RAPPORT DE RECHERCHE

L R I

ALL k-BOUNDED POLICIES ARE EQUIVALENT FOR SELF-STABILIZATION

BEAUQUIER J / JOHNEN C / MESSIKA S

Unité Mixte de Recherche 8623
CNRS-Université Paris Sud – LRI

07/2006

Rapport de Recherche N° 1455

CNRS – Université de Paris Sud
Centre d'Orsay
LABORATOIRE DE RECHERCHE EN INFORMATIQUE
Bâtiment 490
91405 ORSAY Cedex (France)

All k-bounded policies are equivalent for self-stabilization

Rapport de Recherche LRI n^o 1455

Joffroy Beauquier, Colette Johnen, Stéphane Messika

L.R.I./C.N.R.S., Université Paris-Sud 11,
bat 490, 91405 Orsay Cedex, France
jb@lri.fr, colette@lri.fr, messika@lri.fr

Abstract: We reduce the problem of proving the convergence of a randomized self-stabilizing algorithm under k-bounded policies to the convergence of the same algorithm under a specific policy. As a consequence, all k-bounded schedules are equivalent : a given algorithm is self-stabilizing under one of them if and only if it is self-stabilizing under any of them.

Key words: randomized algorithms, distributed algorithm, self-stabilizing system, scheduler.

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

Résumé: Nous réduisons le problème de prouver la convergence d'un algorithme probabiliste pour n'importe quelle politique k-bornée à la convergence du même algorithme pour une seule politique. Ainsi, toutes les politiques k-bornées sont équivalentes : un algorithme converge vers les configuratiosn légitimes pour une politique k-bornée fixée s'il converge pour toutes les politiques k-bornées.

Mots clés: algorithmes probabiliste, algorithme réparti, auto-stabilization, ordonnanceur.

1 Introduction

By their very nature, distributed algorithms have to deal with a non-deterministic environment. The speeds of the different processors or the message delays are generally not known in advance and may vary substantially from one execution to the other.
For representing the environment in an abstract way, the notion of scheduler (also called deamon or adversary) has been introduced. The scheduler is in particular responsible of which processors take a step in a given configuration or of which among the messages in transit arrives first. It is well known that the correctness of a distributed algorithm depends on the considered scheduler. This remark also holds for self-stabilizing distributed algorithm.
Different classes of schedulers have been considered in the literature on self-stabilization. Very often, the scheduler is viewed as a machine that chooses the subset of activable processors to be activated. For instance the synchronous scheduler chooses all enabled processors which take an elementary step concurrently, the central scheduler (central demon) chooses a single processor and then the processors take their steps one after the other, the distributed scheduler chooses a subset of enabled processors which take a step concurrently and the probabilistic scheduler draws randomly a subset of enabled processors. It can be assumed that the scheduler disposes of no memory at all, or of a bounded finite memory, or of an infinite memory. The second case corresponds to bounded schedulers (that can be either centralized or distributed).
If the synchronous scheduler is able to produce one policy, there is infinity of policies produced by the distributed schedulers (corresponding to all possible choices of subsets along the computation). Then stating that an algorithm is correct under a given scheduler means it is correct for each policy 'produced' by the scheduler. A "proof" that does not take into account all policies can hardly be considered as correct even if the algorithm is deterministic, and it is still worst for probabilistic algorithms, since the probabilistic measure of the executions depends on the considered policy. For instance under some policies the algorithm converges in a finite bounded number of steps (the stabilization time) while with others it can not converge at all. Even when the stabilization time is always finite, it can differ according to the policy.
Since it is not feasible to have a special proof for each policy, a convenient way to treat correctly the problem would be to prove general equivalence properties for some classes of policies. These properties would express that if an algorithm is correct for a particular policy of the class, then it is correct for any policy in the class. The aim of this paper is to present such an equivalence property.
We prove that all bounded policies are equivalent for self-stabilization. That means that if an algorithm can be proved to be self-stabilizing under a particular bounded policy, then it is also self-stabilizing for any bounded policy. Note that this class contains the synchronous policy.
Then, as a corollary we get that self-stabilization under the synchronous policy implies self-stabilization under any bounded policy. This leads to drastic simplifications in the proofs of already known results. For instance the result of [1], can be deduced directly from [9].

