Oracles for Self-Stabilizing Leader Election in Population Protocols

Joffroy Beauquier, Peva Blanchard, Janna Burman, Oksana Denysyuk

Oracles for Self-Stabilizing Leader Election in Population Protocols

Joffroy Beauquier
LRI, University of Paris-Sud XI, Orsay, France, jb@lri.fr

Peva Blanchard*
LRI, University of Paris-Sud XI, Orsay, France, blanchard@lri.fr

Janna Burman
LRI, University of Paris-Sud XI, Orsay, France, burman@lri.fr

Oksana Denysyuk
NESC-ID, Instituto Superior Técnico, Universidade Técnica 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, like the eventual *leader detector* $\Omega?$ of Fischer and Jiang, who presented a uniform protocol solving $SSLE$ with the help of $\Omega?$ on complete communication graphs and rings. In this paper, we extend their results. Our first contribution is a precise framework for dealing with oracles. This framework is independent of the notion of real time. Such a design choice avoids some known problems of traditional real time based frameworks. We then formally define $\Omega?$ as well as a *stronger* oracle $\Omega\$$ and a *weaker* one $W\Omega?$. The comparison between the oracles is based on the notion of *implementation*. We prove that $SSLE$ can be implemented with $\Omega\$$ over weakly connected communication graphs, with $W\Omega?$ over oriented trees and with $\Omega?$ over weakly connected communication graphs of bounded degree. Finally, we show that $\Omega?$ can be implemented using $SSLE$ over rings, proving their equivalence. All these results allow to establish relations between the different oracles and $SSLE$, which we summarize in a figure.

*Contact author: LRI, Bât. 650, University of Paris-Sud XI, 91405 Orsay Cedex France. tel: 33 (0)1 69 15 64 32

1 Introduction

The literature on distributed computing includes a number of impossibility results. The impossibility can be related to the system asynchrony, the presence of failures, their type, or too general initial conditions. For instance, the consensus problem has been shown to be impossible in asynchronous systems with only one crash fault [14]. An elegant approach for circumventing the impossibility of consensus is the abstraction known as *failure detectors* introduced by Chandra and Toueg [9]. In some sense, a failure detector can be viewed as an oracle, which provides to the system nodes a supplementary information about failures and allows to solve a given problem. Defining such oracles raises the fundamental question of providing the minimum amount of information sufficient to solve the problem. Among the different failure detectors proposed in this approach to solve consensus in the conventional asynchronous communication model, one of them, the *eventual leader elector* Ω, has been proven to be the *weakest*. Informally, that means that it supplies a minimal supplementary information necessary to obtain a solution [8].

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 possessing 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, especially when failures are possible. One of them concerns the problem of self-stabilizing leader election ($SSLE$) [3, 13]. Self-stabilization [12] is a framework for dealing with transient state-corrupting faults, but can be generally viewed as not allowing presupposed initial configurations. In other words, a protocol solves a problem in a self-stabilizing way if every execution starting from an arbitrary initial configuration solves the problem.

The eventual leader elector Ω of Chandra and Toueg cannot be used with population protocols, because Ω assumes that the network nodes have unique identifiers, unavailable to anonymous agents in population protocols. Moreover, an agent cannot even determine whether two messages are from the same sender, or if two successive messages it has sent have the same receiver. These are some reasons why Fischer and Jiang introduced a new type of oracle, called $\Omega?$ [13]. 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 [13]. 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 from the protocol using it, $\Omega?$ interacts with the protocol, providing information related to the reached configurations. With $\Omega?$, there is some sort of a feedback loop, the outputs of the oracle influence the protocol; and conversely, the protocol influences the outputs of the oracle. Together with that, 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, exactly like Ω. To demonstrate the power of $\Omega?$, [13] gives a solution to $SSLE$ using $\Omega?$ in complete communication graphs and rings. Note that in [13] the oracle is defined in a rather informal way.

The first contribution of this paper is a formal framework adapted to the population protocols for modeling oracles and obtaining relations between them (Sec. 2.2). In this framework, oracles are completely independent of the notion of real time, which also allows to establish a precise hierarchy of oracles. In contrast, the notion of real time (meaningless in asynchronous protocols) raises known problems (e.g., non self-implementable oracles) when establishing such hierarchy in the framework of the classical failure detectors (see, e.g., [10, 11]).

