On the Power of Oracle Omega? for Self-Stabilizing Leader Election in Population Protocols

Joffroy Beauquier, Peva Blanchard, Janna Burman, Oksana Denysyuk

To cite this version:

Joffroy Beauquier, Peva Blanchard, Janna Burman, Oksana Denysyuk. On the Power of Oracle Omega? for Self-Stabilizing Leader Election in Population Protocols. 2014. hal-00839759v2

HAL Id: hal-00839759

https://hal.science/hal-00839759v2

Submitted on 22 May 2014 (v2), last revised 25 Sep 2016 (v3)

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

On the Power of Oracle $\Omega?$ for Self-Stabilizing Leader Election in Population Protocols

[Regular Paper, Eligible for the Best Student Paper Award]

Joffroy Beauquier[1], Peva Blanchard[⋆1], Janna Burman[1], and Oksana Denysyuk[⋆⋆2]

[1] LRI, Université Paris-Sud, Orsay, France, {joffroy.beauquier, peva.blanchard, janna.burman}@lri.fr
[2] INESC-ID, Instituto Superior Técnico, Universidade de Lisboa, Portugal, oksana.denysyuk@ist.utl.pt

Abstract. This paper considers the fundamental problem of *self-stabilizing leader election* ($SSLE$) in the model of *population protocols*. In this model an unknown number of asynchronous, anonymous and finite state mobile agents interact in pairs. $SSLE$ was shown to be impossible in this model without additional assumptions. This impossibility can be circumvented for instance by augmenting the system with an *oracle* (an external module providing supplementary information useful to solve a problem). Thus, Fischer and Jiang have proposed solutions to $SSLE$, for complete communication graphs and rings, using an oracle $\Omega?$, called the *eventual leader detector*. In this paper, we extend their results.
First, we show that $\Omega?$ is powerful enough to allow solving $SSLE$ also over arbitrary communication graphs of bounded degree. Then, over rings, we show that $\Omega?$ has the minimum necessary "power" (or provides the minimum supplementary information) to solve $SSLE$. We prove this by implementing $\Omega?$ using $SSLE$. This result, together with the previous one, provides an equivalence between $\Omega?$ and $SSLE$. This equivalence also means that $\Omega?$ is the *weakest* (i.e., "necessary" and sufficient) oracle to solve $SSLE$ over rings. Moreover, we conjecture that the result over rings can be extended to regular graphs of bounded diameter. On the negative side, we prove that the equivalence between $\Omega?$ and $SSLE$ does not hold on all graphs; in particular, it does not hold on complete graphs and on graphs of bounded degree.

Keywords: networks of mobile agents, population protocols, self-stabilization, leader election, oracles, weakest oracle

⋆ Ph.D. student; Contact author: LRI, Bât. 650, Université Paris-Sud 11, 91405 Orsay Cedex France
⋆⋆ Ph.D. student

1 Introduction

There are fundamental problems in distributed computing that are subject to impossibility results. The impossibility can be related to the system asynchrony, limited resources, the presence of failures, their type, or other general conditions. For instance, the consensus problem has been shown to be impossible in asynchronous systems with only one crash fault [18]. An elegant approach for circumventing the impossibility of consensus is the abstraction known as *failure detectors* introduced by Chandra and Toueg [13]. A failure detector can be viewed as an oracle, which provides to the system nodes a supplementary information about failures allowing to solve a given problem. When designing an oracle, a fundamental issue is to determine the one that provides the minimum amount of information sufficient to solve the problem. Among the different failure detectors proposed to solve consensus in the conventional asynchronous communication model, the *eventual leader elector* Ω, has been proven to be the *weakest* [12]. Informally, that means that it supplies the minimum supplementary information necessary to obtain a solution.

In this work, we consider a very basic communication model of mobile agents called *population protocols*. It has been introduced for large networks of tiny, anonymous and asynchronous mobile agents communicating in pairs [1]. The network has an unbounded but finite population of agents, each with only $O(1)$ states, implying that the size of the population is unknown to the agents. With such minimal assumptions, the impossibility results are not a surprise. For example, consensus is impossible in such a model even without any crash failure [6]. Another impossibility concerns a problem called *self-stabilizing leader election* ($SSLE$). In this work, we focus on this fundamental problem that is shown to be impossible in many different cases [4,17,5]. *Self-stabilization* [16] is a framework for dealing with transient state-corrupting faults and can be viewed as allowing the system to start from an arbitrary configuration. In other words, a protocol solves a problem in a self-stabilizing way if every feasible execution starting from any initial configuration solves the problem.

The eventual leader elector Ω of Chandra and Toueg and other classical failure detectors cannot be used with population protocols, because they assume that the network nodes have unique identifiers, unavailable to anonymous agents in population protocols. Many other previous oracles, like those proposed for anonymous models (e.g., [9]), cannot be used in population protocols either, because of the memory constraints imposed by the model (see a survey in [6]).

To deal with this issue, Fischer and Jiang introduced a new type of oracle, called the *eventual leader detector* [17] and denoted by $\Omega?$. Instead of electing a leader, like Ω, $\Omega?$ simply reports to each agent an (eventually correct) estimate about whether or not one or more leaders are present in the network (see Sec. 2 for a formal definition). This oracle does not require unique identifiers and has additional drastic differences. One of the important differences is motivated by the self-stabilizing nature of the $SSLE$ problem considered in [17]. While Ω is designed to circumvent impossibility related to crash faults, $\Omega?$ is designed to deal with state-corrupting faults. Thus, while Ω is related to a failure pattern and is independent of the protocol using it, $\Omega?$ interacts with the protocol, providing information related to the system configurations reached during the execution. With $\Omega?$, there is some sort of feedback loop: the outputs of the oracle influence the protocol; and conversely, the protocol influences the outputs of the oracle. Yet, there are some features in common with Ω. Both Ω and $\Omega?$ are unreliable in the sense that $\Omega?$ can make errors, that is, to give false information at some point and at some agents, and is only required to eventually provide correct answers, similarly to Ω. Finally, such weak guarantees allow both Ω and $\Omega?$ to be implemented in practice using timeouts and other features often found in real systems (more details about the implementation of $\Omega?$ can be found in [17]; about Ω, in [13]).

To demonstrate the power of $\Omega?$, [17] gives a *uniform* solution to $SSLE$ using $\Omega?$ in complete communication graphs and rings. Uniform means that the solution is independent of the size of the network and the agents do not need to know the network size. Our focus here is on uniform solutions too.[1]

Contributions. In this work, we are interested in investigating more about the capabilities and particularities of $\Omega?$. First, we extend the result of [17] for rings to arbitrary connected communication graphs of bounded degree (by showing that $SSLE$ admits a solution using $\Omega?$ over this family of graphs; see Sec. 4). This extension is more intricate than the solution for rings.
Second, we show that $SSLE$ allows to implement $\Omega?$ on rings (Sec. 5.1). Coupled with our previous result or with the result of [17], this implies that, $\Omega?$ is equivalent to $SSLE$ over rings. In our framework, this also means that $\Omega?$ is the weakest for solving $SSLE$ over this family of graphs.
Then, in contrast with the previous case, we show that over arbitrary communication graphs of bounded degree, $SSLE$ is strictly weaker than $\Omega?$ (see Sec. 5). Intuitively, our results means that, whereas $SSLE$ and $\Omega?$ are not

[1] This is in contrast to the non-uniform solutions given to $SSLE$ over rings in [4] that does not use oracles.

equivalent over certain families of graphs, this difference disappears on rings due to the strong communication constraints imposed by the ring structure. Finally, we conjecture that the result on rings can be extended and we sketch this extension for regular graphs (i.e. of constant degree) with bounded diameter (Sec. 6).

For modeling oracles and problems, and obtaining relations between them, we use the formal framework proposed in [5] and adapted to population protocols (see Sec. 2.2).[1] In this framework, there is no difference between an oracle and a problem, so the relations we exhibit can equivalently be viewed as relations between oracles or between problems. Note that the framework and our results concern an extremely general class of oracles.

