Stabilizing leader election in population protocols

Davide Canepa, Maria Gradinariu Potop-Butucaru

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Stabilizing leader election in population protocols

Davide Canepa — Maria Gradinariu Potop-Butucaru
Université Pierre et Marie Curie (Paris 6), CNRS,INRIA, France

canepa.davide@tiscali.it, maria.gradinariu@lip6.fr

N° ????

Jully 2007

Thème COM

Rapport de recherche

ISSN 0249-6399 ISRN INRIA/RR--????--FR+ENG

Stabilizing leader election in population protocols

Davide Canepa , Maria Gradinariu Potop-Butucaru
Universit Pierre et Marie Curie (Paris 6), CNRS,INRIA, France
canepa.davide@tiscali.it, maria.gradinariu@lip6.fr

Thème COM — Systèmes communicants
Projet REGAL

Rapport de recherche n° ???? — Jully 2007 — 14 pages

Abstract: In this paper we address the stabilizing leader election problem in the population protocols model augmented with oracles. Population protocols is a recent model of computation that captures the interactions of biological systems. In this model emergent global behavior is observed while anonymous finite-state agents(nodes) perform local peer interactions. Uniform self-stabilizing leader election is impossible in such systems without additional assumptions. Therefore, the classical model has been augmented with the eventual leader detector, $\Omega?$, that eventually detects the presence or absence of a leader. In the augmented model several solutions for leader election in rings and complete networks have been proposed. In this work we extend the study to trees and arbitrary topologies. We propose deterministic and probabilistic solutions. All the proposed algorithms are memory optimal — they need only one memory bit per agent. Additionally, we prove the necessity of the eventual leader detector even in environments helped by randomization.

Key-words: Population Protocols, Leader election

Unité de recherche INRIA Rocquencourt
Domaine de Voluceau, Rocquencourt, BP 105, 78153 Le Chesnay Cedex (France)
Téléphone : +33 1 39 63 55 11 — Télécopie : +33 1 39 63 53 30

Election de leader dans le modèle “Population Protocols”

Résumé : Dans ce papier nous nous intéressons à l'élection de leader dans le modèle des “populations protocols”. Le modèle classique des population protocols semble ne pas etre suffisamment puissant pour implémenter l'élection de leader et ceci meme avec l'utilisation de la randomization. Pour palier ce problème des hypothses supplémentaires sont faites sur l'environnement et en particulier l'utilisation des oracles semble un bon compromis. Dans ce travail, nous montrons que pour résoudre le problème de l'élection de leader pour tout type de graphe dans le modèle “population protocols” il suffit d'avoir un oracle de type $\Omega?$ qui founit au bout d'un temps fini à tous les sites l'information si un leader est présent ou non dans le système.

Mots-clés : Election de leader, résultats d'impossibilité

1 Introduction

Recently, the distributed computing community started to investigate the interactions in biological and chemical systems in order to provide efficient computational models for adhoc systems like sensor, peer-to-peer or robots networks. One of the most promising research in this direction is the population protocol model developed by Dana Angulin *et al.* through a serie of papers [1, 2, 3, 4, 5]. In this model, finite-state agents interact in pairs chosen by an adversary, with both agents updating their state according to a joint transition function. For each such transition function, the resulting population protocol is said to stably compute a predicate on the initial states of the agents if, after sufficiently many interactions in a fair execution, all agents converge to having the correct value of the predicate. Agents in this framework have no identity and no ability to distinguish two different interactions with the same agent.

Interestingly, interactions between agents in this model have various incarnations. Sometime interactions are restricted by the choice of a fair or randomized scheduler while in other situations the network topology is the main parameter that defines them. This corresponds perfectly to the real networks. In sensor or robots networks, for example, the topology is the main parameter having a major impact on sensor interactions while in peer-to-peer networks, where interactions are not restricted by topological constraints, agents can collaborate following their personal choice.

1.1 Related work

Several problems have been addressed in the population protocols model: coloration, phase synchronisation, counting, leader election [1, 2, 3, 6, 7]. In [8] the author addressed the fault tolerance issues in this model.