Related works. In [5], Dolev, Israeli and Moran introduced the idea of a two players game between the scheduler and what they call luck, i.e. random values, without defining formally the probabilistic space of computations. The structure (informally presented) behind a scheduler-luck game is a policy (formally defined in this paper) where some branches have being cut. In [15], [13], and [14], Lynch, Pogosyants and Segala present a formal method for analyzing probabilistic I/O automata which model distributed systems. A clear distinction between the algorithm, 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 [8, 7] the notion of randomized distributed algorithms under a fixed policy is studied using methods issued from Markov theory, in [7] the authors adapt these methods to Markov Decision Processes. In [3], we reduce the problem of computing the convergence time of a probabilistic self-stabilizing algorithm to an instance of the Stochastic Shortest Path problem (SSP). The reduction gives us a way to compute automatically the stabilization time against the worst and the best policy.

2 Notion of Markov Decision Process

In this section we adopt the notation of de Alfaro [4].
Informally, a Markov Decision Process is a generalization of the notion of Markov chain in which a set of possible actions is associated to each state. To each state-action pair corresponds a probability distribution on the states, which is used to select the successor state. A Markov chain corresponds thus to a Markov decision process in which there is exactly one action associated with each state. The formal definition is as follows.

Definition 1 (Markov Decision Process) *A Markov decision process (MDP)* $(S,\ Act,\ A,\ p)$ *consists of a finite set* S *of states, a finite set* Act *of actions, and two components* $A,\ p$ *that specify the transition structure.*

- *For each* $s \in S$, $A(s)$ *is the non-empty finite set of actions available at* s.
- *For each* $s, t \in S$ *and* $a \in A(s)$, $p_{st}(a)$ *is the probability of a transition from* s *to* t *when action* a *is selected.*
  Moreover, p *verifies the following property :* $\forall s, \forall a \in A(s)\ \sum_{t \in S} p_{st}(a) = 1$.

Definition 2 (Behavior of MDP) *A behavior of a Markov decision process is an infinite sequence of alternating states and actions, constructed by iterating a two phases selection process. First, given the current state* s, *an action* $a \in A(s)$ *is selected non deterministically; second the successor state* t *of* s *is chosen according to the probability distribution* $P(t|s,a) = p_{st}(a)$.
Given a state s *we denote* Ω_s *the set of all the behaviors starting in* s.

Definition 3 (cylinder sets) *The basic cylinder associated to the sequence* $h = s_0 a_0 s_1 a_1 ... s_n$ *contains all behaviors of a MDP starting at* s_0 *and having the same prefix* h:

$$C_h = \{hw \in \Omega_{s_0}\}$$

Now, we define some measurable sets of behaviors. For every state s, let $B_s \in 2^{\Omega_s}$ be the smallest algebra of subsets of Ω_s, that contains all the basic cylinder sets and that is closed under complement and countable unions and intersections. This algebra is called the Borel *σ-algebra* of the basic cylinder sets and its elements are the measurable sets of behaviors (see [4]).
To be able to talk about the probability of behaviors, we associate to each $\omega \in B_s$ a probability measure $P(\omega)$. However this measure is not well defined, since the probability that a behavior belongs to ω depends on how the actions have been nondeterministically chosen.
To represent these choices, we use the concept of policy (see [4]). Policies are closely related to the adversaries of Segala and Lynch [15] to the schedulers of Lehman and Rabin [11], Vardi [16] and Pnueli and Zuck [12], and to the notion of strategy [10]. Informally, a policy defines the probabilities with which the actions are chosen knowing the history of the machine states.

Definition 4 (Policy) *A policy η is a set of conditional probabilities $Q_\eta(a|s_0 s_1 ... s_n)$, for all $n \geq 0$, all possible sequences of states $s_0, ..., s_n$ and all $a \in A(s_n)$, such that $0 \leq Q_\eta(a|s_0, s_1 ..., s_n) \leq 1$ and $\sum_{a \in A(s_n)} Q_\eta(a|s_0, s_1 ..., s_n) = 1$.*
A policy is deterministic iff for each state s there is an action $a \in A(s)$ such that $Q_\eta(a|s_0, s_1 ..., s_n) = 1$.
A policy η is a memory k-bounded policy if for all $n \geq 0$, all possible sequences of states $s_0, ..., s_n$ we have $Q_\eta(a|s_0, s_1 ..., s_n s'_1 s'_2 ..., s'_k) = Q_\eta(s'_1 s'_2 ..., s'_k)$.
A policy is called memoryless if it is a memory 1-bounded policy.

