HAL
open science

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. 18th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS 2016), Nov 2016, Lyon, France. hal-00839759v3

HAL Id: hal-00839759

https://hal.science/hal-00839759v3

Submitted on 25 Sep 2016

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 Ω? for Self-Stabilizing Leader Election in Population Protocols

Joffroy Beauquier[1], Peva Blanchard[2], Janna Burman[1], and Oksana Denysyuk[3]

[1] LRI, Université Paris-Sud, Orsay, France, {beauquier, burman}@lri.fr
[2] LPD, EPFL, Lausanne, Switzerland, peva.blanchard@epfl.ch
[3] University of Calgary, Canada

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. $SSLE$ has been 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). Fischer and Jiang have proposed solutions to $SSLE$, for complete communication graphs and rings, using the oracle Ω?, called the *eventual leader detector*. In this paper, we investigate the power of Ω? on larger families of graphs. We present two important results.
Our first result states that Ω? is powerful enough to allow solving $SSLE$ over arbitrary communication graphs of bounded degree. Our second result states that, Ω? is the weakest (in the sense of Chandra, Hadzilacos and Toueg) for solving $SSLE$ over rings. We also prove that this result does not extend to all graphs; in particular not to the family of arbitrary graphs of bounded degree.

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

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 even 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. A fundamental issue is to determine the oracle providing the minimum amount of information for solving 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 called *population protocols*. It has been introduced as a model 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* ($SSLE$), which consists in electing a leader (a distinguishable agent) in a self-stabilizing way. *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 this work, we focus on this fundamental problem $SSLE$ that is shown to be impossible in many different cases [4,17,5].

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 bounded state agents in population protocols. Many other previous oracles, like those proposed for anonymous models (e.g., [9]), cannot be used in population protocols either, e.g., because they assume finite, but unbounded memory depending on the size of the network (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 and 3.2 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 actual communication graph; the agents only know the graph family to which the graph belongs. Our focus here is on uniform solutions too.[1]

Contribution. In this work, we investigate the power of Ω?. In particular, in Sec. 4, we show that its power exceeds considerably the case of rings and complete graphs (concerned in [17]). In fact, Ω? is sufficient for solving $SSLE$ on almost all graphs, the only restriction being that the graph must be connected (obvious) and of bounded degree (related to the model requirement of bounded agent states).

In Sec. 5, we show that $SSLE$ allows to implement Ω? on rings. Coupled with the fact that Ω? is sufficient for solving $SSLE$ on rings [17], this implies that any oracle strong enough for solving $SSLE$ on rings can be used to implement Ω? (on rings); i.e. Ω? is the weakest oracle for solving $SSLE$ on rings.

In contrast with the previous case, we also show that over arbitrary communication graphs of bounded degree (and more generally, over *non-simple* graph families), $SSLE$ is not equivalent to Ω? (Th. 2). Intuitively, our results mean that, whereas $SSLE$ and Ω? are not equivalent over certain families of graphs, this difference disappears on rings due to the strong communication constraints imposed by this topology.

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). In this framework, there is no difference between an oracle and a problem, so the relations that 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 Ω?. 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

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

$SSLE$ 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 the current paper, we consider only *global fairness* (classical for population protocols).

In [5], it is shown that the difficulty in solving $SSLE$ in population protocols comes from the requirement of self-stabilization. Indeed, [5] presents a solution for arbitrary graphs *with a uniform initialization without any oracle*. Then, [5] proposes also a solution for $SSLE$ over arbitrary graphs, but the protocol uses a *much stronger oracle*. This oracle can be viewed as a composition of two copies of $\Omega?$, where one copy is used to control the number of (stationary) leaders and another one to control the number of moving tokens. There, tokens are used for eliminating supplementary leaders. In this paper, we prove that, surprisingly enough, there is no need to control the number of tokens and that a single instance of $\Omega?$ is enough (at least, in the case of bounded degree graphs). Finally, [5] shows that $SSLE$ and $\Omega?$ are not equivalent over *complete* communication graphs. Here, we extend this result to so called *non-simple families* of graphs (Th. 2).

2 Model and Definitions

2.1 Population Protocol

We use here the definitions of [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 $\mathcal{Q}^2$.[1] 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 $(q_1, x_1)(q_2, x_2) \to (q_1', q_2')$

[1] 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.

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 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 $0 \leq i < k$, and we say that C' is *reachable* from C given the input trace T_{in}.