Related Work. Being an important primitive in distributed computing, leader election has been extensively studied in various other models, however much less in population protocols. Because of model differences, previous results do not directly extend to the model considered here. For surveys on these previous results in other models, refer to [4,17]. In the following, we mention only the most relevant works to $SSLE$ in population protocols.
It was shown, e.g. in [2,8], that fast converging population protocols can be designed using an initially provided unique leader. Moreover, many self-stabilizing problems on population protocols become possible given a leader (though together with some additional assumptions, see, e.g., [4,7]). Nevertheless, $SSLE$ is impossible in population protocols over general connected communication graphs [4]. Yet, [4] presents a non-uniform solution for $SSLE$ on rings. A uniform algorithm for rings and complete graphs is proposed in [17], but uses $\Omega?$. Recently, [10] showed that at least n agent states are necessary and sufficient to solve $SSLE$ over a complete communication graph, where n is the population size (unavailable in population protocols). For the enhanced model of *mediated population protocols* (MPP) [19], it is shown in [20] that $(2/3)n$ agent states and a single bit memory on every agent pair are sufficient to solve $SSLE$. It is also shown that there is no MPP that solves $SSLE$ with constant agent's state and agent pair's memory size, for arbitrary n. In [11], versions of $SSLE$ are considered assuming $\Omega?$ together with different types of *local fairness* conditions, in contrast with the original population protocols' *global fairness* that is also assumed in the current paper (see Sec. 2).
In [5], it is shown that the difficulty in solving $SSLE$ in population protocols indeed comes from the requirement of self-stabilization, by giving a solution without oracle for arbitrary graphs with a uniform initialization. Then, again, over arbitrary graphs, [5] proposes a protocol for $SSLE$. However, this protocol uses an oracle which is a composition of two copies of $\Omega?$ (and thus is at least as strong as one copy of $\Omega?$ used in this paper). Finally, [5] shows that $SSLE$ and $\Omega?$ are not equivalent over complete communication graphs.

2 Model and Definitions

2.1 Population Protocol

We use here the same definitions as in [1,4,17] with some slight adaptations. A *communication graph* is a directed graph $G = (\mathcal{V}, \mathcal{E})$ with n vertices. Each vertex represents a *finite*-state sensing device called an *agent*, and an edge (u, v) indicates the possibility of a communication (interaction) between u and v in which u is the *initiator* and v is the *responder*. The orientation of an edge corresponds to this asymmetry in the communications. In this paper, every graph is weakly connected.

A *population protocol* $\mathcal{A}(\mathcal{Q}, X, Y, Out, \delta)$ consists of a finite state space $\mathcal{Q}$, a finite input alphabet X, a finite output alphabet Y, an output function $Out : \mathcal{Q} \to Y$ and a transition function $\delta : (\mathcal{Q} \times X)^2 \to \mathcal{P}(\mathcal{Q}^2)$ that maps any tuple (q_1, x_1, q_2, x_2) to a non-empty (finite) subset $\delta(q_1, x_1, q_2, x_2)$ in $\mathcal{Q}^2$.[2] A *(transition) rule* of the protocol is a tuple $(q_1, x_1, q_2, x_2, q_1', q_2')$ s.t. $(q_1', q_2') \in \delta(q_1, x_1, q_2, x_2)$ and is denoted by $(q_1, x_1)(q_2, x_2) \to (q_1', q_2')$. The protocol $\mathcal{A}$ is *deterministic* if for every tuple (q_1, x_1, q_2, x_2), the set $\delta(q_1, x_1, q_2, x_2)$ has exactly one element.

A *configuration* is a mapping $C : \mathcal{V} \to \mathcal{Q}$ specifying the states of the agents in the graph, and an input assignment is a mapping $\alpha : \mathcal{V} \to X$ specifying the input values of the agents. An *input trace* T is an infinite sequence $T = \alpha_1 \alpha_2 \ldots$ of input assignments. It is *constant* if $\alpha_1 = \alpha_2 = \ldots$. An input trace can be viewed as the sequence of input values given to the agents from the outside environment.

We now define agents' interactions (called here *actions*) involving the input values. An *action* is a pair $\sigma = (e, r)$ where r is a rule $(q_1, x_1)(q_2, x_2) \to (q_1', q_2')$ and $e = (u, v)$ is a directed edge of G, representing a meeting of two interacting agents u and v. Let C, C' be configurations, α be an input assignment, and u, v be distinct agents. We say that σ is enabled in (C, α) if $C(u) = q_1, C(v) = q_2$ and $\alpha(u) = x_1, \alpha(v) = x_2$. We say that (C, α) *goes*

[1] In [17], where $\Omega?$ has been introduced, the oracle is defined in a rather informal way.

[2] The input alphabet can be viewed as the set of possible values given to the agents from the outside environment, like sensed values, output values from another protocol or from an oracle. The output alphabet can be viewed as the set of values that the protocol itself outputs outside. X and Y are both the interface values of the protocol.

to C' *via* σ, denoted $(C,\alpha) \xrightarrow{\sigma} C'$, if σ is *enabled* in (C,α), $C'(u) = q'_1, C'(v) = q'_2$ and $C'(w) = C(w)$ for all $w \in \mathcal{V} - \{u, v\}$. In other words, C' is the configuration that results from C by applying the transition rule r to the pair e of two interacting agents. We write $(C,\alpha) \rightarrow C'$ when $(C,\alpha) \xrightarrow{\sigma} C'$ for some action σ. Given an input trace $T_{in} = \alpha_0\alpha_1\ldots$, we write $C \xrightarrow{*} C'$ if there is a sequence of configurations $C_0C_1\ldots C_k$ s.t. $C = C_0$, $C' = C_k$ and $(C_i,\alpha_i) \rightarrow C_{i+1}$ for all $0 \le i < k$, in which case we say that C' is *reachable* from C given the input trace T_{in}.

A *virtual execution* is a maximal sequence of configurations, input assignments and actions (C_0,α_0,σ_0) $(C_1,\alpha_1,\sigma_1)\ldots$ such that for each i, $(C_i,\alpha_i) \xrightarrow{\sigma_i} C_{i+1}$. Virtual executions represent all the computations of a population protocol, but, as in [1,4,17], we consider here only fair executions. A virtual execution $(C_0,\alpha_0,\sigma_0)(C_1,\alpha_1,\sigma_1)\ldots$ is *globally fair*, and is called *execution*, if, for every C, C', α s.t. $(C,\alpha) \rightarrow C'$, if $(C,\alpha) = (C_i,\alpha_i)$ for infinitely many i, then $C' = C_j$ for infinitely many j. This definition together with the finite state space assumption, implies that, if in an execution there is an infinitely often reachable configuration, then it is infinitely often reached [3]. Global fairness can be viewed as an attempt to capture the randomization inherent to real systems, without introducing randomization in the model.

The output function $Out : \mathcal{Q} \rightarrow Y$ is extended from states to configurations and produce an *output assignment* $Out(C) : \mathcal{V} \rightarrow Y$ defined as $Out(C)(v) = Out(C(v))$, given a configuration C. The *output trace* associated to the execution $E = (C_0,\alpha_0,\sigma_0)(C_1,\alpha_1,\sigma_1)\ldots$ is given by the sequence $T_{out} = Out(C_0)Out(C_1)\ldots$. In the sequel, we use the word *trace* for both input and output traces.

2.2 Behaviour, Oracle, Problem and Implementation

The definitions below are adopted from [5] and different from the ones in [4,17]. They are required to obtain a proper framework for defining oracles and establishing relations between them and/or between problems. In particular, this framework is real time independent, which in turn provides self-implementable oracles, in contrast with the traditional failure detectors [14,15]. In short, in this framework, we define a general notion of *behaviour*, which is a relation between input and output traces. A problem and an oracle are defined as behaviours. Then, to compare behaviours, we define a partial order relation using an abstract notion of *implementation* by a population protocol *using* a behaviour.

In the following, a communication graph G is supposed to be fixed and the references to it are implicit, except when mentioning G is particularly relevant.