Definition 5 (Probability measure of a cylinder under a policy) *Let η be a policy. Let $h = s_0 a_0 s_1 a_1 ... s_n$ be a sequence of computation steps. The probability of the basic cylinder associate to the the history h is*

$$P_s^\eta(C_h) = \prod_{k=0}^{n-1} p_{s_k s_{k+1}}(a_k) Q_\eta(a_k|s_0, s_1 ..., s_k)$$

It is well-known that there is an unique extension of the probabilistic measure P_s^η to any element of B_s. Thus the triple (η, B_s, P_s^η) defines a probabilistic space on B_s.
Note that a policy of a randomized distributed algorithm can be seen as a Markov chain.

3 Randomized Distributed Algorithms as Markov Decision Processes

We present how we model a randomized distributed algorithm as a Markov Decision Process (see [8, 7]for more details).
In a distributed system, the topology of the network of machines is usually given under the form of a communication graph $G = (V, E)$, where the set $V = \{1, ..., N\}$ corresponds to the machine set. There is an edge between two vertices when the corresponding machines can communicate directly. We assume that all the machines are finite state machines. A configuration X of the distributed system is the N-tuple of all the states of the machines. Given a configuration X, the state of the i^{th} machine is written $X(i)$. The code is a finite set of guarded rules: (i.e. label:: guard $\rightarrow$ action). The guard of a rule on p is a boolean expression involving p's state. The action of a p rule updates the p state. A machine p is *enabled* in a configuration c, iff a rule guard of p is true, in c. The simultaneous execution by several machines of rules is called a *computation step*.
The MDP associated with a distributed algorithm is defined by (i) S, the set of configurations, (ii) Act, the set of machine sets, (iii) $A(c)$, contain all subsets of enabled machines in c, (iv) $p_{st}(a)$, the probability to reach the configuration t from a configuration s by a computation step where all machines in a execute a rule.

Scheduler. A scheduler (adversary) is a mechanism which selects, at each step, a nonempty subset of enabled machines for applying the guarded rules of the algorithm. 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 must have the ablility to formalize any external behavior.

Definition 6 *Let DS be a distributed system. A scheduler D is a set of DS policies.*

The *synchronous daemon* [9] is the scheduler which "chooses" all enabled machines. This scheduler is a memoryless scheduler. A single deterministic policy is produced by the synchronous scheduler. A computation is *k-bounded* [2] if along any sequence where a machine p is continuously enabled, any other machine p' performs at most k actions before p performs an action. A policy is *k-bounded* if it contains only k-bounded computations. For a randomized distributed algorithm, an infinity of k-bounded policies exist. The k-bounded scheduler, is the set of k-bounded policies.

3.1 Probabilistic Convergence of a randomized protocol

The main idea behind these definitions is simple : To analyze a self-stabilizing algorithm under a scheduler, one has to analyze every Markov chain derived from the MDP associated to the algorithm combined with each policy 'produced' by the scheduler.

Definition 7 (Probabilistic convergence) *Let L be a predicate defined on configurations. A probabilistic distributed algorithm A under a scheduler D probabilistically converges to L iff : In any policy of η of D, from any configuration c, the probability of the set of computations reaching a configuration satisfying L is equal to 1. Formally,* $\lim_{n\to\infty} P_c^{\eta}(\exists m \leq n \mid X_m \in L) = 1$ *where X_m is the reached state after m computation steps in the Markov chain defined as the MDP associated to A, c and η.*

4 Extension to all k-bounded policies

We will show in the sequel that, under some simple conditions, the probabilistic converge under **a** policy guarantees the probabilistic convergence under **any** k-bounded policies. After that, it is only needed to prove the convergence under **a** policy to formally prove the probabilistic convergence under the k-bounded scheduler for any value of k.

The first hypothesis we will assume is the serializability, a classical concurrency notion. It ensures that a schedule for executing concurrent machine rules is equivalent to one that executes the machine rule serially in some order.

Definition 8 *A computation step sas' is serializable iff s' is reachable from s by a series of computation steps where only a machine performs an action.*
An history $h = s_0a_0s_1a_1...s_l$ is serializable iff each computation step of h is serializable.

The second hypothesis is a property of probabilistic algorithms.