An *execution* is a sequence of configurations, input assignments and actions $(C_0, \alpha_0, \sigma_0)\ (C_1, \alpha_1, \sigma_1) \ldots$ such that for each i, $(C_i, \alpha_i) \xrightarrow{\sigma_i} C_{i+1}$. In addition, the sequence satisfies *global fairness* 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 produces 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.[1] 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 is sometimes implicitly referenced.

A *schedule* is a sequence of edges (representing 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. This definition is natural since an agent can only be activated during a meeting, and it makes no sense to allow a

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

change in inputs which cannot be detected by the agents. Note also that the output trace (associated with an execution with a schedule S) is necessarily compatible with S by definition.

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 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 $B(G, H_{in})$, or simply $B(H_{in})$, of output histories with the same schedule S. The output histories of $B(H_{in})$ are the *legal* output histories of B given H_{in}.

In a natural way, behaviours can be composed in series, parallel, or by self-loop. For instance, in the serial composition, an output trace of a behaviour is the input trace of another one. Formally, 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, T_Z denotes a trace with values in Z.

Let S be a schedule on $G \in \mathcal{F}$. 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-loop 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. $(S, T_U, T_W) \in B_1(S, T_U, T_V)$. As already mentioned, the self-loop composition is necessary to describe the interactions between a protocol and an oracle.

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-loop 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, the *behaviour* $Beh(\mathcal{A})$ *associated to the protocol* $\mathcal{A}$ is 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 simply defined as behaviours. Now, we are ready to define what it means for a protocol $\mathcal{A}$ to implement a behaviour (or solve the problem) B using an oracle O. The population protocol $\mathcal{A}$ *implements the behaviour* B (or *solves the problem* B) using the behaviour O if there exists a composition B^* involving the behaviours O 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 self-stabilizing[1] 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 *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.

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 problem of Self-Stabilizing Leader Election ($SSLE$) consists in defining a population protocol that solves $\mathcal{ELE}$ using another behaviour (if necessary) and starting from arbitrary initial configurations.

3.2 Oracle Ω?

Informally, Ω? (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 Ω?).

Formally, Ω? is simply a relation between input and output traces with binary values. 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 $\Omega?(S, T_{in})$.

Ω? is easy to implement in practice, provided that timeouts are available. Each leader periodically broadcasts a "leader signal". Each agent resets the timer when it receives the signal. If the timeout expires, the agent sets a flag to false, signaling the absence of leader. The flag is reset to true when a "leader signal" is received. In a chaotic environment in which communications are bad or nodes are malfunctioning, the implemented oracle can give incorrect answers, making the system unstable. But, eventually, after the environment has regain its consistency, Ω? will give a correct information and the system will stabilize.

[1] In this paper, we are only interested in comparing oracles as far as self-stabilization is concerned.

4 *SSLE* using Ω? 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 Ω? 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 $Self(Beh(\mathcal{A}_d) \circ \Omega?)$ is a sub-behaviour of $\mathcal{ELE}$. We first give a solution over the family of *strongly connected* graphs with bounded degree. The transformation of this solution into one over *weakly connected* graphs with bounded degree is formally presented in the appendix (Sec. A.2, Th. B).

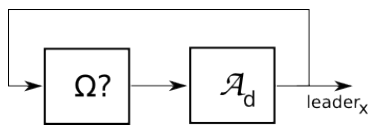

Fig. 1. Serial composition $Beh(\mathcal{A}_d) \circ \Omega?$ followed by a self-loop composition.

We first briefly recall how the Fischer and Jiang's protocol for rings [17] works. As said before, the information given by Ω? does not allow to distinguish between the presence of a single or more leaders. Thus, a leader should try to eliminate other leaders, while avoiding a scenario where all leaders are eliminated infinitely often (without any help from the oracle). On a ring, a strategy performing this goal is relatively simple to install. Leaders send tokens, circulating on the ring in one direction and send also shields, circulating in the opposite direction. Shields absorb tokens when they meet, but a leader that receives a token is eliminated. When there remains a single leader, it sends a token and a shield (in opposite directions) and the ring structure ensures that the token cannot avoid the shield, so that a unique leader cannot eliminate itself.

The situation is completely different on arbitrary graphs, since tokens and shields can take different routes. This requires a completely different management for a single leader not eliminating itself. As the agents are finite-state, a bounded degree is needed for implementing such a management.