A *schedule* is a sequence of edges (meetings). An input or an output trace $T = \alpha_0\alpha_1\ldots$ is said to be *compatible* with the schedule $S = (u_0,v_0)(u_1,v_1)\ldots$ if, for every meeting i, for every agent w different from u_i and v_i, $\alpha_i(w) = \alpha_{i+1}(w)$. That is, any two consecutive assignments of a compatible trace can differ only on the values of the two meeting (neigboring) agents. Note that the output trace (associated with an execution with a schedule S) is necessarily compatible with S by definition. For the input traces, we consider only the compatible ones. Such a compatibility between an input trace and a schedule is required for the protocol to be able to detect the changes in the input. This and the output trace compatibility, allow, in turn, to obtain a real-time independent framework and self-implementability mentioned above.

A *history* H is a couple (S,T) where S is a schedule and T is a trace compatible with S. Depending on the type of the trace, a history can be either an input or an output history. A *behaviour* B over a family of graphs $\mathcal{F}$ is a function that, for a graph $G \in \mathcal{F}$ and a schedule S on G, maps every input history H_{in} with schedule S to a set $B(H_{in})$ of output histories with the same schedule S.

Behaviours can be composed. For instance, in the serial composition, an output trace of a behaviour is the input trace of another one. Consider two behaviours B_1, B_2 over the same family $\mathcal{F}$ of graphs, with input alphabets X_1, X_2 (for the input traces), and output alphabets Y_1, Y_2 (for the output traces). In the following paragraphs, we denote by T_Z a trace with values in Z. Let S be a schedule on $G \in \mathcal{F}$.
Thus, if $Y_1 = X_2 = Z$, the *serial composition* $B = B_2 \circ B_1$ is the behaviour over $\mathcal{F}$, with alphabets X_1, Y_2 s.t. $(S,T_{Y_2}) \in B(S,T_{X_1})$ iff there exists a trace T_Z compatible with S, s.t. $(S,T_Z) \in B_1(S,T_{X_1})$ and $(S,T_{Y_2}) \in B_2(S,T_Z)$.
The *parallel composition* $B = B_1 \otimes B_2$ is the behaviour over $\mathcal{F}$, with alphabets $X_1 \times X_2$, $Y_1 \times Y_2$ s.t. $(S,T_{Y_1},T_{Y_2}) \in B(S,T_{X_1},T_{X_2})$ iff $(S,T_{Y_1}) \in B_1(S,T_{X_1})$ and $(S,T_{Y_2}) \in B_2(S,T_{X_2})$.
If $X_1 = U \times V$ and $Y_1 = U \times W$, the *self composition* $B = Self_U(B_1)$ *on* U is the behaviour over $\mathcal{F}$, with alphabets V, W, s.t. $(S,T_W) \in B(S,T_V)$ iff there exists a trace T_U compatible with S s.t. $(S,T_U,T_W) \in B_1(S,T_U,T_V)$.

Given a (possibly infinite) set $\mathcal{U}$ of behaviours, a *composition of behaviours in* $\mathcal{U}$ is defined inductively as either a behaviour in the family $\mathcal{U}$, or the parallel, serial or self composition of compositions of behaviours in $\mathcal{U}$. The behaviour B_2 is called a *sub-behaviour of* B_1 if they are defined over the same family of graphs $\mathcal{F}$, and for every graph $G \in \mathcal{F}$, for every history H on G, $B_2(G,H) \subseteq B_1(G,H)$.

Given a population protocol $\mathcal{A}$ with input alphabet X and output alphabet Y, we define the *behaviour* $Beh(\mathcal{A})$ *associated to the protocol* $\mathcal{A}$ as the behaviour with input alphabet X, output alphabet Y s.t. $(S, T_Y) \in Beh(\mathcal{A})(S, T_X)$ iff there exists an execution of $\mathcal{A}$ with schedule S, input trace T_X and output trace T_Y.

A *problem* and an *oracle* are defined as behaviours. A population protocol $\mathcal{A}$ *implements a behaviour* B (resp. *solves a problem* B or *implements an oracle* B) using a behaviour B' over a graph family $\mathcal{F}$ if there exists a composition B^* involving the behaviours B' and $Beh(\mathcal{A})$, s.t. B^* is a sub-behaviour of B.

We say that a behaviour B_1 is *weaker* than a behaviour B_2 over a graph family $\mathcal{F}$, denoted by $B_1 \preccurlyeq_{\mathcal{F}} B_2$, if there exists a population protocol that implements B_1 using B_2 over $\mathcal{F}$. The two behaviours are *equivalent* over $\mathcal{F}$, denoted $B_1 \simeq_{\mathcal{F}} B_2$, if $B_1 \preccurlyeq_{\mathcal{F}} B_2$ and $B_2 \preccurlyeq_{\mathcal{F}} B_1$. In the case where B_2 is a problem and B_1 is an oracle, B_1 is then *the weakest* oracle for implementing B_2 over $\mathcal{F}$. The reason is that, because $B_1 \preccurlyeq_{\mathcal{F}} B_2$, *any* oracle that can be used to implement B_2, can be used to implement B_1, and thus, B_1 is weaker than any such oracle.

Remark 1. Note that, the relation $\preccurlyeq_{\mathcal{F}}$ depends on a population protocol (involved in its definition) and thus on its executions. Since our focus is on self-stabilization under global fairness, the relations we establish between behaviours involve implementations (population protocols) with globally fair executions starting from arbitrary initial configurations. For the sake of clarity, the dependence on the considered type of executions is implicit in the notion of implementation.

3 Specific Behaviours

3.1 Eventual Leader Election Behaviour $\mathcal{ELE}$

$\mathcal{ELE}$ is defined with the input alphabet $\{\perp\}$ (i.e., no input) and the output alphabet $\{0, 1\}$ such that, given a graph G and a schedule S on G, a history $(S, T) \in \mathcal{ELE}(S)$ if and only if the output trace T has a constant suffix $T' = \alpha\alpha\alpha\ldots$ and there exists an agent λ such that $\alpha(\lambda) = 1$ and $\alpha(u) = 0$ for every $u \neq \lambda$. In other words, λ is the unique leader. Notice that for all our protocols, there is an implicit output map that maps a state to 1 if it is a leader state, and to 0 otherwise.

In our framework, the informal problem of Self-Stabilizing Leader Election ($SSLE$) mentioned in the introduction consists in obtaining a population protocol that solves $\mathcal{ELE}$ using another behaviour (if necessary) and starting from arbitrary initial configurations.

3.2 Oracle $\Omega?$

Informally, $\Omega?$ (introduced in [17]) reports to agents whether or not one or more leaders are present. Thus, it does not distinguish between the presence of one or more leaders in a configuration (of a protocol composed with $\Omega?$). This is sufficient, for example, to solve $\mathcal{ELE}$ in complete graphs, because two leaders are always neighbours, and eventually one of them can be eliminated, resulting finally in a unique remaining leader [17]. On rings, a rather elaborated mechanism (together with the global fairness) allows to cancel supplementary leaders, without knowing their number [17].

Formally, $\Omega?$ is defined as follows. The input and output alphabets are $\{0, 1\}$. Given an assignment α, we denote by $l(\alpha)$ the number of agents that are assigned the value 1 by α. Given a graph G and a schedule S on G, $(S, T_{out}) \in \Omega?(S, T_{in})$ if and only if the following conditions hold for input and output traces T_{in} and T_{out}. If T_{in} has a suffix $\alpha_0\alpha_1\ldots$ such that $\forall i, l(\alpha_i) = 0$, then T_{out} has a suffix during which at each output assignment at least one agent is assigned 0. If T_{in} has a suffix $\alpha_0\alpha_1\ldots$ such that $\forall s, l(\alpha_s) \geq 1$, then T_{out} has a suffix equal to the constant trace where each agent is permanently assigned the value 1. Otherwise, any T_{out} is in $\Omega?(S, T_{in})$.