Leader election in population protocols model was firstly defined in [2]. In [3] the work was extended to the non-uniform leader election in odd and directed rings. The algorithm foresee probes that the Leaders send in order to kill the other Leaders and assumes global fairness. Fisher and Jiang developed the leader-election problem in the same model in [9]. They introduce $\Omega?$ an oracle that reports true or false if it detects the presence or absence of a leader. Using $\Omega?$, the authors provide uniform and self-stabilizing leader election algorithms for fully connected networks under the assumption of local fairness and for rings under the global fairness assumption. In [9] the authors also prove that uniform leader election is impossible in rings assuming local fairness, even with the help of $\Omega?$.

1.2 Our contribution

In this work we consider the leader election problem in the population protocol model with very weak agents. The considered agents are uniform, have no identification and store permanently only one memory bit — the minimum possible in order to achieve leader election.

The purpose of this paper is to extend the Fisher's work [9] to various topologies and interaction restrictions. We provide deterministic solutions for trees and probabilistic solutions for arbitrary graphs. We also prove the necessity of $\Omega?$ in order to solve the problem under both deterministic and probabilistic assumptions. Additionally we provide results related to the leader election feasibility when the interaction model is restricted to directed one-way interactions.

1.3 Paper Roadmap

The paper is organized as follows. Section 2 recalls the main elements defining the population protocol model. Section 3 defines the eventual leader detector and provides some impossibility results. Sections 4 and 5 discuss the feasability of Leader Election in DAGs and rooted trees. In Section 6 is proposed a probabilistic solution for leader election in general graphs. Some open problems are discussed in Section 7.

2 Model

We propose an overview of the population-protocol model required to present the results in this paper. A more detailed description is available in [3].

We represent a network by a directed graph G = (V,E) with no multi-edges or self-loops. Each vertex represents a finite-state sensing device (agents) and an edge $(u,\ v)$ indicates the possibility of a communication between u and v in which u is the *initiator* and v is the *responder*. For a node u, $\mathcal{N}_u$ is the set of all the processor v_i responder of u.

A protocol P(Q, $\mathcal{C}$,X, Y, O, δ) consists of a finite set of states Q, a set of initial configurations $\mathcal{C}$, a finite set X of input symbols, an output function O:Q$\rightarrow$Y , where Y is a finite set of output symbols, and a transition function δ mapping each element of (Q x X) x (Q x X) to a nonempty subset of Q x Q. If $(p',\ q') \in \delta((p,x),\ (q,y))$, we call $((p,x),\ (q,y)) \rightarrow (p',q')$ a transition. The transition function, and the protocol, is deterministic if $\delta((p,x),\ (q,y))$ always contains just one pair of states. Otherwise we call it probabilistic.

The inputs provide a way for a protocol to interact with an external entity, be it the environment, a user, or another protocol. In this paper, a node i interacts with its leader detector through the input port. A configuration is a mapping $\mathcal{C} : V \rightarrow Q$ specifying the state of each device in the network, and an input assignment is a mapping $\alpha : V \rightarrow X$. A trace T_G(Z) on a graph G(V,E) is an infinite sequence of assignments from V to the symbol set Z: $T_G = \lambda_0,\ \lambda_1,\ \ldots$ where λ_i is an assignment from V to Z. The set Z is called the alphabet of T_G. If Z = X, then each λ_i is an input assignment, and we say T_G is an input trace of the protocol.

Let $\mathcal{C}$ be a configuration and α an input assigment, if it exists a transaction function $\delta((p,x),(q,y))$ such that if $\mathcal{C}(u) = p$ and $\alpha(u) = x$ than it exists a node $v \in N_u$ such that $\mathcal{C}(v) = q$ and $\alpha(u) = x$. Node u is said enabled for v.