For distinguishing between the different possible routes, each agent has to give different (local) names to its neighbors. For that, 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 $O(d^2)$) of possible colors.

The input variables (read-only) of our protocol $\mathcal{A}_d$ 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 *token vector* $token_x$ (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. tokens), 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 token vectors. The purpose of the tokens is to eliminate leaders (line 10), whereas the purpose of the shields is to protect them by absorbing tokens (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 token 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 token, it reloads tokens for every color (line 11). In addition, whenever a leader interacts as an initiator, it loads tokens 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 Ω?_x = 0 then
3      leader_x ← 1
4      ∀c ∈ Colors, token_x[c] ← 1
5      ∀c ∈ Colors, shield_x[c] ← 1
6  end
7  (Move token from y to x, if any)
8  if token_y[c_x] = 1 then
9      if shield_x[c_y] = 0 then
10         leader_x ← 0
11         ∀c ∈ Colors, token_x[c] ← 1
12     token_y[c_x] ← 0
13 end
14 (Move shield from x to y, if any)
15 if shield_x[c_y] = 1 then
16     ∀c ∈ Colors, shield_y[c] ← 1
17     token_y[c_x] ← 0
18     shield_x[c_y] ← 0
19 end
20 (Load tokens if x is a leader)
21 if leader_x = 1 then
22     ∀c ∈ Colors, token_x[c] ← 1
```

Before proving the correctness of the algorithm, we introduce some definitions. 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 token against* x if $token_y[c_x] = 1$.

The crucial idea of the proof relies on the notion of *protected leader*. Intuitively, a leader λ is protected if, in any loop at λ, some agent (the protector) protects λ thanks to a shield against its successor, and no agent between λ and the protector has a token against its predecessor.

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 token 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 token 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.

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. The following lemma constitutes the core of our argument. We give a detailed proof.

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. The assumption on the protocol $2HC$ states that α yields a correct 2-hop coloring. Consider a configuration C' that follows the occurrence of (C, α) in S_E. In particular, $(C, \alpha) \to C'$. We note (x, y) be the pair of edges involved (initiator x, responder y).

When a leader is created, it is already protected by itself since it has a shield against every of its out-neighbors. We thus focus on transition rules that do not involve the creation of a leader. Hence, such a transition may eliminate a leader, or move or create shields and tokens.

Let λ be a protected leader in γ 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. Let $z \in \{x, y\}$ be the agent with the lowest index $ind_\pi(z)$. The previous assumption implies $ind_\pi(z) \leq ind_\pi(\mu)$.

Consider first the case $ind_\pi(z) < ind_\pi(\mu)$. If $z = x$, then z cannot receive a token (from y), i.e., either x has a shield against y or y has no token 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 in 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 shield); hence, λ is still protected by μ in π after the transition. We now focus on the case $\mu = x$. First consider the subcase where y is not the agent that follows the first occurrence of μ in π. Then μ cannot receive a token 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 still has 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 every case, 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 previous argument shows that every configuration in this sequence has a protected leader, in particular C''. Therefore, any configuration in IRC_E has a protected leader. □

Lemma 2. *All configurations in IRC_E have the same number $l \geq 1$ of leaders. In addition, no configuration in IRC_E contains an unprotected leader.*

Proof (Sketch). Full details are presented in the appendix, Sec. A. If there is either no leader, then at some point, Ω? will force the creation of a (protected) leader. If there is always at least one leader, but they are all unprotected, then it means that infinitely often there is a possibility to kill a leader. Global fairness ensures that all the unprotected leaders will eventually be eliminated, which is a contradiction. In all cases, it means that every configuration in IRC_E contains at least one protected leader. In particular, Ω? will not create new leaders. This implies that, once all unprotected leaders have been killed, there is a constant number of protected leaders. □

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

Proof (Sketch). See the appendix, Sec. A for full details. Any configuration in IRC_E has the same number $l \geq 1$ of (protected) leaders. Assume that $l \geq 2$, consider two protected leaders λ_1, λ_2 and the loop π built from the shortest path from λ_1 to λ_2 followed by the one from λ_2 to λ_1. By moving the protector of λ_1 behind λ_2, and making λ_2 fires a token, it is possible to eliminate λ_1. The global fairness ensures that this eventually happens, which reduces the number l; hence a contradiction. Thus, there is eventually a unique leader. □

5 Is Ω? the Weakest Oracle for Solving $SSLE$?