4 $SSLE$ using $\Omega?$ over Graphs with Bounded Degree

In this section, we show that, for any given integer d, the behaviour $\mathcal{ELE}$ can be implemented in a self-stabilizing way using $\Omega?$ over the family of weakly connected graphs with a degree bounded above by d. Precisely, we present a population protocol $\mathcal{A}_d$ and prove that the behaviour given by the composition $Self(Beh(\mathcal{A}_d) \circ \Omega?)$ (see Fig. 1) is a sub-behaviour of $\mathcal{ELE}$. For the sake of clarity, we first give a solution over the family of *strongly connected* graphs with bounded degree. It is possible to transform this solution into one over *weakly connected* graphs with bounded degree (see Theorem 4 in the appendix).

Our protocol is inspired from Fischer and Jiang's protocol [17] for rings. The main design difficulty comes from the fact that the information given by the oracle does not allow to distinguish between the presence of a single or more leaders. Thus, a leader should try to kill possible other leaders, when avoiding a scenario where all leaders are killed infinitely often. This metaphor comes from [17] – leaders sending *bullets* for killing other leaders, and may protect themselves by *shields*. Although the protocol in [17] is not simple, the ring topology is

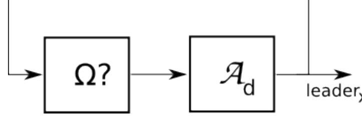

Fig. 1. Serial composition $Beh(\mathcal{A}_d) \circ \Omega?$ followed by a self composition.

of great help. For arbitrary graphs, managing bullets and shields is much more complicated, and agents must in some sense keep a trace of them. As the agents are finite-state, a bounded degree is needed for implementing such a management.

As a basic tool for our protocol, we use the 2-hop coloring self-stabilizing population protocol, denoted by $2HC$, proposed in [4]. A 2-hop coloring is a coloring such that all neighbours of the same agent have distinct colors. We denote by $Colors$ the corresponding set (of size $O(d)$) of possible colors.

4.1 The Protocol $\mathcal{A}_d$ (Algorithm 1)

The input variables (read-only) of $\mathcal{A}_d$ at each agent x are: the *oracle output* $\Omega?_x$ (values in $\{0, 1\}$); and the *agent color* c_x (values in $Colors$), which stores the output of $2HC$. The working variables are: the *leader bit* $leader_x$ (values $\{0, 1\}$); the *bullet vector* $bullet_x$ (vector with values in $\{0, 1\}$ indexed by $Colors$); and the *shield vector* $shield_x$ (vector with values in $\{0, 1\}$ indexed by $Colors$).

The idea of the protocol is the following. An agent may hold several shields (resp. bullets), each of them waiting to be forwarded to an out-neighbour, from initiator to responder, with associated color, lines 14 – 18 (resp. in-neighbour, from responder to initiator, lines 7 – 12). The information required for implementing this is encoded in the shield and bullet vectors. The purpose of the bullets is to kill leaders (line 10), whereas the purpose of the shields is to protect them by absorbing bullets (line 17). A leader is created when the oracle reports that there are no leaders in the system (lines 2, 3). When a leader is created, it comes with (loads) a shield for every color (line 5), and thus is protected from any bullet that could come from one of its out-neighbors. To maintain the protection, each time an agent receives a shield from its in-neighbor, it reloads shields for every color (line 16). Dually, any time an agent receives a bullet, it reloads bullets for every color (line 11). In addition, whenever a leader interacts as an initiator, it loads bullets for every color (line 22).

Algorithm 1: Protocol $\mathcal{A}_d$ - initiator x, responder y

1 (Create a leader at x, if needed)
2 **if** $\Omega?_x = 0$ **then**
3 $\quad leader_x \leftarrow 1$
4 $\quad \forall c \in Colors, bullet_x[c] \leftarrow 1$
5 $\quad \forall c \in Colors, shield_x[c] \leftarrow 1$
6 **end**
7 (Move bullet from y to x, if any)
8 **if** $bullet_y[c_x] = 1$ **then**
9 $\quad$ **if** $shield_x[c_y] = 0$ **then**
10 $\quad\quad leader_x \leftarrow 0$
11 $\quad\quad \forall c \in Colors, bullet_x[c] \leftarrow 1$
12 $\quad bullet_y[c_x] \leftarrow 0$
13 **end**
14 (Move shield from x to y, if any)
15 **if** $shield_x[c_y] = 1$ **then**
16 $\quad \forall c \in Colors, shield_y[c] \leftarrow 1$
17 $\quad bullet_y[c_x] \leftarrow 0$
18 $\quad shield_x[c_y] \leftarrow 0$
19 **end**
20 (Load bullets if x is a leader)
21 **if** $leader_x = 1$ **then**
22 $\quad \forall c \in Colors, bullet_x[c] \leftarrow 1$

4.2 Correctness

Consider a strongly connected graph G of degree (in and out degree together) less than or equal to d. For the sake of clarity, in any execution we consider, we assume that the protocol $2HC$ permanently outputs a correct 2-hop coloring from the beginning (variables c_x, for every agent x).

A *path* in G is a sequence of agents $\pi = x_0 \dots x_r$ such that (x_i, x_{i+1}) is a directed edge of G. If $x_0 = x_r$, π is a *loop at* x_0. If u is an agent that appears in π, we denote it by $u \in \pi$, and by $ind_\pi(u)$ the index of the first occurrence of u in π, i.e. the minimum i such that $x_i = u$. If (x, y) is an edge of G, we say that x *has a shield against* y if $shield_x[c_y] = 1$. Similarly, we say that y *has a bullet against* x if $bullet_y[c_x] = 1$.

Definition 1 (Protected Leader). *Consider a loop $\pi = x_0 \dots x_{r+1}$ at a leader λ $(= x_0 = x_{r+1})$. We say that λ is a* leader protected in π *if there exists $i \in \{0, \dots, r\}$ such that x_i has a shield against x_{i+1} and, if $i \geq 1$, x_i is not a leader and has no bullet against x_{i-1}. In addition, for every $j \in \{1, \dots, i-1\}$, x_j is not a leader, has no shield against x_{j+1} and no bullet against x_{j-1}. The agent x_i is the* protector of λ in π*; the path $x_0 \dots x_i$ is the* protected zone in π. *The agent λ is a* protected leader *if it is protected in every loop at λ.*

Note that a new leader or a leader that receives a shield becomes protected by loading shields for every color.

We use the following notations in the sequel. Given an execution E, S_E denotes the maximum (infinite) suffix of E such that each couple (C, α) (C being a configuration, and α an input assignment) in S_E occurs infinitely often. IRC_E denotes the (finite) set of configurations occurring in S_E, i.e., the set of configurations that occur infinitely often in E.

Lemma 1. *If $C \in IRC_E$ has a protected leader, then every configuration in IRC_E has a protected leader.*

Proof. Consider a couple (C, α) that occurs in S_E, C being a configuration (in IRC_E) and α an input assignment. Let C' be a configuration s.t. (C, α) goes to C' via an action involving a directed edge (x, y). By global fairness, $C' \in IRC_E$ too, and we show that it has a protected leader.

Note that when a leader is created, it is already protected by itself since it has a shield against every of its out-neighbors (line 5). We thus focus on transition rules that do not involve the creation of a leader. Let λ be a protected leader in C and π be any loop at λ. Let μ be the protector of λ in π. If x and y do not appear in the protected zone in π, then after the transition, the states of the agents in the protected zone have not changed and λ is still protected in π. Then, assume that x or y appear in the protected zone and let $z \in \{x, y\}$ be the agent with lowest index $ind_\pi(z)$. By the choice of z, $ind_\pi(z) \leq ind_\pi(\mu)$.

Consider first the case $ind_\pi(z) < ind_\pi(\mu)$. If $z = x$, then z cannot receive a bullet (from y), i.e., either x has a shield against y or y has no bullets against x. Otherwise, the path that goes from λ to (the first occurrence of) $z = x$ followed by any path that goes from y to λ yields a loop within which λ is not protected in C; hence a contradiction. Hence, if $z = x$, after the transition, λ is still protected by μ in π. Now, if $z = y$, y may only receive a shield, and thus, after the transition, λ is still protected in π (by μ or y).

