound of ϵ_{m+1} (the probability of tokens merging during the m+1th turn).

Theorem 5.2 *In the initial configuration c0, there are two tokens in the ring. The token distances are strictly greater than 1. Let k1 and k2 be two positive integers. Let j, K and M be positive integers. Let st be a rotating strategy from c0 defined by the sequence of positive integers $tr_1, tr_2, tr_3...$ such that $-K \leq tr_{i+1} - tr_i$. There exists $C_j > 0$ such that $\forall m \geq M' = M + 2j - 2, P_{st}(conv_m^j) \geq \beta_m \times C_j$ where $j \in [1, n-1]$.*

Proof: If $0 < tr_i - tr_{i+1}$ then we denote by K_i the difference $tr_i - tr_{i+1}$ ($K_i \leq K$). We have $\mu_i \leq \frac{2^{tr_{i+1}} + 2^{K_i} 2^{tr_{i+1}} - 2}{2^{tr_{i+1}} - 1} \leq \frac{2^{tr_{i+1}} + 2^{K_i} 2^{tr_{i+1}}}{2^{tr_{i+1}} - 1} \leq \frac{4.2^{K_i} 2^{tr_{i+1}}}{2^{tr_{i+1}}} \leq 4.2^{K_i} \leq 4.2^K$.

If $0 \geq tr_i - tr_{i+1}$ then we denote by K_i the difference $tr_{i+1} - tr_i$. We have $\mu_i \leq \frac{2^{tr_i} + 2^{K_i} 2^{tr_i} - 2}{2^{K_i} 2^{tr_i} - 1}$. $\mu_i \leq \frac{2(2^{K_i} 2^{tr_i} - 1)}{2^{K_i} 2^{tr_i} - 1} \leq 2$.

We denote by μ the value 4.2^K. We have $\forall i \geq M, \mu_i \leq 4.2^K = \mu$.
$\forall m \geq M' = M + 2j - 2, P_{st}(conv_m^j)/\beta_m \geq \sum_{i=1}^{(m-M'+2)/2} P_{st}(conv_{m-2i}^{j-1}) \prod_{l=1}^{i-1}(1 - \mu\beta_{m-21})$.

$\forall i \geq M$, we have $P_{st}(conv_i^1) \geq \beta_i$.

Let j be strictly greater than 1. Assume that $\exists C_{j-1}$ such that $\forall i \geq M' - 2, P_{st}(conv_i^{j-1}) \geq C_{j-1}\beta_i$.
We have $\mu P_{st}(conv_i^{j-1}) \geq C_{j-1} \times (1 - (1 - \mu\beta_i))$.
$P_{st}(conv_m^j) \times \mu / C_{j-1} \geq \beta_m \times (\sum_{i=1}^{(m-M'+2)/2} \prod_{l=1}^{i-1}(1 - \mu\beta_{m-2l}) - \prod_{l=1}^{i}(1 - \mu\beta_{m-2l}))$.
Therefore, $P_{st}(conv_m^j) \times \mu / C_{j-1} \geq \beta_m \times (1 - \prod_{l=1}^{(m-M'+2)/2}(1 - \mu\beta_{m-2l}))$.
$P_{st}(conv_m^j) \times \mu / C_{j-1} \geq \beta_m \times (1 - (1 - \mu\beta_{M'+2}))$ or $P_{st}(conv_m^{j+1}) \times \mu / C_{j-1} \geq \beta_m \times (1 - (1 - \mu\beta_{M'+3}))$.
$P_{st}(conv_m^j) \geq \beta_m \times C_j$ where $C_j = min(C_{j-1}\beta_{M'+2}, C_{j-1}\beta_{M'+3})$. □

The condition $tr_i - tr_{i+1} \geq K$ is needed to compute a lower bound for the probability of quasi-invariance of token distances between two turns.