Now, we come to the second important result of this paper. The search for weakest oracles, since the weakest failure detector of Chandra and Toueg, has been a constant quest in distributed computing. The weakest oracle, for solving a problem elsewhere impossible, represents the minimum supplementary information needed. As this supplementary information can be provided naturally by the environment, its determination is of great interest for implementing a solution. Our approach here is different from the approach of failure detectors, in the sense that the oracles we consider are in a larger class. Indeed, failure detectors only observe a pattern of failures, whereas oracles like Ω? are able to react to the output of the protocol. We

would like to emphasize the fact that, for dealing with transient failures (state corruptions) an oracle must have access to the states of the agents. It is not a choice that we make, but a necessity. In such a general setting, the issue of the weakest oracle is reduced to the issue of the equivalence of such an oracle with the problem itself. In consequence, we have to prove that $SSLE$ allows to implement this oracle. We answer this issue relatively to different communication topologies.

5.1 An Impossibility Result for Non-Simple Families of Graphs

It turns out that for some graph families, a negative answer (Th. 2) holds. A somewhat similar result, 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 exists a graph $G \in \mathcal{F}$, and two disjoint subgraphs G_1, G_2 of G such that $G_1, G_2 \in \mathcal{F}$. 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 d-regular graphs.

The following theorem states the impossibility of a self-stabilizing implementation of $\Omega?$ using $\mathcal{ELE}$ over any non-simple family of graphs. Coupled with the result of Sec. 4, i.e. $\Omega? \succcurlyeq \mathcal{ELE}$ over connected arbitrary graphs of bounded degree, we have $\Omega? \succ \mathcal{ELE}$ over the same graph family. Similarly, $\Omega?$ is not the weakest oracle for $SSLE$ over complete graphs, and, by the $SSLE$ protocol of [17], we have $\Omega? \succ \mathcal{ELE}$ over complete graphs. The proof of Theorem 2 uses a classical *partitioning argument* and appears in the appendix (Sec. B).

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 $B = Beh(A) \circ \mathcal{ELE} \subseteq \Omega?$). In particular, $\Omega? \succ \mathcal{ELE}$ over complete graphs and over arbitrary connected graphs of bounded degree.*

5.2 $\Omega?$ is the weakest oracle for $SSLE$ over rings

Now, we show that $\Omega?$ can be implemented in a self-stabilizing way given the behaviour $\mathcal{ELE}$ over *oriented* rings. Note that this is not about detecting the agent selected by $\mathcal{ELE}$ (which would be trivial). Instead, we define a protocol which uses the eventual presence of a distinguishable agent (guaranteed by $\mathcal{ELE}$), hereafter called the *master*, to detect the presence or absence of leaders in the input trace. This implementation is given by the *RingDetector* protocol presented below (see Algorithm 2). This result is 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; i.e., $\Omega?$ is the weakest oracle for solving $SSLE$ over rings.

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

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 Ω?. The working variables are: the *probe field* $probe_x$ (with values: $\perp$ - no probe, or 0 - white probe, or 1 - black probe); the *token field* (with values: $\perp$ - 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 Ω?.

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).

Correctness

We use the same notations S_E and IRC_E as in the previous section. By the definition of $\mathcal{ELE}$, a unique agent eventually becomes the master permanently. We focus on the corresponding suffix of the execution (S_E is included in this suffix). Furthermore, we denote by $C(x).token$ (resp. $C(x).probe$, etc.) the value of the variable *token* (*probe*, etc.) in the configuration C at agent x. Similarly, we denote by $\alpha(x).leader$ (resp. $\alpha(x).master$) the value of the variable *leader* (resp. *master*) in the input assignment α at x. The following lemma states that, eventually, a unique token circulates in the ring.

Lemma 3. *In any configuration $C \in IRC_E$, there is exactly one token (white or black) in C, i.e., there exists a unique agent x such that $C(x).token \neq \perp$.*

Proof (Sketch). If there are no tokens, some probe sent by the master will return to the master with the color white (recall that the probes and tokens move in opposite directions). This causes the master to fire a token. Two colliding tokens merge into one. This implies that there will always be at least one token. In particular, all the probes sent by the master will return to the master with the color black; thus no more tokens are created. Moreover, thanks to the global fairness, if there are several tokens, they eventually all merge into a unique token. □

This unique circulating token (from the lemma above) allows to divide the execution into *rounds*. We define a *round* to be a segment of S_E that begins with the token loaded at the master, and ends up right before the token returns to the master. The following lemma describes the output of the master at the end of each round.

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