Now, assume that $ind_\pi(z) = ind_\pi(\mu)$. This implies that $z = \mu \in \{x, y\}$, and that every agent in the protected zone, except μ, is different from x and y. If $\mu = y$, then during the transition, μ may only receive a shield (which merges with its own shield); hence, λ is still protected by μ in π after the transition. The case $\mu = x$ is more complicated. First consider the subcase where y is not the agent that follows the first occurrence of μ in π. Then μ cannot receive a bullet during the transition, otherwise, the same argument as above shows the existence of a loop at λ within which λ is not protected in C. After the transition, (the first occurrence of) μ has still a shield against the agent right after it, which proves that λ is still protected in π. Consider now the subcase where y is the agent that follows the first occurrence of μ in π. If y is not a leader, then after the transition, y becomes the new protector of λ in π. If y is a leader, then after the transition, λ is no longer protected, but y is protected since the reception of a shield produces shields for every color. In both cases, after the transition, there is a protected leader in C'.

We thus have shown that, in all cases, C' contains a protected leader. Given any configuration $C'' \in IRC_E$, there must be a sequence of actions from (C, α) to (C'', α'') during S_E, for some input assignment α''. Since C has a protected leader, the proof shows that every configuration in this sequence, and in particular C'', has also a protected leader. Therefore, any configuration C'' in IRC_E has a protected leader. □

Lemma 2. *If no configuration in IRC_E has a leader, then in every input assignment in S_E, $\Omega?_x = 0$ for some agent x. If every configuration in IRC_E has a leader, then in every input assignment in S_E, $\Omega?_x = 1$ for every agent x.*

Proof. This stems from the definition of $\Omega?$. □

Lemma 3. *Every configuration in IRC_E has a leader.*

Proof. Assume that some configuration C in IRC_E lacks a leader. On the one hand, if no configuration in IRC_E has a leader, then by Lemma 2, in every input assignment in S_E, $\Omega?_x = 0$ for some agent x. Hence, in S_E, during a transition involving x, a *protected* leader is created (lines 2 – 5). On the other hand, if IRC_E contains a configuration C' with a leader, then there is a sequence of actions from (C, α) to (C', α') for some input assignments α, α', since both C and C' occur infinitely often in S_E. According to the protocol, during some of these actions, a *protected* leader must be created. In both cases, there is a configuration $C'' \in IRC_E$ with a protected leader. By Lemma 1, this implies that every configurations in IRC_E, and in particular C, has a protected leader; hence a contradiction. □

Lemma 4. *All configurations in IRC_E have the same number of leaders.*

Proof. By the lemmas 2 and 3, in every input assignment in S_E, $\Omega?_x = 1$ for every agent x. Thus no leader is created during S_E. Assume that there exists two configurations C, C' in IRC_E such that the number l of leaders in C is different from the number l' of leaders in C'. Without loss of generality, we can assume $l < l'$. By definition of S_E, there must be a sequence of actions from (C, α) to (C', α') for some input assignments α, α'. The fact that $l < l'$ implies that during this sequence a leader is created; hence a contradiction. □

Lemma 5. *No configuration in IRC_E contains an unprotected leader.*

Proof. By the lemmas 2 and 3, in every input assignment in S_E, $\Omega?_x = 1$ for every agent x. Assume that $C \in IRC_E$ contains an unprotected leader λ. Since λ is not protected in C, there exists a path $\pi = x_0 \ldots x_r$ from $x_0 = \lambda$ to some agent x_r such that for every $0 \leq i < r$, x_i has no shield against x_{i+1}, and x_r is either a leader or has a bullet against x_{r-1}. If x_r is a leader, then in any transition where it is the initiator, it creates a bullet against x_{r-1}. Thus, in both cases, there is a bullet that, by moving (backward) along this path to λ, can kill this non-protected leader. Thus, a configuration C' within which λ is not a leader is reachable from C. During the sequence of actions from C to C', no leaders are created. Thus, C' has fewer leaders than C. The global fairness ensures that $C' \in IRC_E$. This contradicts Lemma 4. □

Theorem 1. *The protocol $\mathcal{A}_d$ solves the problem $\mathcal{ELE}$ using $\Omega?$ (i.e., $\Omega? \succcurlyeq \mathcal{ELE}$) over strongly connected graphs with degree less than or equal to d.*

Proof. By the previous lemmas, every configuration $C \in IRC_E$ has $l \geq 1$ protected leaders and no unprotected leaders; and in every input assignment in S_E, $\Omega?_x = 1$ for every agent x. Assume, by contradiction, that $l \geq 2$. Let λ_1, λ_2 be two protected leaders in C. Consider a shortest path p_1 (resp. p_2) from λ_1 to λ_2 (resp. from λ_2 to λ_1). Consider the loop $\pi_1 = p_1 p_2$ at λ_1, and the loop $\pi_2 = p_2 p_1$ at λ_2. Denote by μ_1 (resp. μ_2) the protector λ_1 (resp. λ_2) in π_1 (resp. π_2). By construction, in C, the first occurrence of μ_1 (resp. μ_2) is in p_1 (resp. p_2). By definition and according to the protocol, it is possible to move the (first occurrence of the) protector μ_1 to the position right before λ_2. Another movement makes the protector transfer its shield to λ_2, thus turning λ_1 into a non-protected leader (λ_2 is still a protected leader). Then λ_2 can fire a bullet that kills λ_1. Since no leader is created during the described sequence of actions ($\Omega?_x = 1$ for every agent x), the reached configuration C' has $l - 1$ leaders. As global fairness ensures that $C' \in IRC_E$, this contradicts Lemma 4. Therefore, all configurations in IRC_E have a unique leader. Since a leader cannot move, there is a permanent leader. □

5 Is $\Omega?$ the Weakest Oracle for Solving $SSLE$?

We answer this question when considering different communication topologies. For rings, the answer is positive. It is presented in the following sub-section 5.1. In Sec. 6, we give a sketch of how to extend the result on rings to regular graphs with bounded diameter. For some other topologies, a negative answer (Corollary 1) follows from Theorem 2 below. Somewhat similar theorem and its proof, for the case of complete graphs, has been presented in our previous work [5]. Here we present a more general result that applies to infinite families of graphs, called here *non-simple* (like in [4]). A family $\mathcal{F}$ is non-simple if there is $G \in \mathcal{F}$ such that two disjoint subgraphs of G are also in $\mathcal{F}$. Complete and arbitrary graphs of bounded degree are some examples of non-simple families of graphs. In contrast, notable *simple* families of graphs include rings, or, more generally, connected regular graphs.

The following theorem 2 proves the impossibility of a self-stabilizing implementation of $\Omega?$ using $\mathcal{ELE}$ over any non-simple family of graphs (for definitions, see Sec. 2.2 and remark 1). Coupled with the result of Sec. 4, i.e. $\Omega? \succcurlyeq \mathcal{ELE}$ over connected arbitrary graphs of bounded degree, we have that $\Omega? \succ \mathcal{ELE}$ over the same graph family, when self-stabilizing implementation is concerned. Thus, by definition, $\Omega?$ is not the weakest oracle for $SSLE$, over connected arbitrary graphs of bounded degree. Similarly, $\Omega?$ is not the weakest oracle for $SSLE$ over complete graphs (and $\Omega? \succ \mathcal{ELE}$), following the same theorem 2 coupled with the $SSLE$ protocol over complete graphs given in [17]. This is summarized in Corollary 1 below. The proof of Theorem 2 uses a classical *partitioning argument*. Due to the lack of space, it has been moved to the appendix.

Theorem 2. *For any non-simple family of graphs $\mathcal{F}$, there is no self-stabilizing population protocol A implementing $\Omega?$ over $\mathcal{F}$ using the behaviour $\mathcal{ELE}$.*

Corollary 1. *$\Omega? \succ \mathcal{ELE}$ over complete graphs and over arbitrary connected graphs of bounded degree. Moreover, $\Omega?$ is not the weakest oracle to solve $\mathcal{ELE}$ over these two families, and more generally, over any non-simple family of graphs.*

5.1 $\Omega?$ is the weakest oracle for $SSLE$ over rings

In this section, we show that $\Omega?$ can be implemented in a self-stabilizing way given the behaviour $\mathcal{ELE}$ over oriented rings. This implementation is given by the *RingDetector* protocol presented below (see Algorithm 2). These results are straightforward to extend to non-oriented rings thanks to the self-stabilizing ring orientation protocol presented in [4]. The meaning of this result, coupled with the result of Sec. 4, is that $\Omega? \simeq_{rings} \mathcal{ELE}$, when self-stabilization is concerned. This also implies that $\Omega?$ is the weakest oracle.