Let $\mathcal{C}$ and $\mathcal{C}'$ be configurations α be an input assignements and $u_1,\ u_2,\ \ldots\ u_m,\ v_1,\ v_2,\ldots$ v_m, distincts nodes. We say that $(\mathcal{C},\ \alpha)$ goes to $\mathcal{C}'$ via 'set of pair' $\underline{e}=(e_1,e_2,\ \ldots\ e_m)$, where

e_i=(u_i, v_i), denoted ($\mathcal{C}$, α) $\rightarrow \mathcal{C}'$, if each pair ($\mathcal{C}'(u_1)$, $\mathcal{C}'$ (v_1)) is in δ(($C(u_1)$, $\alpha(u_1)$), ($\mathcal{C}(v_1)$, $\alpha(v_1)$)) and for all $w \in$ V-{ u_1, u_2, ... u_m, v_1, v_2, ... v_m} we have $\mathcal{C}$'(w)=$\mathcal{C}(w)$.

We say that α can go to $\mathcal{C}'$ in one step denoted ($\mathcal{C}$, α) $\rightarrow \mathcal{C}'$ if ($\mathcal{C}$, α) $\rightarrow \underline{e}$ $\mathcal{C}'$ for some set of edge $\underline{e} \in$ E. That is, a passage from a configuration $\mathcal{C}$ to a configutation $\mathcal{C}'$ occurs due to the execution of a nonempty subset of processus u_i enabled for another subset of node v_i such that any node of a subset occurs two times in it subset or once in the other subset.

2.1 Behavior, Implementation and Self-stabilization

A self-stabilizing system can start at an arbitrary configuration and eventually exhibit 'good' behavior. We define a behavior B on a network G(V,E) to be a set of traces on G that have the same alphabet. We write B(Z) to be explicit about the common alphabet Z. A behavior B is constant if every trace in B is constant.

If the output trace of every fair execution of a protocol P(Q,$\mathcal{C}$,X,Y,O,δ) starting from any configuration in $\mathcal{C}$ is in some behavior B_{out}(Y), we say P is an implementation of output behavior B_{out}. Given a behavior B(Z), we define the corresponding stable behavior B^s(Z): T $\in B^s$ if and only if Z is T's alphabet, and there exists T' $\in B$ such that T' is a suffix of T. Thus, an execution in a stable behavior may have a completely arbitrary finite prefix followed by an execution with the desired properties. If P(Q,$\mathcal{C}$,X,Y,O,δ) is an implementation of B^s, and $\mathcal{C}$ is the set of all possible configurations, we say that P is a self-stabilizing implementation of B.

Definition 1 (Leader Election) *The leader-election behavior LE on graph G = (V,E) is the set of all constant traces β,β, . . . such that for some $v \in V$, $\beta(v) = L$ and for all $u \neq v$, $\beta(u) = N$. Informally, there is a static node with the leader mark L, and all other nodes have the non-leader mark N in every configuration.*

2.2 Fairness

In [9] the authors defined two fairness conditions for population protocol model: the local fairness and the global fairness. Let $e = (C_0, \alpha_0), (C_1, \alpha_1), \ldots, (C_i, \alpha_i), \ldots$ be an execution.

Definition 2 (Global fairness) *For every C,α and C' such that $(C, \alpha) \rightarrow C'$, if $(C, \alpha) = (C_i, \alpha_i)$ for infinitely many i, then $(C_i, \alpha_i) = (C, \alpha)$ and $C_{i+1} = C'$ for infinitely many i. (Hence, the step $(C, \alpha) \rightarrow C'$ is taken infinitely many times in e.)*

Definition 3 (Local fairness) *For every action σ, if σ is enabled in (C_i, α_i) for infinitely many i, then $(C_i, \alpha_i) \rightarrow C_{i+1}$ for infinitely many i. (Hence, the action σ is taken infinitely many times in E.)*

Global fairness asserts that each step $(C, \alpha) \rightarrow C'$ that can be taken infinitely often is actually taken infinitely often. By way of contrast, local fairness only asserts that each action σ that can be taken infinitely often is actually taken infinitely often.

2.3 Eventual Leader Detector

The *eventual leader detector*, $\Omega?$ was introduced in the context of population protocols in [9]. This detector supplies a Boolean input to each process at each step so that the following conditions are satisfied by every execution e.