```
(if the master is the responder, it creates a white probe)
if master_y = 1 then probe_y ← 0

(raise flags if needed)
if leader_x ∨ leader_y then flag_x ← flag_y ← 1

(move probe from y to x)
if probe_y ≠ ⊥ then
    (the probe becomes black when meeting a token)
    if token_x ≠ ⊥ then probe_x ← 1
    otherwise, keeps the same color or merges)
    else if probe_x ∈ {⊥, 0} then probe_x ← probe_y
    probe_y ← ⊥
end

(if the master receives a white probe, it loads a token)
if master_x = 1 and probe_x = 0 then token_x ← flag_x
(move token from x to y)
if token_x ≠ ⊥ then
    (the token becomes black when meeting a flag)
    if flag_y = 1 then token_y ← 1
    (otherwise, keeps the same color or merges)
    else if token_y ∈ {⊥, 0} then token_y ← token_x
    (the flag is cleared)
    flag_y ← 0
    token_x ← ⊥
end
(if the master receives a token, it changes its output and whitens the token)
if master_y = 1 and token_y ≠ ⊥ then
    out_y ← token_y
    token_y ← 0
(a non-master responder copies the output of the initiator)
if master_y = 0 then out_y ← out_x
```

Lemma 4. *Consider a round R in S_E. We denote by $(C_0, \alpha_0) \dots (C_r, \alpha_r)$ the corresponding sequence of configurations and input assignments. Case (a) If there are no leaders during 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 of the round, all the agents have their flags cleared (set to zero). Case (b) If there are no leaders during R, and if all the agents have their flags cleared at the beginning of the round, then after the last action of the round, the master outputs 0 and all the agents have their flags cleared. Case (c) If there is at least one leader in each assignment during R, i.e., for every $0 \leq i \leq r$, there is some agent x_i such that $\alpha_i(x_i).leader = 1$, then after the last action of the round, the master outputs 1.*

Proof (Sketch). We only prove here the case (c). Full proof details are presented in the appendix, Sec. B. Assume that there is a leader in each input assignment. Let μ be an agent that holds a leader in assignment α_0, i.e., $\alpha_0(\mu).leader = 1$. During the round, there must be some i, such that $\mu = v_i$ is the responder and the initiator u_i holds the token. If μ holds a leader in assignment α_i, then after the transition, the token must have turned black. If μ does not hold a leader in assignment α_i, since μ did hold a leader in assignment α_0, there must be some $j < i$ 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 its flag, 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 i, the token turns black 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. Moreover, $\Omega? \simeq_{rings} \mathcal{ELE}$ (by [17]), and thus $\Omega?$ is the weakest oracle for solving $\mathcal{ELE}$ over rings.*

Proof (Sketch). See the appendix, Sec. B, Th. D, for full details.We divide the execution in rounds as defined above. If there are no leader forever, then Lemma 4 ensures that after a finite number of rounds, the master permanently outputs 0. If there is a leader in each input assignment, then Lemma 4 ensures that after a finite number of rounds, the master permanently outputs 1. In both cases, the propagation of the master's output ensures that the output trace of the protocol satisfies the oracle $\Omega?$ conditions (see Sec. 3.2 for its definition). □

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, J. Burman, and S. Kutten. The weakest oracle for symmetric consensus in population protocols. In *ALGOSENSORS*, pages 41–56, 2015.
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

A $\mathcal{ELE}$ using Ω? over Weakly Connected Graphs with Bounded Degree

A.1 Solution over strongly connected graphs

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).

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 A *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. The assumption on the protocol $2HC$ states that α yields a correct 2-hop coloring. Consider a configuration C' that follows the occurrence of (C, α) in S_E. In particular, $(C, \alpha) \to C'$. We note (x, y) be the pair of edges involved (initiator x, responder y).

When a leader is created, it is already protected by itself since it has a shield against every of its out-neighbors. We thus focus on transition rule that do not involve the creation of a leader. Hence, such a transition may kill a leader, or move or create shields and tokens.

Let λ be a protected leader in γ 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. Let $z \in \{x, y\}$ be the agent with lowest index $ind_\pi(z)$. The previous assumption implies $ind_\pi(z) \leq ind_\pi(\mu)$.