In the proposed framework, we give a precise definition of $\Omega?$ and also propose two new unreliable oracles: a stronger oracle $\Omega\$$, and a weaker $W\Omega?$ (Sec. 3.2). Informally, $\Omega?$ reports to each agent a guess about whether or not one or more leaders are present in the system. $\Omega\$$ has the additional capacity to distinguish between the cases of exactly one leader and more than one leader in the system. $W\Omega?$ is only required to report its guesses about leaders "fixed" at some agents (in contrast, $\Omega?$ reports its guesses about a leader even if the agent bearing a leader mark is constantly changing). To demonstrate the

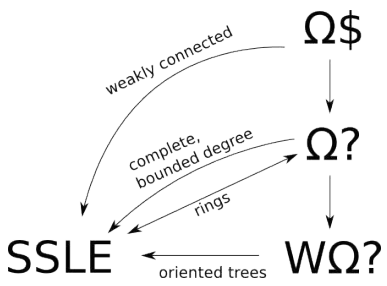

Figure 1: An arrow $\Theta_1 \xleftarrow{\mathcal{F}} \Theta_2$ depicts that Θ_1 has a population protocol implementation using Θ_2. In other words, Θ_1 is weaker than Θ_2 over a family of graphs $\mathcal{F}$ (i.e. $\Theta_1 \preccurlyeq_{\mathcal{F}} \Theta_2$ as defined in Sec. 2)

.

relative power of these oracles, we solve $SSLE$ using each of them separately in different communication topologies. In particular, we prove that $SSLE$ can be implemented with $\Omega\$$ over weakly connected communication graphs (Sec. 4), with $W\Omega?$ over oriented trees (Sec. 5) and with $\Omega?$ over weakly connected communication graphs of bounded degree (Sec. 6). Finally, we show that $\Omega?$ can be implemented using $SSLE$ over rings, proving their equivalence (Sec. 7). When the two former results related to $\Omega\$$ and $W\Omega?$ are more intuitive and simpler, the two latter ones related to $\Omega?$ are less intuitive and much more intricate. Due to the lack of space, almost all proofs are sketched. All complete proofs appear in the appendix.

All these paper results allow to establish relations between the different oracles and $SSLE$, which we summarize in Fig. 1.

Related Work. Being an important primitive in distributed computing, leader election has been extensively studied. Below, we mention only the most relevant works to the current paper.

It was shown, e.g. in [2, 5], 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., [3, 4]). Nevertheless, $SSLE$ is impossible in population protocols over general connected communication graphs [3]. Together with that, [3] presents a non-uniform solution for $SSLE$ on rings. A uniform algorithm for rings and complete graphs is proposed in [13], but uses $\Omega?$. Recently, [6] 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, unknown to agents. For the enhanced model of *mediated population protocols* [15], the work of [16] shows that $(2/3)n$ agent states are sufficient to solve $SSLE$. In [7], 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.

2 Model and Definitions

2.1 Population Protocol

We use the same definitions as in [13] with some slight modifications. A network is represented by a directed graph $G = (V, E)$ 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 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 network is weakly connected.

A *population protocol* $\mathcal{A}(\mathcal{Q}, X, \delta)$ consists of a finite state space $\mathcal{Q}$, a finite input alphabet X 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 $\mathcal{Q}^2$. A *(transition) rule* of the protocol is a tuple $(q_1, x_1, q_2, x_2, q_1', q_2')$ such that $(q_1', q_2') \in \delta(q_1, x_1, q_2, x_2)$ and is denoted by $(q_1, x_1)(q_2, x_2) \to (q_1', q_2')$. The population 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 $C : V \to \mathcal{Q}$ specifying the state of each agent in the network, and an

input assignment is a mapping $\alpha : V \to X$. A *trace* T on a graph $G(V, E)$ is an infinite sequence of assignments from V to the set Z, i.e., $T = \alpha_1\alpha_2 \ldots$ where $\alpha_i : V \to Z$. When $Z = X$, then each α_i is an input assignment, and we say T is an input trace of the protocol. The trace is *constant* if $\alpha_1 = \alpha_2 = \ldots$. The trace is *uniform constant* if it is constant and the common assignment assigns the same value to every node in the network, i.e., $\alpha = \alpha_1 = \alpha_2 = \ldots$ and for every $u, v \in V$, $\alpha(u) = \alpha(v)$.

An *action* is a pair (e, r) where r is a rule $(q_1, x_1)(q_2, x_2) \to (q_1', q_2')$ and $e = (u, v)$ an edge of G. Let C, C' be configurations, α be an input assignment, and u, v be distinct nodes. 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 $w \in V - \{u, v\}$. In other words, C' is the configuration that results from C by applying the transition rule r to the node pair e. We note $(C, \alpha) \to C'$ 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 $C_0C_1 \ldots C_k$ such that $C = C_0$, $C' = C_k$ and $(C_i, \alpha_i) \to C_{i+1}$ for all $0 \leq i < k$, in which case we say that C' is *reachable* from C given the input trace T_{in}.