Implementing $\Omega?$ by the *RingDetector* protocol using $\mathcal{ELE}$ (Algorithm 2)

For the sake of clarity, the unique leader provided by $\mathcal{ELE}$ is called the *master*, whereas the output of $\Omega?$ reports about the *leaders*. Hence, the goal consists in the master detecting the presence or the absence of leaders in the graph, that is to mimic $\Omega?$.

Let us define the self-stabilizing protocol *RingDetector*. The input variables (read-only) at each agent x are: the *master bit* $master_x$ (values in $\{0,1\}$) that keeps the output of $\mathcal{ELE}$; and the *leader bit* $leader_x$ (values in $\{0,1\}$), which represents the input of $\Omega?$. The working variables are: the *probe field* $probe_x$ (with values: $\bot$ - no probe, or 0 - white probe, or 1 - black probe); the *token field* (with values: $\bot$ - no token, or 0 - white token, or 1 - black token); the *flag bit* $flag_x$ (with values: 0 - cleared, 1 - raised); and the *output bit* (values in $\{0,1\}$), which represents the corresponding output of $\Omega?$.

Each time an agent has its leader bit set to 1, it raises its flag (and the flag of the other agent in the interaction) – line 5. A token moves clockwise, and its purpose is to detect a leader (actually, a raised flag) and to report it to the master (lines 18 –26). A probe moves counter-clockwise, and its purpose is to report to the master the lack of tokens (lines 7 – 13). The master loads a white probe each time it is the responder of an interaction (line 2). When a probe meets a token, the probe becomes black (line 10). When two probes meet, they merge into a black probe if one of them was black, and into a white probe otherwise (line 12). The master loads a token colored with its flag only when it receives a white probe (line 17). Each time a token meets an agent with its flag raised, the token becomes black (line 21) and the flag is cleared (line 25). Two meeting tokens merge into a black token if one of them is black, and into a white token otherwise (line 23). When the master receives a token, it whitens the token, and it outputs 0 if the token is white, and 1 otherwise (lines 28 –31). In any interaction, the responder copies the output of the initiator, unless the responder is the master (line 33).

Algorithm 2: Protocol *RingDetector* - initiator x, responder y

1 (if the master is the responder, it creates a white probe)
2 **if** $master_y = 1$ **then** $probe_y \leftarrow 0$
3
4 (raise flags if needed)
5 **if** $leader_x \vee leader_y$ **then** $flag_x \leftarrow flag_y \leftarrow 1$
6
7 (move probe from y to x)
8 **if** $probe_y \neq \bot$ **then**
9   (the probe becomes black when meeting a token)
10   **if** $token_x \neq \bot$ **then** $probe_x \leftarrow 1$
11   otherwise, keeps the same color or merges)
12   **else if** $probe_x \in \{\bot, 0\}$ **then** $probe_x \leftarrow probe_y$
13     $probe_y \leftarrow \bot$
14 **end**
15
16 (if the master receives a white probe, it loads a token)
17 **if** $master_x = 1$ *and* $probe_x = 0$ **then** $token_x \leftarrow flag_x$
18 (move token from x to y)
19 **if** $token_x \neq \bot$ **then**
20   (the token becomes black when meeting a flag)
21   **if** $flag_y = 1$ **then** $token_y \leftarrow 1$
22   (otherwise, keeps the same color or merges)
23   **else if** $token_y \in \{\bot, 0\}$ **then** $token_y \leftarrow token_x$
24   (the flag is cleared)
25     $flag_y \leftarrow 0$
26     $token_x \leftarrow \bot$
27 **end**
28 (if the master receives a token, it changes its output and whitens the token)
29 **if** $master_y = 1$ *and* $token_y \neq \bot$ **then**
30     $out_y \leftarrow token_y$
31     $token_y \leftarrow 0$
32 (a non-master responder copies the output of the initiator)
33 **if** $master_y = 0$ **then** $out_y \leftarrow out_x$

Correctness

In the following, the input trace $T = \alpha_0\alpha_1\ldots$ of every execution E is assumed to provide a unique master, i.e., there exists a unique agent λ in E such that $\alpha_i(\lambda).master = 1$ for all i.[1] By the definition of $\mathcal{ELE}$ and *RingDetector*, such an input trace exists in an infinite suffix of every E of *RingDetector*. For the correctness proof, we focus only on such suffixes, for every execution.

The leader bit component in the input trace represents the input of $\Omega?$. In this trace, leaders can appear or disappear almost freely, during each meeting. In particular a leader can "jump" from u to v during an interaction

[1] We precise the notations. α being an assignment (resp. C a configuration), $\alpha.v$ (resp. $C.v$) is the projection of α (resp. C) on the variable v; and $\alpha(x).v$ (resp. $C(x).v$) is the value of this projection at agent x.

between u and v. Though, a leader cannot "jump" to a distant (non neighbouring) agent on the ring, by the compatibility of an input trace with a schedule (see Sec. 2.2). The fact that a leader can "jump" counter-clockwise from the responder to the initiator introduce some subtleties in *RingDetector*. Without taking care, such a "jumping" leader could be undetectable. To ensure its detection, the flag bits of both the responder and the initiator are raised, even if the leader is detected only at one of the two interacting agents (line 5).

We use below the same notations S_E and IRC_E defined in Sec. 4.

Lemma 6. *For any execution E, in any configuration of IRC_E, there is a unique agent holding a token (black or white).*

Proof. Consider a configuration $C \in IRC_E$. We first prove that in C at least one agent holds a token. By contradiction, assume that, for every agent x, $C(x).token = \bot$. The following scenario will produce a token. First, the master λ interacts as a responder and produces a white probe at λ. Then, all the other probes move (counter-clockwise) to the master. Then the white probe at λ visits all agents and returns to λ. Since there are no tokens in the graph, the white probe does not turn black. Then, the white probe arriving at λ produces a token (line 17). This scenario does not depend on the presence of leaders. Hence, there exists a configuration C' with at least one token such that $C \xrightarrow{*} C'$, for any input trace. By global fairness, $C' \in IRC_E$. Together with that, no rule of the protocol can remove all tokens. In line 26, the token is removed from an initiator x, but is present or created in the responder y (line 23). No other instruction removes a token. Thus C cannot occur infinitely often; hence a contradiction. Hence, in C at least one agent holds a token.

Assume now that C has at least two tokens. Since two meeting tokens merge into one, there is a configuration C' with exactly one token such that $C \xrightarrow{*} C'$, for any input trace. By global fairness, C' belongs to IRC_E. Since C also occurs infinitely often in the execution, $C' \xrightarrow{*} C$, for any input trace. To reach C, a token should be created. It can happen only if the master receives a white probe. Thus, the master should receive infinitely many white probes during S_E. However, once there is a token, since the tokens move clockwise and the probes counter-clockwise, any probe arriving at the master must be black; hence a contradiction. Therefore, C has exactly one token. □

Thus, in the suffix S_E, there is a unique token moving clockwise. We divide S_E into *rounds*, defined as follows. A *round* begins with an interaction in which the master holds the token and is the initiator; the round ends with the first event in which the master is the responder and the initiator holds the token. In other words, a round corresponds to the token traveling around the whole ring starting and ending at the master.

Lemma 7. *Let R be a round in S_E. We denote by $(C_0, \alpha_0) \ldots (C_r, \alpha_r)$ the sequence of configurations and input assignments corresponding to R. Case (a) If there are no leaders in R (i.e., for every $0 \leq i \leq r$, and every agent x, we have $\alpha_i(x).leader = 0$), then after the last action in R, all the agents have their flags cleared (set to 0). Case (b) If there are no leaders in R, and if all agents have their flags cleared at the beginning of the round, then at the end of the round, the master outputs 0 and all agents have their flags cleared. Case (c) If there is at least one leader at each assignment α_i during the round, i.e., for every $0 \leq i \leq r$ there is some agent x_i such that $\alpha_i(x_i).leader = 1$, then at the end of the round, the master outputs 1.*

Proof. The proofs of cases (a) and (b) are relatively simple. They have been moved to the appendix.