Corollary 5.2 *In the initial configuration c0, there are two tokens in the ring. The token distances are strictly greater than 1. Let k1 and k2 be two positive integers. Let K and M be positive integers. Let st be a rotating strategy from c0 defined by the sequence of positive integers $tr_1, tr_2, tr_3...$ such that (i) $\forall i \geq M, tr_i \leq \log_2(k2(i + k1))$ and (ii) $-K \leq tr_{i+1} - tr_i$. We have $P_{st}(\mathcal{EL}_{st}) = 1$.*

Proof: According to theorem 5.2, there exists C such that $\forall i \geq M + 2n - 4 = M'', P_{st}(conv_i^{n-1}) \geq \beta_i \times C \geq \frac{C}{2 \times k2(i+k1)}$ (see observation 5.1). $P_{st}(\mathcal{EL}_{st}) \geq 1 - \prod_{l=M''}^{\infty}(1 - P_{st}(conv_l^{n-1}))$. We have $P_{st}(\mathcal{EL}_{st}) = 1$. □

Corollary 5.3 *In the initial configuration c0, there are two tokens in the ring. The token distances are strictly greater than 1. Let k be a positive integer. Let st be a rotating strategy from c0 defined by the sequence of positive integers $tr_1, tr_2, tr_3...$ such that $\forall i, tr_i \leq k$. We have $P_{st}(\mathcal{EL}_{st}) = 1$. The expectation time is inferior to $(2^{k+1})^{n-1}$.*

Proof: At any turn $m \geq 2n-2$, the probability of the token merging is greater that $\frac{1}{(2^{k+1})^{n-1}}$ in st. The expectation time is inferior to $(2n-2) + 2^{(k+1)\times(n-1)}$ (in term of turns). □

5.3 Expectation time of convergence

We compute now an upper bound of the expectation time for some specific strategies (i.e. the expected number of computation steps required by the algorithm to converge, in these strategies). For some other strategies, we will prove that the expectation time is unbounded.

Theorem 5.3 *In the initial configuration c0, there are two tokens in the ring. The token distances are strictly greater than 1. Let k1 be a positive integer. Let st be a rotating strategy from c0 defined by the sequence of positive integers $tr_1, tr_2, tr_3...$ such that $\forall k \geq 1, \forall i \in [2^{2k}-1, 2^{2k+2}-2]$ we have $tr_i = k + r$. The expectation time is bounded.*