A *virtual execution* is an infinite 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, $(C_i, \alpha_i) \xrightarrow{\sigma_i} C_{i+1}$. An *execution* is a virtual execution that satisfies the global fairness condition:

(Global Fairness) a virtual execution $(C_0, \alpha_0, \sigma_0)(C_1, \alpha_1, \sigma_1) \ldots$ is globally fair when, for every C, C', α such that $(C, \alpha) \to C'$, if $(C, \alpha) = (C_i, \alpha_i)$ for infinitely many i, then $C' = C_j$ for infinitely many j.

If E is an execution, we denote by SE the (infinite) suffix of execution of E such that each couple (C, α) (C being a configuration, and α an input assignment) in SE occurs infinitely often in SE. This suffix is well-defined because the number of couples (C, α) that occurs finitely often in E is bounded. Then, we denote by IRC_E the set of configurations that occur in SE, i.e., the set of configurations that occur infinitely often in E. IRC stands for Infinitely Recurring Configurations.

An *output map* is a mapping $O : \mathcal{Q} \to Y$ from the state space to some finite output alphabet Y. Such an output map is extended to take a configuration C and produce an *output assignment* $O(C)$ defined as $O(C)(v) = O(C(v))$. 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 $T_{out} = O(C_0)O(C_1) \ldots$.

2.2 Run, Behaviour, Oracle and Implementation

The definitions of runs, behaviours and oracles that we give below are different from the ones in [13, 3] and are required to obtain a proper framework (independent of real time) for defining oracles and establishing relations between them. For instance, in this framework, the oracles are self-implementable, in contrast with the traditional failure detectors' frameworks. The following notion of *compatibility of a trace with a schedule* is important the framework, because it involves that the changes in a trace are only caused by the interactions.

A *schedule* on a network $G(V, E)$ is a sequence of edges $S = e_1e_2 \ldots$, i.e., $e_i \in E$ for all i. Given a graph G, a trace $T = \alpha_0\alpha_1 \ldots$ on G is said to be *compatible* with the schedule $S = (u_0, v_0)(u_1, v_1) \ldots$ on G if, for every i, for every $w \in V - \{u_i, v_i\}$, we have $\alpha_i(w) = \alpha_{i+1}(w)$. That is, two consecutive assignments of a compatible trace can differ only in the assignment values of the two agents in the corresponding edge in the schedule. Note that, by the definition of population protocols, the output trace induced by any execution with schedule S of any population protocol on G is necessary compatible with S.

A *run* $R(X, Y)$ with input alphabet X and output alphabet Y on a network $G(V, E)$ is a triple (T_{in}, T_{out}, S) where T_{in} is a trace with alphabet X on G, T_{out} is a trace with alphabet Y on G and S a schedule on G such that T_{in} and T_{out} are both compatible with S. A *behaviour* B is given by a family $\mathcal{D}(B)$ of graphs (the domain of B), an input alphabet X, an output alphabet Y and a function that maps any graph G in $\mathcal{D}(B)$ to a set $B(G)$ of runs with input alphabet X and output alphabet Y.

We define the notion of *composition* of behaviours. Consider two behaviours B_1, B_2 with input

alphabets X_1, X_2, output alphabets Y_1, Y_2 and a family $\mathcal{F} \subset \mathcal{D}(B_1) \cap \mathcal{D}(B_2)$. We denote by T_X a trace with values in X. The *parallel composition* $B = B_1||B_2$ is the behaviour with alphabets $X_1 \times X_2$, $Y_1 \times Y_2$ such that, for every $G \in \mathcal{F}$, $B(G)$ is the set of runs $((T_{X_1}, T_{X_2}), (T_{Y_1}, T_{Y_2}), S)$ with $(T_{X_1}, T_{Y_1}, S) \in B_1(G)$ and $(T_{X_2}, T_{Y_2}, S) \in B_2(G)$. If $Y_1 = U \times V$ and $X_2 = V \times W$, the *serial composition* $B = B_2 \circ_V B_1$ *over* V is the behaviour over alphabets $X_1 \times W$, $U \times Y_2$. For every $G \in \mathcal{F}$, $B(G)$ is the set of runs $((T_{X_1}, T_W), (T_U, T_{Y_2}), S)$ for which there exists a trace T_V such that $(T_{X_1}, (T_U, T_V), S) \in B_1$ and $((T_V, T_W), T_{Y_2}, S) \in B_2$. If $X_1 = U \times V$ and $Y_1 = U \times W$, the *self composition* $B = Self(B_1/U)$ *on* U is the behaviour with alphabet V, W, where, for every $G \in \mathcal{F}$, $B(G)$ is the set of runs $((T_U^{in}, T_V^{in}), (T_U^{out}, T_W^{out}), S) \in B$ such that $T_U^{in} = T_U^{out}$. Given a (possibly infinite) family $\mathcal{B}$ of behaviours, a *composition involving the behaviours* $\mathcal{B}$ is either a behaviour in the family $\mathcal{B}$, or the parallel, serial or self composition of compositions involving the behaviours $\mathcal{B}$.