Case (c). Assume that there is a leader at each assignment during the round. Let μ be a leader agent in the assignment α_0, i.e., $\alpha_0(\mu).leader = 1$. During the round, there must be some action i, such that $\mu = v_i$ is the responder, and the initiator u_i holds the token. If μ is a leader in an assignment α_i, then after the transition, the token turns black. If μ is not a leader, in assignment α_i, since μ is a leader in the assignment α_0, there must be some action $j < i$ such that $\alpha_j(\mu).leader = 1$ and $\alpha_{j+1}(\mu).leader = 0$. Now, since the input trace is compatible with the schedule, μ must be the initiator u_j or the responder v_j in the transition $(C_j, \alpha_j) \to C_{j+1}$. Hence, μ must raise a flag in both the responder v_i and the initiator u_i (line 5), i.e., we have $C_{j+1}(\mu).flag = 1$ $(j+1 \leq i)$. Recall that there is a unique token, so the flag cannot be cleared during the remaining actions until i. Hence, at action i, the token turns black (line 21) when the token moves from the initiator u_i to the responder $v_i = \mu$. In all cases, the master receives a black token at the end of the round, and thus outputs 1. □

Theorem 3. *The protocol RingDetector is a self-stabilizing implementation of $\Omega?$ using $\mathcal{ELE}$ (i.e., $\mathcal{ELE} \succcurlyeq \Omega?$) over oriented rings.*

Proof. Consider a globally fair execution E and focus on the suffix S_E. By Lemma 6, in S_E, there is a unique token moving clockwise. Let $S_E = \ldots R_1 R_2 \ldots R_i \ldots$, where each R_i is a round.

Consider first the case where the input trace $T = \alpha_0\alpha_1\ldots$ in S_E permanently assigns no leader everywhere, i.e., for every i, for every agent x, $\alpha_i(x).leader = 0$. By Lemma 7, at the end of R_1, all flags are cleared. Hence, at the end of R_2, the master outputs 0 and all flags are cleared. By iteration, at the end of each round R_i, $i \geq 2$, the master outputs 0. Since the master updates its output only when it receives the token, and since this happens exactly at the end of a round, in the suffix $R_2R_3\ldots$, the master permanently outputs 0. The fact that the responder always copies the output of the initiator (unless the responder is the master) implies that there is a suffix during which all agents permanently output 0.

Assume now that the input trace in S_E is such that there is at least one leader at every input assignment. By Lemma 7, at the end of each R_i, the master outputs 1. The same argument as above shows that there is a suffix of execution during which all agents permanently output 1.

Note that, in the remaining cases of input traces in S_E, that is when there are input assignments with a leader and some other without, nothing has to be proven, because then, the output of $\Omega?$ is arbitrary. □

Remark 2. Note that a simpler solution managing only tokens, sent periodically by the master, and without managing any probes, would not be correct. To see this, consider an input trace where there is one leader in every input assignment, but this leader moves repeatedly clockwise, "jumping" from one agent to its successor on the ring. By the definition of $\Omega?$, in this scenario, the master should eventually and permanently output 1. However, it is infinitely often possible that there are two tokens directly following the leader one after the other, during the whole tour, from the master to the master. In this case, the first token arriving at the master is black, but the following token is white. This is because the first token has cleared every flag raised by the leader. The repetition of this scenario causes an oscillation of the output of the master between 0 and 1.

The corollary below follows from theorems 3 and 1, the ring orientation protocol of [4], and the definition of the weakest oracle.

Corollary 2. *$\Omega? \simeq_{rings} \mathcal{ELE}$, i.e., $\Omega?$ is the weakest oracle to solve $\mathcal{ELE}$ over rings.*

6 Perspectives

This work opens many questions about the solvability of $SSLE$ in population protocols. What are other additional families of graphs over which $SSLE$ can be solved with the help of $\Omega?$? For example, is $\Omega?$ still sufficient over connected arbitrary graphs (non-simple family; cf. Sec. 5) or over connected regular graphs (simple family). If not, what can be sufficient oracles, and the most interestingly, which of those are necessary?

We sketch below an answer for some of these questions. Specifically, we conjecture that the result on rings can be extended to other simple families of graphs and we present here a possible extension. Consider the family of regular graphs (of constant degree) with bounded diameter. We claim that, over this family, $\Omega?$ is equivalent to (or is the weakest oracle for solving) $SSLE$. Note that the protocol given in Sec. 4 is applicable also in this case, so it remains to show that $\Omega?$ can be implemented given $SSLE$. We sketch below the corresponding protocol, called *TreeDetector*.

This protocol uses two other protocols from [4]. One performs a two-hop coloring and, another one builds a spanning tree, with the master at the root (the latter protocol requires constant degree and bounded diameter). Protocol *TreeDetector* uses the spanning tree structure to detect the leaders in the input. It is necessarily more complicated than *RingDetector*, because now, during a meeting, a leader can "jump" all over the graph, and not only over the edges of a tree, used for traversal (in the ring case, all the edges are used for traversal).

Like in *RingDetector* the presence of a leader raises a flag. The flag detection is made by waves of tokens that start at leaves and go upward the tree, to the master. Each agent knows the number of its children and propagates the wave only after having met all of them (then, if some leaf generates tokens faster than another leaf, they are blocked somewhere). In particular, the master updates its output only when it has met all its children. This provides a sort of synchronization mechanism allowing to have something analogous to the rounds of *RingDetector*.

A key point is that, during a meeting, if one of the agents is a leader, then a flag is raised in both interacting agents. Thus, if a leader "jumps" on a distant agent of the tree, two flags are raised. This ensures that a leader is detected even if it has "jumped" to the tree branch where the last token wave has already passed.

Again, like in *RingDetector*, tokens start white, and when meeting a raised flag, become black and clear the flag. If the master receives a black token from one of its neighbours in the tree, it outputs 1 and 0 otherwise. The technical part of the proof is to show that successive "jumps" of a leader through the graph cannot cause an oscillation of the output of the master.

References

1. D. Angluin, J. Aspnes, Z. Diamadi, M. J. Fischer, and R. Peralta. Computation in networks of passively mobile finite-state sensors. *Distributed Computing*, 18(4):235–253, 2006.
2. D. Angluin, J. Aspnes, and D. Eisenstat. Fast computation by population protocols with a leader. *Distributed Computing*, 21(3):183–199, 2008.
3. D. Angluin, J. Aspnes, D. Eisenstat, and E. Ruppert. The computational power of population protocols. *Distributed Computing*, 20(4):279–304, 2007.
4. D. Angluin, J. Aspnes, M. J. Fischer, and H. Jiang. Self-stabilizing population protocols. *ACM Trans. Auton. Adapt. Syst.*, 3(4), 2008.
5. J. Beauquier, P. Blanchard, and J. Burman. Self-stabilizing leader election in population protocols over arbitrary communication graphs. In *OPODIS*, pages 38–52, 2013.
6. J. Beauquier, P. Blanchard, and J. Burman. A weakest oracle for symmetric consensus in population protocols. Technical report, INRIA, May 2014. http://hal.archives-ouvertes.fr/hal-00992524.
7. J. Beauquier and J. Burman. Self-stabilizing synchronization in mobile sensor networks with covering. In *DCOSS*, volume 6131 of *Lecture Notes in Computer Science*, pages 362–378. Springer, 2010.
8. J. Beauquier, J. Burman, J. Clement, and S. Kutten. On utilizing speed in networks of mobile agents. In *PODC*, pages 305–314. ACM, 2010.
9. F. Bonnet and M. Raynal. Anonymous asynchronous systems: The case of failure detectors. In *DISC*, pages 206–220, 2010.
10. S. Cai, T. Izumi, and K. Wada. How to prove impossibility under global fairness: On space complexity of self-stabilizing leader election on a population protocol model. *Theory Comput. Syst.*, 50(3):433–445, 2012.
11. D. Canepa and M. G. Potop-Butucaru. Self-stabilizing tiny interaction protocols. In *WRAS*, pages 10:1–10:6, 2010.
12. T. D. Chandra, V. Hadzilacos, and S. Toueg. The weakest failure detector for solving consensus. *J. ACM*, 43(4):685–722, 1996.
13. T. D. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems. *J. ACM*, 43(2):225–267, 1996.
14. B. Charron-Bost, M. Hutle, and J. Widder. In search of lost time. *Inf. Process. Lett.*, 110(21):928–933, 2010.
15. A. Cornejo, N. A. Lynch, and S. Sastry. Asynchronous failure detectors. In *PODC*, pages 243–252, 2012.
16. E. W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Commun. of the ACM*, 17(11):643–644, Nov. 1974.
17. M. Fischer and H. Jiang. Self-stabilizing leader election in networks of finite-state anonymous agents. In *OPODIS*, pages 395–409, 2006.
18. 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, Apr. 1985.
19. O. Michail, I. Chatzigiannakis, and P. G. Spirakis. Mediated population protocols. *Theor. Comput. Sci.*, 412(22):2434–2450, 2011.
20. R. Mizoguchi, H. Ono, S. Kijima, and M. Yamashita. On space complexity of self-stabilizing leader election in mediated population protocol. *Distributed Computing*, 25(6):451–460, 2012.