- If all but finitely many configurations of e lack of leader, then each process receives input *false* at all but finitely many steps.
- If all but finitely many configurations of e contain one or more leaders, then each process receives input *true* at all but finitely many steps.

Note that eventual leader detector $\Omega?$ is a weaker version of the oracle Ω introduced first in [10] and proved to be the weakest failure detector to solve consensus. Instead of electing a leader (as Ω does), $\Omega?$ reports to each node whether or not at least one leader is present in the network. Note that the guess may be correct or not and different guesses may be reported to different nodes. The only guarantee offered is that from some point onward if there is continously a leader or if there is continously no leader, $\Omega?$ eventually accuratly reports this fact to all nodes.

2.4 Work hypothesis

In the following we assume *weak agents*. Weak agents do not have unique identification and they are uniform — all of them execute the same algorithm. Moreover, they are provided with a memory limited to a single bit. We also assume each agent receives boolean inputs from the eventual leader oracle that reports true if a leader is present in the network or false otherwise. The network topology we consider is predefined and does not change during the algorithm execution. The interactions between agents are restricted by the topology and the schedular choice. In the following we assume undeterministic schedulars verifying the local fairness property. Note that we do not consider the randomized schedulers however some of our algorithms are randomized.

3 Leader election and necessity of $\Omega?$

In the following we show the necessity of eventual leader oracle in order to provide uniform solutions for leader election.

Lemma 1 *The leader detector $\Omega?$ is necessary to implement leader election.*

Proof: $\Omega?$ plays a key role in the creation of new leaders. In the following we prove that without $\Omega?$ it's impossible to know if in the graph there is a leader, hence nodes may introduce leaders infinitely and the algorithm never converges to a legitimate configuration (a single leader is present in the system). Consider a chain topology and two initial configurations one without leaders, the other with a leader in node D (see Figure 1).

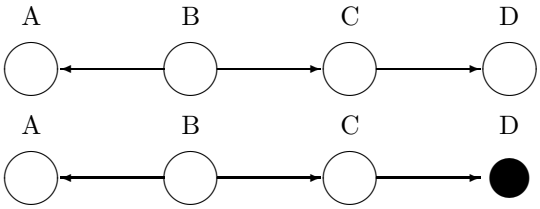

Figure 1: Two initial configurations for Lemma 1

Consider node B. It can interact only with nodes A and C. Nodes A an C are not leaders in both graphs so in the B's view these two configurations are identical. If B decides to introduce a leader, it must to do it in both graphs which will transform the second configuration in an illegitimate configuration since it will contain two leaders. From this configuration, if B decides to become non-leader, it changes its state in the first graph too. So, the configuration of the first graph will be back to the initial situation (no leader in the system). Hence, we can exhibit an infinite execution that never converges to a legitimate configuration. Overall, without Ω? it's impossible to decide if or not a new leader should be introduced in the system. ■

In the following we define the *sibling leader detector* which behaves as Ω? for nodes that are not in the leader state. Nodes that are in the leader state receive *true* if there is another leader in the system or $false$ otherwise. A possible implementation of the sibling leader detector is as follows. A leader detects the presence of another leader, if they interact directly (i.e. they are initiator and/or responder in a interaction), or indirectly, if one of them interacts with probes of the other leader.

Lemma 2 *In the absence of a sibling leader detector there is no deterministic or probabilistic leader-election algorithm in arbitrary graphs even with the help of leader detector* Ω?*.*

Proof: Intuitively the proof goes as follows. Suppose the presence of two leaders and none of them can notice the existence of the other one. Even with the help of leader detector Ω?, for each one it's impossible to decide if it is the only leader or there is another leader.

Consider two configurations, c and c_1, on the same topology: one with two leaders in nodes B and D and the other one with a leader in node B (see Figure 2).