A behaviour B_2 *implements a behaviour* B_1 *over a family* $\mathcal{F}$ *of graphs* when the family $\mathcal{F} \subset \mathcal{D}(B_1) \cap \mathcal{D}(B_2)$, and for every graph $G \in \mathcal{F}$, $B_2(G) \subset B_1(G)$. Given a population protocol $\mathcal{A}$ with input alphabet X and output alphabet Y, we define the *behaviour* $B\mathcal{A}$ *associated to protocol* $\mathcal{A}$ as follows. Its domain is all the graphs, the input alphabet is X, the output alphabet is Y, and, for any graph G, for any run (T_{in}, T_{out}, S) on G, we have $(T_{in}, T_{out}, S) \in B\mathcal{A}(G)$ if and only if there exists an execution of $\mathcal{A}$ on G with input trace T_{in} and schedule S that induces the output trace T_{out}.

We say that a behaviour B_1 is *weaker* than a behaviour B_2 over $\mathcal{F}$, which we denote by $B_1 \preccurlyeq_{\mathcal{F}} B_2$, when there exists a composition B involving B_2 and population protocol behaviours such that B implements B_1 over $\mathcal{F}$. The two behaviours are *equivalent* if $B_1 \preccurlyeq_{\mathcal{F}} B_2$ and $B_2 \preccurlyeq_{\mathcal{F}} B_1$. A *problem* and an *oracle* are simply defined as behaviours. A population protocol $\mathcal{A}$ *solves a problem* P (resp. *implements an oracle* Θ) *using the behaviour* B *over a family* $\mathcal{F}$ *of graphs* if there exists a composition involving B, the behaviour $B\mathcal{A}$ associated to $\mathcal{A}$ and other population protocol behaviours, that implements the behaviour P (resp. Θ) over $\mathcal{F}$.

Note that with these definitions, any oracle Θ implements itself. Moreover, if there exists a population protocol that solves the problem P_1 using the problem P_2, then $P_1 \preccurlyeq_{\mathcal{F}} P_2$.

3 Specific Behaviours, Oracles and Tools

3.1 Self-Stabilizing Leader Election Behaviour

The self-stabilizing leader election behaviour $SSLE$ is defined as follows. The domain of the behaviour is all the graphs, the input alphabet is $\{\perp\}$ (no input), the output alphabet is $\{0, 1\}$ and a run $(\perp, T, S)$ belongs to $SSLE$ if and only if T has a constant suffix $T' = \alpha\alpha\alpha\ldots$ and there exists a node 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.

3.2 Oracles $\Omega?$ and $\Omega\$$

Here, we present and formally define the oracles we use with our protocols. The first one is the oracle of Fischer and Jiang [13]. It has the particularity not to distinguish between the presence of one or more leaders in a configuration. This is sufficient for solving $SSLE$ in complete graphs, because two leaders are always neighbors, and eventually "kill" each other. In rings, a rather elaborated mechanism (together with the global fairness) allows to cancel supplementary leaders, without knowing their number. The situation is different in an arbitrary graph (non ring). Indeed, consider the case where leaders are fixed to nodes. Due to a greater number of paths in the network, it is difficult for a leader to send an information to kill other leaders while also being protected against it. The leaders could move to meet and kill each other, but then no leader could ever be fixed at some node unless it eventually knows it is unique. That is the reason why we introduce $\Omega\$$, which distinguishes between one and more than one leaders.

$\underline{\Omega?:}$ The oracle $\Omega?$ is defined as follows. The input alphabet is $\{0, 1\}$ and the output alphabet is $\{0, 1\}$. The domain of $\Omega?$ is all the graphs. Given an assignment α, we denote by $l(\alpha)$ the number

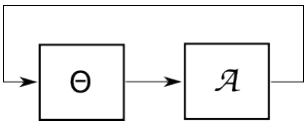

Figure 2: Serial composition $\Theta \circ \mathcal{A}$ followed by a self composition.

of vertices that are assigned the value 1 by α. Given a graph G and a run (T_{in}, T_{out}, S) on G, we have $(T_{in}, T_{out}, S) \in \Omega?(G)$ when the following conditions hold. If T_{in} has a suffix $\alpha_0\alpha_1\dots$ such that $\forall s, l(\alpha_s) = 0$, then T_{out} has a suffix equal to the uniform constant trace 0. If T_{in} has a suffix $\alpha_0\alpha_1\dots$ such that $\forall s, l(\alpha_s) \geq 1$, then T_{out} has a suffix equal to the uniform constant trace 1. Otherwise, any T_{out} is legal.