Appendix

Theorem 2. *For any non-simple family of graphs $\mathcal{F}$, there is no self-stabilizing population protocol A implementing $\Omega?$ over $\mathcal{F}$ using the behaviour $\mathcal{ELE}$ (i.e., there is no composition $B = \mathcal{ELE} \circ Beh(A) \subseteq \Omega?$).*

Proof. We prove the result by contradiction using a classical partitioning argument. Assume such a protocol A and consider a graph G from a given non-simple family $\mathcal{F}$, such that there are two disjoint subgraphs of G, G^1 and G^2 that are also in $\mathcal{F}$. Every execution E of A has an input trace (T, T_{in}), where T is an output trace of $\mathcal{ELE}$ and T_{in} represents the input trace of $\Omega?$. Given a schedule S on G, $(S, T_{out}) \in B(S, T, T_{in})$ such that T_{out} is an output trace of A (corresponding to the one of $\Omega?$) induced by S.

By the definition of $\mathcal{ELE}$, each T eventually permanently assigns 1 to a unique agent λ and 0 to every other; we denote by β this assignment. W.l.o.g., assume that $\lambda \in G^1$. We choose the trace T_{in} to be the constant trace $\alpha\alpha\ldots$ where α assigns 1 to some agent $\mu \in G^2$, and 0 to every other.

By the assumption on A, the output trace T_{out} has a suffix equal to the constant trace assigning 1 to every agent. Thus, for every couple (C, γ) in S_E, $\gamma = (\beta, \alpha)$ and the output associated to C assigns 1 to every agent. If we restrict (C, γ) to the graph G^1, we obtain a configuration and input assignment (C^1, γ^1). The agent λ is still the unique agent to be assigned 1 by β^1, and α^1 assigns 0 to every agents in G^1. Since the protocol must be self-stabilizing, and since $G^1 \in \mathcal{F}$, there is a sequence of actions, involving all the agents of G^1 and having the

input assignment γ^1 during the sequence. This leads to a configuration C'^1 that outputs 0 at at least one agent in G^1. This involves that there is a sequence of execution $(C,\gamma)(C_1,\gamma)(C_2,\gamma)\dots(C',\gamma)$ such that C' outputs 1 at the agents of G^2 and 0 at some agent in G^1. The global fairness ensures that C' occurs in S_E; hence a contradiction. □

Theorem 4. *Given any population protocol $\mathcal{A}$ implementing a behaviour B over a family of* strongly *connected graphs, there is a population protocol $\mathcal{A}'$ (given in the proof) implementing B over a family of* weakly *connected graphs.*

Proof. We give a constructive proof. We show how to transform $\mathcal{A}$ into a population protocol $\mathcal{A}'$. Given $\mathcal{A}$, we define below a (possibly) non-deterministic protocol $\mathcal{A}^{ND}$. It can be transformed into a deterministic one by the transformer proposed in [4], if $\mathcal{A}^{ND}$ implements an *elastic behaviour* (as $\mathcal{ELE}$ in our case).

$\mathcal{A}^{ND}$ has the same state space, inputs and outputs as $\mathcal{A}$, and the following transition rules. The rule $(p,x)(q,y) \to (p',q')$ is a rule of $\mathcal{A}^{ND}$ if and only if $(p,x)(q,y) \to (p',q')$ is a rule of $\mathcal{A}$ or $(q,y)(p,x) \to (q',p')$ is a rule of $\mathcal{A}$. For instance, if $\mathcal{A}$ has a unique rule $(p,x)(q,y) \to (p',q')$, then $\mathcal{A}^{ND}$ has two rules, $(p,x)(q,y) \to (p',q')$ and $(q,y)(p,x) \to (q',p')$. In this example $\mathcal{A}^{ND}$ is deterministic but it would not be the case if $\mathcal{A}$ had also a rule $(q,y)(p,x) \to (q'',p'')$. Intuitively, $\mathcal{A}^{ND}$, executing over a weakly connected graph G, simulates $\mathcal{A}$ over a strongly connected graph which is the symmetric closure G_{sym} of G. Alternatively, it is as if $\mathcal{A}^{ND}$ simulated a scheduler, over a non directed graph induced by G, which could choose at every interaction which agent is the initiator, and which is the responder. We now show that $\mathcal{A}^{ND}$ implements the same behaviour B as $\mathcal{A}$ over a family of weakly connected graphs.

Given a weakly connected graph G, the symmetric closure G_{sym} of G is a strongly connected graph. We show that, if $E = (C_0,\alpha_0,\sigma_0)(C_1,\alpha_1,\sigma_1)\dots$ is a globally fair execution of $\mathcal{A}^{ND}$ on G, then there is a sequence of actions σ'_i, $i \in \mathbb{N}$, such that the sequence $E' = (C_0,\alpha_0,\sigma'_0)(C_1,\alpha_1,\sigma'_1)\dots$ is a globally fair execution of $\mathcal{A}$ on G_{sym}. Hence, if $\mathcal{A}$ solves B over G_{sym} then $\mathcal{A}^{ND}$ solves B over G. The corresponding sequence of actions σ'_i is defined as follows. If $\sigma_i = (u_i, v_i, (q,y)(p,x) \to (q',p'))$ but $(q,y)(p,x) \to (p',q')$ is not a rule of $\mathcal{A}$, then define $\sigma'_i = (v_i, u_i, (p,x)(q,y) \to (p',q'))$. If $(q,y)(p,x) \to (q',p')$ is a rule of $\mathcal{A}$, then define $\sigma'_i = \sigma_i$. □

Lemma 7, cases (a) and (b). *Let R be a round in S_E. We denote by $(C_0,\alpha_0)\dots(C_r,\alpha_r)$ the sequence of configurations and input assignments corresponding to R. Case (a) If there are no leaders in R (i.e., for every $0 \le i \le r$, and every agent x, we have $\alpha_i(x).leader = 0$), then after the last action in R, all the agents have their flags cleared (set to 0). Case (b) If there are no leaders in R, and if all agents have their flags cleared at the beginning of the round, then at the end of the round, the master outputs 0 and all agents have their flags cleared.*

Proof. *Case (a).* Assume there are no leaders during the round R. Since the token moves clockwise from the master to the master, and since a token clears any flag it encounters, at the end of the round, the token has cleared all the possible raised flags in the ring.

Case (b). Assume that there are no leaders during R, and that all the flags are cleared at the beginning. During the first action in R, the master holds the token and colors it in white. Since there are no leaders in R, in every configuration within the round, all the flags are cleared. Hence, when moving clockwise from the master to the master, the token meets no raised flags and stays white. At the end of the round, the master receives a white token and outputs 0. □