Node B has visibility only on its neighbours and Ω? can notify only if there is at least a leader in the graph, so from its point of view, the two configurations are identical. If B decides to become non-leader, it must do it in both configurations. Two new configurations are obtained: c' and c'_1 and in c'_1 there is no leader. If in c'_1, B and D decide to become leader, since to both of them Ω? returns false, the system returns to a configuration similar to the initial configuration. Overall, even helped by Ω? it's impossible to assert if the leader-election configuration is reached or not. ■

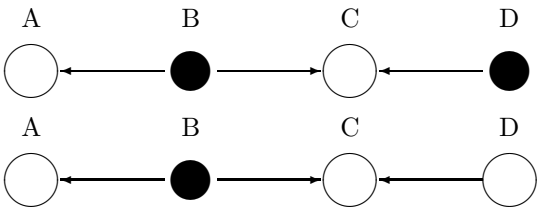

Figure 2: Two initial configurations for Lemma 2

4 Leader Election in Directed Acyclic Graphs

In the following we discuss impossibility results related to leader election in DAGs.

Lemma 3 *If a DAG contains more than one sink node the leader election is impossible when the interactions are conducted only in the sens initiator → receiver.*

Proof: Assume a graph with more than one sink node and two leader marks situated in two different parts of the graph. Since interactions are conducted only from the initiator to receiver, if the DAG contains more than one sink node the two leaders may move towards different sinks of the graph. Once the two leader marks reach the sinks they are blocked. The execution will not verify the leader election behavior. ■

Lemma 2 and 3 show that necessarily in any leader election algorithm leaders should perceive the presence of other leaders. In the following we prove that in general acyclic graphs leader election is impossible even with the help of Ω? and sibling detector.

Lemma 4 *No deterministic leader-election algorithm exists in an general acyclic graph but oriented chains and rooted trees, even with the help of leader detector* Ω? *and the sibling detector.*

Proof: Assume a leader election algorithm for general acyclic graphs, that work under local fairness with help of Ω?. We consider a graph with two leader L_1 and L_2 (see Figure 3). We call the nodes with more than two edges *traffic light*. These *traffic lights* are always red in the direction of one of the two leaders, so one leader cannot enter in the *traffic light* node while the other one can cross all the edges of that node but the red one. In such conditions we show that deterministic leader election algorithm impossible, even under the global fairness assumption. Due to the red light the two leaders never interact. Thanks to the *red/light*, the graph is divided in two parts. By the fairness assumption each leader visits each node of its component. Since the fair scheduler changes the direction of the *red/light* infinitely times, each leader visits each node infinitely often without ever interact with the other leader Figure 3. Moreover the leader detector Ω? become useless because the *red light* works regardless of its indications. Since the two leaders never interact the leader election behavior is never verified. ■

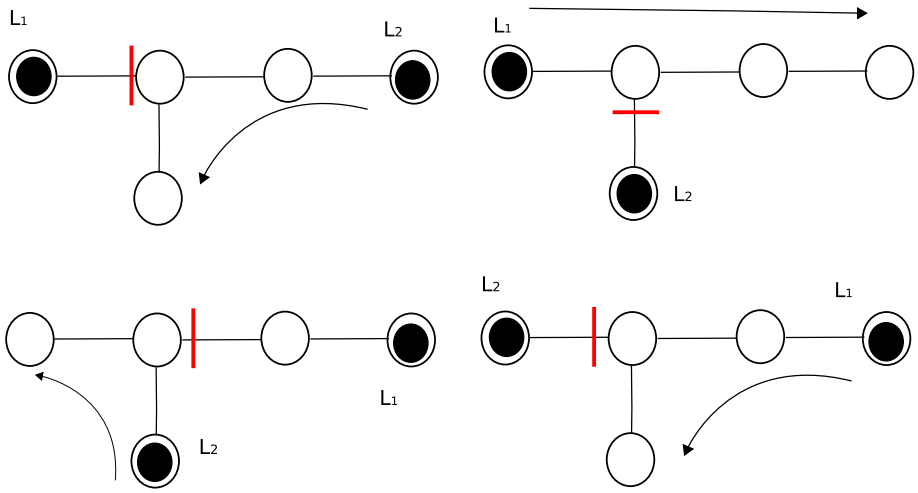

Figure 3: No leader election

5 Leader Election in Rooted Trees

In this section we propose a deterministic and silent solution for leader election in population protocols with rooted tree topology. Our solution is uniform (all nodes execute the same algorithm) and space optimal.