Consider first the case $ind_\pi(z) < ind_\pi(\mu)$. If $z = x$, then z cannot receive a token (from y), i.e., either x has a shield against y or y has no tokens 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 in 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 shield); hence, λ is still protected by μ in π after the transition. We now focus on the case $\mu = x$. First consider the subcase where y is not the agent that follows the first occurrence of μ in π. Then μ cannot receive a token 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 still has 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 every 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 previous argument shows that every configuration in this sequence has a protected leader, in particular C''. Therefore, any configuration in IRC_E has a protected leader. □

Lemma B *If no configuration in IRC_E has a leader, then all input assignments in S_E equal an input assignment that assigns 0 to every variable $\Omega?_x$ and yields a 2-hop coloring. If every configuration in IRC_E has a leader, then all input assignments in S_E equal an input assignment that assigns 1 to every variable $\Omega?_x$ and yields a 2-hop coloring.*

Proof. This stems from the definition of Ω? and the assumption on $2HC$. □

Lemma C *Every configuration in IRC_E has at least one leader, and every input assignment in S_E is equal to an input assignment α^E that assigns 1 to every variable $\Omega?_x$ and yields a 2-hop coloring.*

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 B, every input assignment

in S_E assigns 0 to every $\Omega?_x$. Hence, in S_E, during every transition, a *protected* leader is created. 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 one of the actions, a *protected* leader must be created. In both cases, we have a configuration $C'' \in IRC_E$ with a protected leader. By Lemma A, this implies that all configurations in IRC_E has a protected leader, in particular C; hence a contradiction. Thus any configuration in IRC_E has a leader. The assumption on the protocol $2HC$ and Lemma B yield the last claim. □

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

Proof. By Lemma C, every input assignment in S_E assigns 1 to every variable $\Omega?_x$. Thus no leader is created during S_E. Assume 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, there must be a sequence of actions in S_E 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 E *No configuration in IRC_E contains an unprotected leader.*

Proof. Suppose that $C \in IRC_E$ contains an unprotected leader λ. By Lemma C, there is an input assignment α^E such that (C, α^E) occurs in S_E and α^E assigns 1 to every variable $\Omega?_x$. We describe a sequence of actions with the input assignment α^E at each step. Since λ is not protected in C, there exists a path $\pi = x_0 \dots x_r$ from agent $x_0 = \lambda$ to some agent x_r such that for every $0 \leq i < r$, agent x_i has no shield against x_{i+1} and x_r either is a leader or has a token against x_{r-1}. If x_r is a leader, any transition where x_r is a initiator makes x_r creating a token against x_{r-1}. Then by moving (backward) the token along this path, it is possible to kill the non-protected leader λ. We reach a configuration C' within which λ is not a leader. Since no leaders have been created during the sequence, C' has fewer leaders than C. The global fairness ensures that $C' \in IRC_E$; this contradicts Lemma D. □

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

Proof. By Lemma D and E, we know that any configuration in IRC_E has the same number l of protected leaders and no unprotected leaders; and also that all input assignments are equal to some α^E that gives a 2-hop coloring and assigns the value 1 to every variable $\Omega?_x$. Lemma C ensures that $l \geq 1$. Assume, by contradiction, that $l \geq 2$. Let $C \in IRC_E$. Let λ_1, λ_2 be two protected leaders in C. Consider p_1 (resp. p_2) the shortest path from λ_1 to λ_2 (resp. from λ_2 to λ_1). We define the loop $\pi_1 = p_1 p_2$ at λ_1 and the loop $\pi_2 = p_2 p_1$ at λ_2. We note μ_1 (resp. μ_2) the protector λ_1 (resp. λ_2) in π_1 (resp. π_2). Necessarily, the first occurrence of μ_1 (resp. μ_2) is in p_1 (resp. p_2). We describe a sequence with input assignment α^E at every step. The protocol allows to move the (first occurrence of the) protector μ_1 right before λ_2. Another such action

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 token that kills λ_1. Since, no leader is created during the sequence, we reach a configuration C' with less than l leaders. The global fairness ensures that $C' \in IRC_E$. This contradicts Lemma D. Therefore, all configurations in IRC_E have a unique leader. Since the leaders cannot move, there is a permanent leader. □

A.2 From Strongly to Weakly Connected Graphs

Now we show that the results above can be extended to the more general family of weakly connected graphs with bounded degree. We actually exhibit a slightly more general transformation.