Definition 9 *A probabilistic distributed algorithm is potentially stable if and only if for each guarded rule there is a no zero probability that the execution of the rule does not change the machine state.*

Notation 1 *$|h|$ denotes the length of the sequence h.*
N is the number of machines in the system.

Lemma 1 *Let A be a potentially stable algorithm. Assume that there exist s, s', and a such that $p_{ss'}(a) > 0$ and a contains a machine. Then, there is a real number $\varepsilon > 0$ such that in any k-bounded policy η, for any initial configuration s_0, for any history h ending at the configuration s,*

there exists a sequence h' of computation steps such that
(i) $P^{\eta}_{s_0}(w \in C_{hh'}) > P^{\eta}_{s_0}(w \in C_h)\varepsilon$, (ii) $|h'| \leq kN$, (iii) $\varepsilon \geq {\varepsilon_b}^{kN^2} p_{ss'}(a)$, and (iv) the last configuration of h' is s'.

Proof:
We prove that under any policy, it possible to reach s' after an history reaching s, in less that kN computation steps with a probability greater than ε.
As the number of rules is finite, there exists a real number ε_d, such that for any rule performed by any machine, the probability that this machine state does not change, is at least ε_d. Thus, in any case, the probability that no machine changes its state during a computation step is at least ${\varepsilon_b}^N$.

Let η be a k-bounded policy. Let h be an history $s_0 a_0 s_1 a_1 s_2 ... a_m s$ such that $P^{\eta}_{s_0}(w \in C_h) \neq 0$.

An action a is the set of machines that perform a rule during the associated computation step. We study the case where only one machine performs a rule; we name it machine 1.

The η policy is k-bounded, thus there exists an action a_i in which the machine 1 executes a rule such that (i) $Q_{\eta}(a_i | s_0, s_1 ..., s_{m+i-1}) > 0$ assuming that $s_{m+j} = s \; \forall j \in [0, i-1]$, and (ii) $1 \leq i < kN$.
Note that by definition of a k bounded policy, we have the following property: $\forall l \in [0, i-1]$ there exists an action a_l such that $Q_{\eta}(a_i | s_0, s_1 ..., s_{m+1}) > 0$ assuming that $s_{m+j} = s \; \forall j \in [0, l]$.
We denote by h' the sequence $sa_1 sa_2 .. sa_i s'$ defined as: (i) during the execution of a_j where $j < i$ no machine changes its state, and (ii) during the execution of a_i only the machine 1 changes its states. We have (i) $P^{\eta}_{s_0}(w \in C_{hh'}) > P^{\eta}_{s_0}(w \in C_h)\varepsilon$, (ii) $\varepsilon > {\varepsilon_b}^{i+1N^2} p_{ss'}(a)$, (iii) $|h'| < kN$. □

Lemma 2 *Let A be a potentially stable algorithm. Assume that there exist s, s', and a such that $p_{ss'}(a) > 0$ and $cs = sas'$ is serializable. Then, there is a real number $\varepsilon > 0$ such that in any k-bounded policy η, for any initial configuration s_0, for any history h ending at the configuration s, there exists a sequence h' of computation steps such that*
(i) $P^{\eta}_{s_0}(w \in C_{hh'}) > P^{\eta}_{s_0}(w \in C_h)\varepsilon$, (ii) $|h'| \leq kN^2$, (iii) $\varepsilon \geq {\varepsilon_b}^{kN^3} p_{ss'}(a)$, and (iv) the last configuration of h' is s'.

Proof:
We prove that under any policy, it possible to reach s' after an history reaching s, in less that kN^2 computation steps with a probability greater than ε.
$cs = sas'$ is serializable thus there exists a sequence of computation steps $s_0 a'_1 s'_1 a'_2 s'_2 .. a'_n s'_n$ of length $n < N$ that reaches s' from s and along this sequence, at most one machine performs an action at a time. We call i the machine executing a rule during a_i. $\forall i \in [0, n-1]$, we have $p_{s'_i s'_{i+1}}(a'_i) > 0$ and, by definition, $p_{ss'}(a) = \prod_{i=0}^{n-1} p_{s'_i s'_{i+1}}(a'_i)$.

Let η be a k-bounded policy. Let h be an history $s_0 a_0 s_1 a_1 s_2 ... s$ such that $P^{\eta}_{s_0}(w \in C_h) \neq 0$.