5.1 Data Structures

In the following we borrow the notation used in [9]. Each node has a memory slot that can hold either a leader mark '♠' or nothing '-' for a total of two states and each node receives its current input true (T) or false (F) from Ω?.

5.2 Algorithm Description

Intuitively the algorithms works as follows. A non-leader becomes a leader, when the leader detector signals the absence of a leader, and the responder is not a leader (Rule 2). When two leaders interact, the responder becomes non-leader (Rule 1). If the responder is a leader and the initiator is a non-leader, the latter becomes leader and the former becomes non-leader (Rule 3). Otherwise, no state change occurs. Each node outputs L when it holds a ♠ , otherwise it outputs N. Note that the wild symbol, *, is used to replace any value.

Definition 4 *A configuration of Algorithm 5.1 is legitimate if and only if only one node is leader and the leader is positioned at the root of the tree.*

Rule 1. $((\spadesuit, *), (\spadesuit, *)) \longrightarrow ((\spadesuit), (-))$
Rule 2. $((-, F), (-, *)) \longrightarrow ((\spadesuit), (-))$
Rule 3. $((-, *), (\spadesuit, *)) \longrightarrow ((\spadesuit), (-))$

Algorithm 5.1: Leader election in Rooted Trees

Lemma 5 *Let e be an execution of Algorithm 5.1 starting from an arbitrary configuration. Then* e *converges to a legitimate configuration in a finite number of steps.*

Proof: Let c is the initial configuration of e and let k $>$ 0 be the number of leaders in c. Assume first there is more than one leader in the system. The following situations can occur.

1. the root node is the responder of no one, so if a leader mark is positioned in the Root, no rule cancel, move or remove it.
2. if there is only one leader (not positioned in the root), thanks to Rule 3, the leader mark at level (j) moves to level (j $-$ 1). Gradually the leader moves towards level 0 which is the root level (see item 1).
3. if a leader mark k_1 interacts with the leader mark k_2 which is responder, thanks to Rule 1, the responder is cancelled.
4. if there are more than one leaders, they all move to the root, according to item 2; if a leader meet another leader, one of them is cancelled according to item 3, otherwise they continue to go towards the root.
5. if a leader is positionated in the root, all the other leaders will move towards it (item 4) and eventually interact with it.
6. according to the items 1 and 3, when the leader in the root, interacts with another leader, the later one is cancelled.

Overall starting from a configuration with $k > 0$ leaders the system reaches a configuration with an unique leader in the root.

Assume there are no leaders in e, so from some point on, every node receives false from the leader detector. By Rule 2 a set of initiators selected by the scheduler declare themselves leaders. If there is only a leader in the root following item 1 this leader will not change. If there is only a leader in a node that is not the root following item 2, this leader moves gradually to the root. If there are $k > 1$ leaders then items from 1 to 6 are applied and eventually only one leader stays in the system. ■

Lemma 6 *A leader cannot to be blocked in a leaf or in an internal node.*

Proof: Assume a leader L in a node and the scheduler never choose it. If there are other leaders, all of them reach the root according to Lemma 5, finally if L is the only leader in a node different from the root, it must be chosen by the scheduler, being the only node for which a Rule of the algorithm is enabled (Rule 1 if L is a neighbour of the root or Rule 2 otherwise). ■

Theorem 1 *Given $\Omega?$, Algorithm 5.1 is a self-stabilizing implementation of the leader-election behavior (LE) correct under an arbitrary scheduler.*

Proof: The proof follows from Lemma 5 and Lemma 6. ■

Lemma 7 *In acyclic graphs with only one sink-node there exists a deterministic silent leader-election algorithm.*

Proof: When the graph is a rooted tree Algorithm 5.1 proposes a self-stabilizing silent and uniform solution for leader election. In graphs with an unique node without sons, the proof is totally symmetric but Rule 3 of Algorithm 5.1 should change to:

Rule 3. $((\spadesuit, *), (-, *)) \longrightarrow ((-), (\spadesuit))$.