First, we give some specific definitions. Given an edge $e = (x, y)$ of a graph G, we denote by $\bar{e} = (y, x)$ the opposite edge (not necessarily an edge of G). The symmetric closure of a graph G, is the graph G_{sym} with the same set of vertices obtained from G by adding the opposite edges. Consider a family $\mathcal{F}$ of graphs closed by symmetric closure, $G \in \mathcal{F} \Rightarrow G_{sym} \in \mathcal{F}$, and let B be a behaviour over $\mathcal{F}$. We say that B is *undirected* when, for any graph $G \in \mathcal{F}$, for any schedule $S = e_1 e_2 \dots$ on G, $(S, T_{out}) \in B(G, S, T_{in})$ if and only if $(S', T_{out}) \in B(G_{sym}, S', T_{in})$ for any schedule $S' = e'_1 e'_2 \dots$ on G_{sym} such that $e'_i \in \{e_i, \bar{e}_i\}$. Intuitively, a behaviour is undirected if the legal output histories do not depend on the orientation of the edges of G.

Theorem B *Consider a graph family $\mathcal{F}_0$, and let $\mathcal{F}$ be the family of graphs G such that $G_{sym} \in \mathcal{F}_0$. The family $\mathcal{F}$ is closed by symmetric closure. Let B, O be undirected behaviours over $\mathcal{F}$ such that there exists a population protocol $\mathcal{A}$ implementing B using O over the family $\mathcal{F}_0$. Then there exists a population protocol $\mathcal{A}'$ (given in the proof) implementing B using O over the family $\mathcal{F}$.*

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 using the transformer proposed in [4], provided that the behaviour B is an elastic behaviour (see [4] for more details).

$\mathcal{A}^{ND}$ 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 graph $G \in \mathcal{F}$, simulates $\mathcal{A}$ over the symmetric closure $G_{sym} \in \mathcal{F}_0$. 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 claim 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, since $\mathcal{A}$ solves B over G_{sym} using the oracle O, the protocol $\mathcal{A}^{ND}$ solves B over

G using the same oracle O. 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 $\sigma'_i = \sigma_i$.

Note that the fact that B and O are undirected ensures that the input history of $\mathcal{E}'$ on G_{sym} is a legal output history of O on G_{sym}, and the output history associated with $\mathcal{E}'$ is a legal history of B. □

Taking $\mathcal{F}_0$ to be the family of strongly connected graphs (in and out) degree bounded by $2d$, $\mathcal{F}$ becomes the family of weakly connected graphs with (in and out) degree bounded by d. The previous theorem yields the following corollary:

Corollary 1. *The protocol $\mathcal{A}_{2d}$ can be transformed in a protocol $\mathcal{A}'_d$ which solves $\mathcal{ELE}$ using $\Omega?$ over the family of weakly connected graphs with (in and out) degree bounded by d.*

B Is $\Omega?$ the Weakest for Solving $SSLE$?

Theorem C *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 $B = Beh(A) \circ \mathcal{ELE} \subseteq \Omega?$).*

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 $(C, \gamma)(C_1, \gamma)(C_2, \gamma)\ldots(C', \gamma)$ 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. □

Now, we prove that *RingDetector* is a self-stabilizing implementation of Ω? using $\mathcal{ELE}$. We use the same notations S_E and IRC_E as in the previous section. In addition, the input trace $T = \alpha_0\alpha_1\ldots$ of the 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. 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.

Lemma F *In any configuration $C \in IRC_E$, there is exactly one token (white or black) in C, i.e., there exists a unique agent x such that $C(x).token \neq \perp$.*

Proof. Consider a configuration $C \in IRC_E$. We first prove that C contains at least one token. On the contrary, assume that, for every agent x, $C(x).token = \perp$. The following scenario will produce a token. First, let the master λ interacting as a responder to produce a white probe at λ. Then, move (counter-clockwise) all the other probes, if any, to the master. Then move the white probe at λ so as to visit all the agents and return to λ again. Since there are no tokens in the graph, the white probe will not turn black. Then, the white probe arriving at λ will make λ produce a token. This scenario does not depend on the possibly present leaders. Hence, we have shown that there exists a configuration C' with at least one token such that $C \xrightarrow{*} C'$, whatever the input trace is during this sequence. By global fairness, we know that C' belongs to IRC_E. But, the rules of the protocol are such that, once there is at least one token in the graph, there is always at least one token in the graph in any subsequent configuration. Thus C cannot occur infinitely often; hence a contradiction. Hence C has at least one token.

Assume now that C has at least two tokens. Since two meeting tokens merge into one token, there is a configuration C' with exactly one token such that $C \xrightarrow{*} C'$, whatever the input trace is. By global fairness, C' belongs to IRC_E. Since C also occurs infinitely often in the execution, and since the only way to create a token is by having the master receive a white probe, this means that the master receives infinitely many white probes during S_E. But once there is a token in the graph, 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. □