According to lemma 1, for $1 \leq i \leq n$, there exists an history $h_i = h_{i-1} a_1 s_2 a_2 ...$ having the following properties (i) $h_0 = h$, (ii) $P^{\eta}_{s_0}(w \in C_{h_i}) > P^{\eta}_{s_0}(w \in C_{h_{i-1}})\varepsilon_i > 0$, (iii) $|h_i| < h_{i-1} + kN$, (iv) $\varepsilon_i \geq {\varepsilon_b}^{kN^2} p_{s_{i-1} s'_i}(a'_i)$, and (v) the last configuration of h_i is s'_i.

We conclude that the history h_n has the following properties (i) $P^{\eta}_{s_0}(w \in C_{h_n}) > P^{\eta}_{s_0}(w \in C_h)\varepsilon$ (ii) $|h_n| < |h| + kN^2$, (iii) $\varepsilon \geq {\varepsilon_b}^{kN^3} p_{ss'}(a)$, and (iv) the last configuration of h_n is s'. □

Lemma 3 *Let A be a potentially stable algorithm. Assume that there exists η_s, a policy such that there is an history h_s of length l reaching a legitimate configuration and there is a real number $\varepsilon_s > 0$ such that (i) $P_{c0}^{\eta_s}(w \in C_{h_s}) > \varepsilon_s$, and (ii) h_s is serializable.*
Then, there is a real number $\varepsilon > 0$ such that in any k-bounded policy η, for any initial configuration c, for any history h ending at the configuration $c0$, there is a sequence h' such that (i) $P_c^{\eta}(w \in C_{hh'}) > P_c^{\eta}(w \in C_h)\varepsilon$, (ii) $|h'| \leq lkN^2$, (iii) $\varepsilon > {\varepsilon_b}^{lkN^3}\varepsilon_s$, and (iv) h' reaches a legitimate configuration.

Proof:
We prove that under any policy, it possible to reach a legitimate configuration after an history reaching $c0$, in less that lkN^2 computation steps with a probability greater than ${\varepsilon_b}^{lkN^3}\varepsilon_s$.

There exists a sequence $h_s = c_0a_0c_1a_1c_2..a_{l-1}c_l$ such that $P_{c_0}^{\eta_s}(w \in C_{h_s}) > \varepsilon_s$ and c_l is a legitimate configuration.
We have $\varepsilon_s = \prod_{i=1}^{l} p_{c_{i-1}c_i}(a_{i-1}) > 0$. Thus $\forall i \in [1, l]$, we have $p_{c_{i-1}c_i}(a_{i-1}) > 0$. Moreover, $c_{i-1}a_{i-1}c_i$ is serializable, $\forall i \in [1, n]$.

Let η be a k-bounded policy. Let h be an history such that $P_c^{\eta}(w \in C_h) \neq 0$ and the last configuration of h is $c0$.

According to lemma 2, for $1 \leq i \leq n$, there exists an history h_i having the following properties (i) $P_c^{\eta}(w \in C_{h_i}) > P_c^{\eta}(w \in C_{h_{i-1}})\varepsilon_i > 0$ where $h_0 = h$ (ii) $\varepsilon_i > {\varepsilon_b}^{kN^3} p_{c_{i-1}c_i}(a_{i-1})$, (iii) $i < kN^2$ and (iii) the last configuration of h_i is c_i.

We conclude that the history h_l has the following properties (i) $P_c^{\eta}(w \in C_{h_l}) > P_c^{\eta}(w \in C_h)\varepsilon$, (ii) $|h_l| < |h| + lkN^2$, (iii) $\varepsilon > {\varepsilon_b}^{lkN^3}\varepsilon_s$, and (iv) the last configuration of h_l is legitimate. □

Lemma 4 *Let A be a potentially stable algorithm. Assume that there exist a policy η_s, a real number $\varepsilon_s > 0$ and an integer l such that from any initial configuration c there is an history h reaching a legitimate configuration with (i) $P_c^{\eta_s}(w \in C_h) > \varepsilon_s$, (ii) h is serializable, and (iii) $|h| \leq l$.*
Then there is a real number $\varepsilon > 0$ such that in any k-bounded policy η, for any initial configuration c, for any history h there is an sequence h' such that (i) $P_c^{\eta}(w \in C_{hh'}) > P_c^{\eta}(w \in C_h)\varepsilon$, (ii) $|h'| \leq lkN^2$, (iii) $\varepsilon > {\varepsilon_b}^{lkN^3}\varepsilon_s$, and (iv) h' reaches a legitimate configuration.