■

6 Leader Election in Arbitrary Graphs

Leader election is impossible in cyclic graphs using only local fairness and $\Omega?$ [9]. However, some possibility results have be provided in [3, 9].

Note 1 *In a complete graph it's possible for each leader to detect the presence of the other leaders since each node is linked with all others nodes. An algorithm for leader-election in complete graphs was proposed in [9].*

Note 2 *Note that in order to solve leader election in cyclic graphs [9] uses probes ("bullets") that a leader sends in the network in order to destroy other leaders and assumes global fairness. The algorithm is not silent and not optimal. This algorithm extends the work in [3] which proposes a non-uniform leader election algorithm assuming global fairness.*

In the following we propose a probabilistic algorithm that solves leader election using the leader detector $\Omega?$. The algorithm is memory optimal.

The algorithm works as follows. Each node has a memory slot that can hold a bit that can have two states: the leader mark $\spadesuit$ or $-$. Each node receives as input the current value reported by $\Omega?$. A non-leader becomes leader when there is no leader in the system $\Omega?$=F (Rule 2). If two leaders interact (one of them as initiator and the other as responder), the responder becomes non-leader (Rule 1). If a leader interacts with a non-leader, the leader

mark is moved from the initiator to the responder with a probability of 1/2 (Rule 3) and if the initiator is not-leader and the responder is leader, then the leader mark is moved with probability 1/2 from the latter to the former. Rule 2 introduces leaders when $\Omega?$ reports the absence of leaders. Rule 1 destroys extra leaders. Rules 3 and 4 allow leader marks to travel in the network in order to meet and eventually be destroyed via Rule 1.

$$\textbf{\textit{Rule 1.}} \quad ((\spadesuit, *), (\spadesuit, *)) \longrightarrow ((\spadesuit), (-))$$
$$\textbf{\textit{Rule 2.}} \quad ((-, F), (-, *)) \longrightarrow ((\spadesuit), (-))$$
$$\textbf{\textit{Rule 3.}} \quad ((\spadesuit, *), (-, *)) \longrightarrow \begin{cases} Pr(1/2) & ((-), (\spadesuit)) \\ Pr(1/2) & ((\spadesuit), (-)) \end{cases}$$
$$\textbf{\textit{Rule 4.}} \quad ((-, *), (\spadesuit, *)) \longrightarrow \begin{cases} Pr(1/2) & ((-), (\spadesuit)) \\ Pr(1/2) & ((\spadesuit), (-)) \end{cases}$$

Algorithm 6.1: Optimal probabilistic leader election

Definition 5 (Legitimate configuration) *The system executing Algorithm 6 is in a legitimate configuration if there is only one leader mark in the system and $\Omega?$ outputs true to every node in the system.*

Lemma 8 *A leader mark covers under local fairness assumption infinitely often a virtual ring that includes all nodes in the system.*

Proof: Assume there is a node of the graph that is never visited by the leader mark. Either the leader mark is blocked in a node or the leader mark cycles in a part of the graph. In the first case the leader node is enabled either for the Rule 3 or for the Rule 4. The probability for this node to keep the leader mark infinitely is 0: $lim_{s\to\infty}[(\frac{1}{2})^s]$. In the second case, either the leader mark is pushed back and forth between two nodes or the leader mark travels in a cycle. Both cases are impossible due to the local fairness assumption. ■

Corollary 1 *Two leader marks that cover two virtual rings visit at least one common node.*

Lemma 9 *Let e be an execution of Algorithm 6 starting in a configuration with two leader marks. Eventually, the two leader marks interact under local fairness assumptions.*

Proof: Let p be a node of the system and let L_1 and L_2 two leader marks. Lemma 8 proves that eventually the leader mark L_1 visits p and the leader mark L_2 also visits p. Let d_1 be the smallest distance between L_1 and p and let d_2 be the smallest distance between L_2 and p. Due to fairness assumptions eventually the distance between L_1 and p decreases until L_1 reaches p. L_2 eventually reaches a neighbor of p in a finite number of steps. Meantime, L_1 does not change its position (we assume a bounded scheduler). Once L_1 and L_2 are neighbors, by Rule 1, one of the two leader marks is destroyed. ■