Thanks to the previous lemma, we know that during S_E a unique token circulates clockwise. We define a *round* to be a segment of S_E that begins with the token loaded at the master, and ends up right before the token returns to the master.

Lemma G *Consider a round R in S_E. We denote by $(C_0, \alpha_0)\ldots(C_r, \alpha_r)$ the corresponding sequence of configurations and input assignments. Case (a) If there are no leaders during 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 of the round, all the agents have their flags cleared (set to zero). Case (b) If there are no leaders during R, and if all the agents have their flags cleared at the beginning of the round, then after the last action of the round, the master outputs 0 and all the agents have their flags cleared. Case (c) If there is*

at least one leader in each assignment during R, i.e., for every $0 \leq i \leq r$, there is some agent x_i such that $\alpha_i(x_i).leader = 1$, then after the last action of the round, the master outputs 1.

Proof. *Case (a).* Assume there are no leaders in the round. Since the token moves clockwise from the master to the master, and since a token clears any flag it encounters, after the last action in the round, the token must have cleared all the possible raised flags in the ring.

Case (b). Assume that there are no leaders in the round, and that all the flags are clear at the beginning. During the first action, the master holds the token and colors it in white (the master holds no leader). Since there are no leaders in the round, 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.

Case (c). Assume that there is a leader in each input assignment. Let μ be an agent that holds a leader in assignment α_0, i.e., $\alpha_0(\mu).leader = 1$. During the round, there must be some i, such that $\mu = v_i$ is the responder and the initiator u_i holds the token. If μ holds a leader in assignment α_i, then after the transition, the token must have turned black. If μ does not hold a leader in assignment α_i, since μ did hold a leader in assignment α_0, there must be some $j < i$ 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 $(C_j, \alpha_j) \rightarrow C_{j+1}$. Hence, μ must raise its flag, 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 i, the token turns black 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 D *The protocol RingDetector is a self-stabilizing implementation of Ω? using $\mathcal{ELE}$ over oriented rings.*

Proof. Consider a globally fair execution E and focus on the suffix S_E. For the sake of simplicity, we assume that there is a unique master from the beginning. By Lemma F, we know that in S_E there is a unique token moving clockwise. Without loss of generality, we assume that S_E begins with the token being hold by the master. We then write $S_E = R_1 R_2 \ldots$ 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 leaders everywhere, i.e., for all i, for every agent x, $\alpha_i(x).leader = 0$. By Lemma G, we know that at the end of R_1, all the flags are cleared. Hence, at the end of R_2, the master outputs 0 and all the 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, we know that in the suffix $R_2 R_3 \ldots$, 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 the agents permanently output 0.

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

Note that, in the remaining cases of input traces in S_E, that is when there are some input assignments with a leader and some other without, nothing has to be proven, because then, the output of Ω? is arbitrary (see definition in Sec. 3.2). □

Remark 1. Note that as leaders can appear and then disappear completely, one require a kind of recurring procedure verifying if leaders have reappeared. The permanent master guaranteed by $\mathcal{ELE}$ plays the role of an arbiter initiating this recurring procedure by sending the exploring tokens. Tokens look for on flags and switch them off, to be prepared for the next exploration tour. One may think that there is a simpler solution using no probes. That is, a solution, where, e.g., master sends the tokens repeatedly after some bounded number of interactions it participates in, otherwise exploring token may never exist or be created. However, such a solution without probes might be incorrect. To see this, consider an input trace where there is one leader in every input assignment, but this leader moves repeatedly clockwise, "jumping" from initiator to responder on the oriented ring. By the definition of Ω?, in this scenario, the master should eventually and permanently output 1. However, it is infinitely often possible that there are two tokens (generated by the master) 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 master's output between 0 and 1.