Proof: We prove that under any policy, it possible to reach a legitimate configuration after any history, in less that lkN^2 computation steps with a probability greater than ${\varepsilon_b}^{lkN^3}\varepsilon_s$.

Let η be a k-bounded policy. Let h be an history $s_0a_0s_1a_1s_2...s_n$ such that $P_c^{\eta}(w \in C_h) \neq 0$.

According to the hypothesis, there is an history h_s of length lesser than l reaching a legitimate configuration such that (i) $P_{s_n}^{\eta_s}(w \in C_{h_s}) > \epsilon_s$ and (ii) h_s is serializable.
According to lemma 3, there exists a sequence h' having the following properties (ii) $P_{c0}^{\eta}(w \in C_{hh'}) > P_{c0}^{\eta}(w \in C_h)\epsilon > 0$, (iii) $\varepsilon > {\varepsilon_b}^{lkN^3}\varepsilon_s$, (iv) $|h'| < klN^2$ and (iii) the last configuration of h' is legitimate. □

Theorem 1 *Let A be a potentially stable algorithm. Assume that there exist a policy η_s, a real number $\varepsilon_s > 0$ and an integer l such that from any initial configuration c there is an history reaching a legitimate configuration with (i) $P_c^{\eta_s}(w \in C_h) > \epsilon_s$, (ii) h is serializable, and (iii) $|h| \leq l$.*

Under the k-bounded scheduler, Algorithm A probabilistically converges to the legitimate configuration set.

Proof: Let η be a k-bounded policy. Let c be a configuration.
According to lemma 4, there is a real number $\varepsilon > 0$ and an integer D such that for any history h there is an sequence h' such that (i) $P_c^\eta(w \in C_{hh'}) > P_c^\eta(w \in C_h)\varepsilon$, (ii) $|h'| \leq D$, (iii) h' reaches a legitimate configuration.

Let L be the set of legitimate configurations.
Thus we have $P_c^\eta(X_n \; reaches \; L) > 1 - (1-\varepsilon)^n$ where X_n contains all the histories of length lesser or equal to Dn. We conclude that $\lim_{n\to\infty} P_c^\eta(X_n \; reaches \; L) = 1$. Under the policy η, Algorithm A probabilistically converges to the legitimate configurations set.

In summary under any k-bounded policy, algorithm A probabilistically converges to the legitimate configurations set. The expected number of computation steps for reaching a legitimate configuration is bounded by $\frac{D}{\varepsilon}$ Notice that $\varepsilon > {\varepsilon_b}^{lkN^3}\varepsilon_s$ and $D \leq lkN^2$, according lemma 4. □

5 Examples

The aim of these examples is to illustrate how our results can be used. For each algorithm, we exhibit a particular k-bounded policy for which the stabilization proof is easy. Then, we get, than the algorithm is stabilizing for any k-bounded policy.

5.1 Self-stabilizing vertex coloring

Algorithm 1 Self-stabilizing vertex coloring algorithm

Constant in p:
B is a constant in N, we assume that $B > \Delta$ (the degree)

Variable on p: c_p color of p machine, taking its values in B

Action on p:
$\mathcal{R}$:: $\exists q \in N_p$ such that $c_p = c_q \longrightarrow c_p = random(1, B)$

In this section, we study a very simple self-stabilizing vertex coloring algorithm (Algorithm 1). The algorithm converges from any configuration to a configuration where neighboring machines do not have the same color. A machine that has the same color as one of its neighbors is enabled. An enabled machine can randomly choose any color in the colors set (i.e. execute the $\mathcal{R}$ action). All colors have the same probability to be chosen: $1/B$ (B being the color set size). We assume that B is greater than the maximum machine degree, denoted Δ.

Let us study the algorithm under the memoryless policy η that chooses at each computation step one of the enabled machine. At each computation step, the probability that the executing machine chooses a color distinct of its neighbor colors is at least $\frac{B-\Delta}{B}$. Using the *measure* technique proposed in [6], one proves that from any initial configuration c, there is an history h reaching a

legitimate configuration such that (i) $P_c^{\eta}(w \in C_h) > (\frac{B-\Delta}{B})^{N-1}$, and (ii) $|h| \leq N-1$.