Lemma 10 *Let e be an execution of Algorithm 6 starting in an arbitrary configuration. e converges to a legitimate configuration.*

Proof: Suppose there are no leaders in the initial configuration of e. So from some point on, every node receives false from the leader detector $\Omega?$. By Rule 2 the initiators declare themselves leaders and the system reaches a configuration with one ore more leaders. Starting from this configuration, some non-leader nodes may receive false from their detector and continue to inject leaders but there is a point in the execution from which $\Omega?$ returns true to every node in the system. From this point onward no new leader is injected in the system. Suppose the system in a configuration with more than two leaders and $\Omega?$ returns true to every node in the system. Let k the number of leader marks in this configuration. By Lemma 9 two leaders in this set eventually interact and by Rule 1 one of them disappears. So starting from a configuration with k leader marks in a finite number of steps the number of leaders drops to $k-1$. The process is iterated until the system reaches a legitimate configuration. ■

7 Conclusions and discussions

In this paper we focused on the leader election solutions in population protocol model augmented with the eventual leader detector, $\Omega?$. The eventual leader detector eventually reports the presence or the absence of a leader. In the augmented model several solutions for leader election in rings and complete networks have been proposed mainly in [9, 2, 3]. In this work we have extended the study to trees and arbitrary topologies. We also considered a very weak model of agents: anonymous, uniform and with restricted memory — only one bit of memory per agent. In this model we proposed deterministic and probabilistic solutions for leader election and proved the necessity of the eventual leader detector even in environments helped by randomization. All the proposed algorithms are memory optimal.

An interesting open issue of this work is the deterministic or probabilistic implementation of $\Omega?$ and the minimal memory cost that allows its implementation.

References

[1] Dana Angluin, James Aspnes, Zoë Diamadi, Michael J. Fischer, and René Peralta. Computation in networks of passively mobile finite-state sensors. *Distributed Computing*, pages 235–253, March 2006.

[2] Dana Angluin, James Aspnes, Melody Chan, Michael J. Fischer, Hong Jiang, and René Peralta. Stably computable properties of network graphs. In *DCOSS*, pages 63–74, 2005.

[3] Dana Angluin, James Aspnes, Michael J. Fischer, and Hong Jiang. Self-stabilizing population protocols. In *Principles of Distributed Systems; 9th International Conference,*

OPODIS 2005; Pisa, Italy; December 2005; Revised Selected Papers, volume 3974 of *Lecture Notes in Computer Science*, pages 103–117, December 2005.

[4] Dana Angluin, James Aspnes, Zoë Diamadi, Michael J. Fischer, and René Peralta. Computation in networks of passively mobile finite-state sensors. In *PODC*, pages 290–299, 2004.

[5] Dana Angluin, Michael J. Fischer, and Hong Jiang. Stabilizing consensus in mobile networks. In *DCOSS*, pages 37–50, 2006.

[6] Dana Angluin, James Aspnes, and David Eisenstat. Fast computation by population protocols with a leader. In *DISC*, pages 61–75, 2006.

[7] J. Beauquier, J. Clement, S. Messika, and B. Rozoy. Self-stabilizing counting in mobile sensor networks. Technical report, LRI, 2007.

[8] Carole Delporte-Gallet, Hugues Fauconnier, Rachid Guerraoui, and Eric Ruppert. When birds die: Making population protocols fault-tolerant. In *DCOSS*, pages 51–66, 2006.

[9] Michael Fischer and Hong Jiang. Self-stabilizing leader election in networks of finite-state anonymous agents. In *OPODIS*, pages 395–409, 2006.

[10] Tushar Deepak Chandra, Vassos Hadzilacos, and Sam Toueg. The weakest failure detector for solving consensus. *Journal of ACM*, 43(4):685–722, 1996.

Unité de recherche INRIA Rocquencourt
Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)

Unité de recherche INRIA Futurs : Parc Club Orsay Université - ZAC des Vignes
4, rue Jacques Monod - 91893 ORSAY Cedex (France)
Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)
Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399