Proof: $\exists C$ such that $\forall m \geq 2n-4, P_{st}(conv_m^{n-1}) \leq \frac{C}{2^{tr_m}}$ (see the theorem 5.2).
Let E be the value $\sum_{i=2n-4}^{\infty} \frac{C(i+1)}{2^{tr_i}} \prod_{j=2n-4}^{i-1}(1 - \frac{C}{2^{tr_j}})$. E is an upper bound of the expectation time because $\forall m < 2n-4, \epsilon_{m+1} \geq 0$ and $\forall m \geq 2n-4, \epsilon_{m+1} \geq P_{st}(conv_m^{n-1})$.
Let E' be the value $\sum_{i=1}^{\infty} \frac{C(i+1)}{2^{tr_i}} \prod_{j=1}^{i-1}(1 - \frac{C}{2^{tr_j}})$.
We have $E = E'/C2 - C1$ where $C1 = \sum_{i=1}^{2n-3} \frac{C(i+1)}{2^{tr_i}} \prod_{j=1}^{i-1}(1 - \frac{C}{2^{tr_j}})$ and $C2 = \prod_{j=1}^{2n-3}(1 - \frac{C}{2^{tr_j}})$.
We have $\frac{C(i+1)}{2^{tr_i}} = (i+1) \times (1 - (1 - \frac{C}{2^{tr_i}}))$; Therefore $E' = 1 + \sum_{i=1}^{\infty} \prod_{j=1}^{i-1}(1 - (C.2^{-tr_j}))$.
$E' = \sum_{k=1}^{\infty} E_k$ where $\forall k > 1$, $E_k = \sum_{i=2^{2k}-1}^{2^{2+2k}-2} \times \prod_{j=1}^{i-1}(1 - (C.2^{-tr_j}))$ and
$E_1 = \sum_{i=1}^{2^4-2} \times \prod_{j=1}^{i-1}(1 - (C.2^{-tr_j}))$.
We denote by θ_k the value $\frac{C}{2^k}$. We have $\forall i \in [2^{2k}-1, 2^{2k+2}-2]$, $C.2^{-tr_i} = \theta_k$.
For all $k > 1$, we denote by λ_k the value $(1-\theta_k)^{3.2^{2k}} = \prod_{j=2^{2k}-1}^{2^{2k+2}-2}(1 - (C.2^{-tr_j}))$.
We denote by λ_1 the value $(1-\theta_1)^{14} = \prod_{j=1}^{2^4-2}(1 - (C.2^{-tr_j}))$.
We denote by Λ_i the value $\prod_{j=1}^{i-1} \lambda_j = \prod_{j=1}^{2^{2i}-2}(1 - (C.2^{-tr_j}))$.
$E_k = \prod_{j=1}^{k-1} \lambda_j \times \frac{1-\lambda_k}{\theta_k} = (\Lambda_k \times \frac{1}{\theta_k}) - (\Lambda_{k+1} \times \frac{1}{\theta_k})$.
$E' = 1 + \frac{1}{\theta_1} + \sum_{k=2}^{\infty}(\frac{1}{\theta_k} - \frac{1}{\theta_{k-1}}) \times \Lambda_k = 1 + \frac{1}{\theta_1} + \sum_{k=2}^{\infty} \frac{\Lambda_k}{\theta_{k-1}} = 1 + \frac{1}{\theta_1} + \sum_{k=2}^{\infty} \prod_{j=1}^{k-1}(2.\lambda_j)$.
The series $\lambda_1, \lambda_2, ...$ is decreasing and there exists L such that $\forall k \geq L, \lambda_k \leq e^{-1}$.
We conclude that : $E' \leq 1 + \frac{1}{\theta_1} + \sum_{k=2}^{L-1} \frac{\Lambda_k}{\theta_{k-1}} + (\frac{\Lambda_L}{\theta_{L-1}} \sum_{k=0}^{\infty}(\frac{2}{e})^k)$.
Therefore, $E' \leq 1 + \frac{1}{\theta_1} + \sum_{k=2}^{L-1} \frac{\Lambda_k}{\theta_{k-1}} + (\frac{\Lambda_L \times e}{\theta_{L-1} \times (e-2)})$. Thus E is bounded. □

We will compute a lower bound on the expectation time (called E) for a specific class of rotating strategies. The strategies of this class are such that $\forall i \in [2^l - 1, 2^{l+1} - 2]$, $tr_i = l + k1$ where $k1$ is an integer ($tr_1, tr_2, tr_3...$ being the sequence of positive integers that defines the strategy).

If Tx catches up with Ty during the m+1 turns then Tx is at distance 1 of Ty before the mth turn. Initially Tx has at distance $d > 1$ of Ty. Tx is at distance 1 of Ty before the mth turn only if at some turn $(m - 2k + 1)$ the distance from Tx to Ty decreases to the value 1 and stay "quasi-invariant" until the turn $m-1$. Therefore, the probability that Tx catches up with Ty during the m+1th turn is inferior to $P_{st}(conv_m^2)$. First, we will compute an upper bound on $P_{st}(conv_m^2)$ for the strategies of the studied class.

Lemma 5.1 *In the initial configuration $c0$, there are two tokens in the ring. The token distances are strictly greater than 1. Let $k1$ be a positive integer. Let st be a rotating strategy from $c0$ defined by the sequence of positive integers $tr_1, tr_2, tr_3...$ such that $\forall i$ we have $tr_{i+1} \leq tr_i + 1$. We have $P_{st}(conv_m^2) \leq \frac{2}{3.2^{tr_m}}$.*