Using the theorem 1, we directly establish that the vertex coloring algorithm converges under any k-bounded policy.

5.2 Token circulation

Consider the following property:

Proposition 1 *there is a real $\varepsilon_s > 0$ and an integer l such that from any initial configuration c, there is an history h reaching a legitimate configuration such that (i) $P_c^{\eta_s}(w \in C_h) > \epsilon_s$, (ii) h is serializable, and (iii) $|h| \leq l$.*

We showed in the previous section that once this proposition is true for a policy η then the algorithms converges to its legitimate configuration under any k-bounded policy.
Herman [9] has proposed a token circulation algorithm under unidirectional rings of size 2N+1. This algorithm is a randomly delayed circulation (see code in Algorithm 2). Only a machine 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. Finally, the specification is that eventually, only one token circulates in the ring. In [9], the algorithm was proven under the synchronous policy. This algorithm is very interesting to analyse because there exists memoryless policy under which the algorithm does not converge. For instance, under the memoryless policy that chooses at each computation step one of the token in the set FAR, the set of tokens at maximum distance of predecessor.

Algorithm 2 token circulation on anonymous and unidirectional rings

Variables on p: v_p is a boolean variable;

Random Variables on p:
$rand\_bool_p$ taking value in {1, 0}. Each value has a probability 1/2.

Action on p: lp is the machine preceeding p
$\mathcal{R}$:: $v_p == v_{lp} \rightarrow$ if $rand\_bool_p = 1$ then $v_p := (v_p + 1) \bmod 2$;

Let us study the algorithm under the memoryless policy η that chooses at each computation step one of the tokens in the set $NEAR$, the set of tokens at minimum distance from teh predecessor. All computations under this policy are serializable, because in a computation step, a single machine performs an action. Using the *measure* technique proposed in [6], it can be proven that from any initial configuration c, there is an history h reaching a legitimate configuration such that (i) $P_c^{\eta}(w \in C_h) > \frac{1}{2^{2N}}$, and (ii) $|h| \leq 2N$.
Using the theorem 1, we directly establish that Herman's algorithm converges under any k-bounded policy. Note that the policy η is not a k-bounded policy. Five years after the publication of this algorithm, Beauquier and al., [1], have proven the convergence of this algorithm under any k-bounded memory policy.

In the next section, we prove that a k-bounded memory policy is a k-bounded policy the converse is not true.

6 Comparison of k-bounded and memory k-bounded policies

In this section, we assume all policies to be fair. A policy η is unfair if (i) there is an infinite computation in which a machine is continously enabled and never activated (ii) any prefix of this computation has a positive probability.

Definition 10 (Fairness) *Let η be a policy. Let comp be a computation where p is always enabled. eta is said fair iff it exists n_{comp} such that $p \in a$ and $Q_\eta(a|$prefix of length n_{comp} of comp$)> 0$ or P^η(prefix of length n_{comp} of comp$)= 0$.*

Proposition 2 *Let A be an algorithm such that any machine has a bounded number T of states. Any fair memory k-bounded policy is α-bounded with $\alpha = T^{N^k} + k + 1$.*

Proof: Let η be a fair memory k-bounded policy.
Note that the number of distincts configuration sequences of length k is bounded by T^k.
Consider a computation *comp* of length $\alpha = T^{N^k} + k + 1$ where some machine p is always enabled and never performs a rule with $P^\eta(comp) > 0$. If no such computation exists then the algorithm is α-bounded. In *comp*, A same sequence of length k, s necesseraly appears twice: Thus $comp = s_0, s, s', s, s_f,$. We have $P^\eta(s, s') > 0$. Let us study the computation $comp' = (s, s)^*$. p is always enabled during $comp'$, for any value of n we have P^η(prefix of length n of comp')> 0. η is not fair because p is never in the set of selected machines by the policy along $comp'$: if $Q_\eta(a|$prefix of length n_{comp} of comp$) > 0$ then p is not an element of a. There is a contradiction: η is a fair policy. □

This proves that the class of memory k-bounded policies is included in the class of k-bounded policies.