Proof: We have $\forall i, tr_{i+1} - tr_i = K_i \leq 1$. We have $\mu_i \geq 2$ if $K_i \leq 0$. If $K_i = 1$ then we have $\mu_i \geq \frac{3.2^{tr_i}-2}{2.2^{tr_i}-1} \geq \frac{3}{2}$. In st, $\forall i, \mu_i \geq 3/2 = \rho$.
$P_{st}(conv_m^2) \times \rho \leq \beta_m \times (\sum_{i=1}^{m/2} \rho\beta_{m-2i} \prod_{l=1}^{i-1}(1 - \rho\beta_{m-21}))$. We have $\rho\beta_i = 1 - (1 - \rho\beta_i)$.
$P_{st}(conv_m^2) \times \rho \leq \beta_m \times (\sum_{i=1}^{(m-M')/2} \prod_{l=1}^{i-1}(1 - \rho\beta_{m-2l}) - \prod_{l=1}^{i}(1 - \rho\beta_{m-2l}))$.
Therefore, $P_{st}(conv_m^2) \times \rho \leq \beta_m \times (1 - \prod_{l=1}^{(m-M')/2}(1 - \rho\beta_{m-2l})) \leq \beta_m \leq 2^{-tr_m}$. □

Theorem 5.4 *In the initial configuration $c0$, there are two tokens in the ring. The token distances are strictly greater than 1. Let st be a rotating strategy from $c0$ defined by the sequence of positive integers $tr_1, tr_2, tr_3...$ such that $\forall k \geq 1, \forall i \in [2^k - 1, 2^{k+1} - 2]$ we have $tr_i = k$. The expectation time is unbounded.*

Proof: Let E be the value $\sum_{i=1}^{\infty} \frac{2i+2}{3.2^{tr_i}} \prod_{j=1}^{i-1}(1 - \frac{2}{3.2^{tr_j}})$. E is a lower bound of the expectation time because $\epsilon_{m+1} \leq P_{st}(conv_m^2) \leq \frac{2}{3.2^{tr_m}}$ (we have $tr_{i+1} \leq tr_i + 1$).

We have $\frac{i+1}{3.2^{tr_i-1}} = (i+1) \times (1 - (1 - \frac{1}{3.2^{tr_i-1}}))$; Therefore $E = 1 + \sum_{i=1}^{\infty} \prod_{j=1}^{i-1}(1 - (2^{-tr_j+1}/3))$.
$E = \sum_{k=1}^{\infty} E_k$ where $E_k = \sum_{i=2^k-1}^{2^{1+k}-2} \times \prod_{j=1}^{i-1}(1 - 2^{-tr_j+1}/3)$.
We denote by θ_k the value $\frac{2}{3\times 2^k}$. We have $\forall i \in [2^k - 1, 2^{k+1} - 2]$, $\frac{2}{3\times 2^{-tr_i}} = \theta_k$.
We denote by λ_k the value $(1 - \theta_k)^{2^k} = \prod_{j=2^k-1}^{2^{k+1}-2}(1 - (2^{-tr_j+1}/3))$.
We denote by Λ_i the value $\prod_{j=1}^{i-1} \lambda_j = \prod_{j=1}^{2^i-2}(1 - (2^{-tr_j+1}/3))$.
$E_k = \prod_{j=1}^{k-1} \lambda_j \times \frac{1-\lambda_k}{\theta_k} = (\Lambda_k \times \frac{1}{\theta_k}) - (\Lambda_{k+1} \times \frac{1}{\theta_k})$.
$E = 1 + \frac{1}{\theta_1} + \sum_{k=2}^{\infty}(\frac{1}{\theta_k} - \frac{1}{\theta_{k-1}}) \times \Lambda_k$.
We conclude that : $E = 1 + \frac{1}{\theta_1} + \sum_{k=2}^{\infty} \frac{\Lambda_k}{\theta_{k-1}}$. Therefore, $E \geq \sum_{k=4}^{\infty} \frac{3}{2} \prod_{j=1}^{k-1}(2.\lambda_j)$.
The series $\lambda_1, \lambda_2, ...$ is increasing; thus $\forall j \geq 4, \lambda_j > \lambda_4 > \frac{1}{2}$.
We have $E \geq \sum_{i=4}^{\infty} \frac{3}{2} \prod_{j=1}^{3}(2.\lambda_j) = +\infty$. □

5.4 Summary of results

In the initial configuration $c0$, there are two tokens in the ring. The token distances are strictly greater than 1.