Policy 1 A k-bounded policy that is not a memory K-bounded policy

---

Constant: N, the network size

Initialisation: counter0 := 1; counter1 := 0; pn := 0;

Policy choice in fonction of history length:
if the history length is an even number then all enabled machines are selected;
else
if (count1 == 0) then counter0 := counter0*2; counter1 := counter0; np := np+1 mod N;
else counter1 := counter1-1;
fi
while machine np is not enabled do np:=np+1 mod N; done
The machine np is selected;
fi

There are k-bounded policies that are not memory K-bounded policies (cf. policy 1 below). Policy 1 is 2-bounded, because all enabled machines execute a rule during even computation steps, but it is not memory K-bounded for any K.
A possible sequence of choices is:

$$(\{p1, p2\}, p1)^{2^1}, (\{p1, p2\}, p2)^{2^2}, (\{p1, p2\}, p1)^{2^3}, (\{p1, p2\}, p1)^{2^4}, (\{p1, p2\}, p2)^{2^5}, ...$$

7 Conclusion

In this paper we show that under assumptions all the k-bounded policies are equivalent for self-stabilization. Then, when an algorithm can be proven to be self-stabilizing for a particular k-bounded policy, it is also self-stabilizing for any k-bounded policy. This property is specially interesting when the self-stabilization proof is easy for a particular policy. The more obvious choice is the synchronous policy, but as we demonstrate it in the examples, some othe policies may be used in each particular case. The property allows to simplify existing proofs, to make some of them unnecessary (Herman's example).

References

[1] J. Beauquier, S. Cordier, and S. Delaët. Optimum probabilistic self-stabilization on uniform rings. In *WSS95 Proceedings of the Second Workshop on Self-Stabilizing Systems*, pages 15.1–15.15, 1995.

[2] J. Beauquier, M. Gradinariu, and C. Johnen. Memory space requirements for self-stabilizing leader election protocols. In *Proc. of the 18th Annual ACM Symposium on Principles of Distributed Computing (PODC'99)*, pages 199–208, 1999.

[3] J. Beauquier, C. Johnen, and S. Messika. Brief announcement: Computing automatically the stabilization time against the worst and the best schedulers. In *Proc. 20th Int. Conf. on Distributed Computing (DISC 2006)*, 2006.

[4] L. de Alfaro. *Formal Verification of Probabilistic systems.* PhD Thesis, Stanford University, 1997.

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

[6] M. Duflot, L. Fribourg, and C. Picaronny. Finite-state distributed algorithms as markov chains. In *Distributed Computing 15th International Symposium (DISC01), Springer-Verlag LNCS:2180*, pages 240–254, 2001.

[7] L. Fribourg and S. Messika. Brief announcement: Coupling for markov decision processes - application to self-stabilization with arbitrary schedulers. In *Proc. of the 24th Annual ACM Symposium on Principles of Distributed Computing (PODC05)*, page 322, 2005.

[8] L. Fribourg, S. Messika, and C. Picaronny. Coupling and Self-stabilization. In *Proc. 18th Int. Conf. on Distributed Computing (DISC 2004), Springer-Verlag, LNCS 3274*, pages 201–215. Springer, 2004.

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

[10] Colette Johnen. Service time optimal self-stabilizing token circulation protocol on anonymous unidirectional rings. In *SRDS 2002 Proceedings of the 21th Symposium on Reliable Distributed Systems.* IEEE, October 2002.

[11] D. Lehmann and M. O. Rabin. On the advantages of free choice: a symmetric and fully-distributed solution to the dining philosophers problem. In *Proc. 8th Annual ACM Symp. on Principles of Programming Languages (POPL'81)*, pages 133–138, 1981.

[12] A. Pnueli and L. Zuck. Verification of multiprocess probabilistic protocols. *Distributed Computing*, 1(1):53–72, Jan. 1986.

[13] A. Pogosyants and R. Segala. Formal verification of timed properties of randomized distributed algorithms. In *In Proc. of the 14th Annual ACM Symposium on Principles of Distributed Computing (PODC95)*, pages 174–183, 1995.

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

[15] R. Segala and N. Lynch. Probabilistic simulations for probabilistic processes. In *5th International Conference on Concurrency Theory (CONCUR'94), Springer-Verlag, LNCS:836*, pages 481–496, 1994.

[16] M. Y. Vardi. Automatic verification of probabilistic concurrent finite-state programs. In *26th Annual Symposium on Foundations of Computer Science, IEEE Computer Society (FOCS'85)*, pages 327–338, 1985.