Let $k1$, $k2$, $k3$, $k4$ and M be positive integers. Let α be a positive real.

Let st be a rotating strategy from $c0$ defined by the sequence of positive integers $tr_1, tr_2, tr_3....$

The first table presents our results about the convergence in st according to tr_i values. We have defined the borderline between the rotating strategies that do converge and the other ones (i.e $tr_i = log_2(k1(i + k2))$).

Definition of rotating strategy	**Convergence in st?**
$\forall i \geq 1, tr_i > \log_2(C_{1+\alpha}(i+1)^{1+\alpha})$	no
$\forall i \geq M, tr_i \leq k1$	yes
$\forall i \geq M, tr_i \leq \log_2(k1(i+k2))$ and $tr_i - tr_{i+1} \geq -k3$	yes

The second table presents our results about the expectation time in st according to tr_i values (when the algorithm converges). We have not found the exact borderline between an unbounded and a bounded expectation time. The main difficulty is to compute precisely the value ϵ_i (i.e. the probability of convergence during the ith turn). Nevertheless we know that the borderline is located between $log_2(k1(i+k2))$ and $log_2((i+1))^{1/2}$.

Definition of rotating strategy	**Expectation time**
$\forall i, tr_i \leq k1$	$\leq (2n-4) + 2^{(k+1)\times(n-1)}$
$\forall i \geq M, tr_i \leq \log_2(k1(i+k2))^{1/2}$; and $tr_i - tr_{i+1} \geq -k3$	bounded
$\forall i \geq M, \log_2(k1(i+k2))^{1/2} < tr_i < \log_2(i+1)$ and $tr_i - tr_{i+1} \geq -k3$	???
$\forall i \geq M, \log_2(i+1) \leq tr_i \leq \log_2(k1(i+k2))$ and $tr_i - tr_{i+1} \geq -k3$	unbounded

References

[1] E. Anagnostou and V. Hadzilacos. Tolerating transient and permanent failures. In *WDAG93 Distributed Algorithms 7th International Workshop Proceedings, Springer-Verlag LNCS:725*, pages 174–188, 1993.

[2] 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.

[3] J. Beauquier, M. Gradinariu, and C. Johnen. Randomized self-stabilizing and space optimal leader election under arbitrary scheduler on rings. Technical Report 1225, L.R.I, December 1999.

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

[5] G. Bracha and S. Toueg. Asynchronous concensus and broadcast protocols. *Journal of the ACM*, 32:324–340, April 1985.

[6] JE Burns, MG Gouda, and RE Miller. On relaxing interleaving assumptions. In *Proceedings of the MCC Workshop on Self-Stabilizing Systems, MCC Technical Report No. STP-379-89*, 1989.

[7] EW Dijkstra. Ewd391, self-stabilization in spite of distributed control. In *Selected Writings on Computing: A Personal Perspective*, pages 41–46. Springer-Verlag, 1982. EWD391's original date is 1973.

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

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

[10] M. Duflot, L. Fribourg, and C. Picaronny. Finite-state distributed algorithms as markov chains. In *DISC00 Distributed Computing 15th International Symposium, Springer-Verlag LNCS:2180*, pages 240–255, 2001.

[11] M. H. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of consensus with one faulty process. *Journal of the ACM*, 32(2):374–382, April 1985.

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

[13] S.T. Huang, L.C. Wuu, and M.S. Tsai. Distributed execution model for self-stabilizing systems. In *ICDCS94 Proceedings of the 14th International Conference on Distributed Computing Systems*, pages 432–439, 1994.

[14] N. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.

[15] 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.

[16] A. Pogosyants, R. Segala, and N. Lynch. Verification of the randomized concensus algorithm of aspnes and herlihy: a case study. In *WDAG97 Distributed Algorithms 11th International Workshop Proceedings, Springer-Verlag LNCS:1320*, pages 22–36, 1997.

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

[18] 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.