$\underline{\Omega\$}$: The oracle $\Omega\$$ is defined as follows. The input alphabet is $\{0,1\}$ and the output alphabet is $\{0,1,2\}$. The domain of $\Omega\$$ is all the graphs. Given a graph G and a run (T_{in}, T_{out}, S) on G, we have $(T_{in}, T_{out}, S) \in \Omega\$(G)$ when the following conditions hold. If T_{in} has a suffix $\alpha_0\alpha_1\dots$ such that $\forall s, l(\alpha_s) = 0$, then T_{out} has a suffix equal to the uniform constant trace 0. If T_{in} has a suffix $\alpha_0\alpha_1\dots$ such that $\forall s, l(\alpha_s) = 1$, then T_{out} has a suffix equal to the uniform constant trace 1. If T_{in} has a suffix $\alpha_0\alpha_1\dots$ such that $\forall s, l(\alpha_s) \geq 1$, then T_{out} has a suffix with values in $\{1,2\}$. If T_{in} has a suffix $\alpha_0\alpha_1\dots$ such that $\forall s, l(\alpha_s) \geq 2$, then T_{out} has a suffix equal to the uniform constant trace 2. Otherwise, any T_{out} is legal.

There are some subtleties in the definitions of the previous oracles. Indeed, with $\Omega?$, the condition for the oracle to give meaningful information is that the number of leaders satisfies some condition *at every step*, e.g. $l \geq 1$. This implies that this oracle is able to detect "moving" leaders. The oracle below is weaker in the sense that it gives information about fixed leaders. Given a trace T, we denote by $\lambda(T)$ the number of vertices that are permanently assigned the value 1 during T.

$\underline{W\Omega?}$: The oracle $W\Omega?$ is defined as follows. The input alphabet is $\{0,1\}$ and the output alphabet is $\{0,1\}$. The domain of $W\Omega?$ is all the graphs. Given a graph G and a run (T_{in}, T_{out}, S) on G, we have $(T_{in}, T_{out}, S) \in W\Omega?(G)$ when the following conditions hold. If T_{in} has a suffix T'_{in} such that $\lambda(T'_{in}) = 0$, then T_{out} has a suffix equal to the uniform constant trace 0. If T_{in} has a suffix T'_{in} such that $\lambda(T'_{in}) \geq 1$, then T_{out} has a suffix equal to the uniform constant trace 1. Otherwise, any T_{out} is legal. It is straightforward to check that $W\Omega? \preccurlyeq \Omega? \preccurlyeq \Omega\$$ over all the graphs.

3.3 Tools

In Sec. 4 (resp. Sec. 6), we present protocols that implement $SSLE$ using $\Omega\$$ (resp. $\Omega?$) over the family of *strongly* connected graphs (resp. strongly connected graphs of bounded degree). Actually, in each case, we present a population protocol, say $\mathcal{A}$, and prove that the behaviour given by the composition $Self(\mathcal{A} \circ \Theta)$ (cf. Fig. 2), where Θ is the corresponding oracle, implements $SSLE$.

Below, we explain how to extend these results to the family of *weakly* connected graph (resp. weakly connected graphs of bounded degree). The procedure applies to both protocols. Consider a population protocol $\mathcal{A}$. First, we define the non-deterministic protocol $\mathcal{A}^{ND}$ with the same state space and input alphabet 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 $\mathcal{A}$. In other words, there is a non-deterministic choice that selects which agent is the initiator, and which is the responder, in a rule of $\mathcal{A}$. Given a weakly connected graph G, the symmetric closure G_{sym} of G is necessarily a strongly connected graph (with the same maximal degree). 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[1] 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 $SSLE$ on G_{sym} using an oracle Θ such that $\Theta(G) = \Theta(G_{sym})$, then $\mathcal{A}^{ND}$ solves $SSLE$ on G using the

[1] If $\sigma_i = (u_i, v_i, (q,y)(p,x) \to (q',p'))$ with $(p,x)(q,y) \to (p',q')$ 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 $\sigma'_i = \sigma_i$.

oracle Θ. It is then possible to transform $\mathcal{A}^{ND}$ into a deterministic protocol that implements $SSLE$ using Θ over G. It can be done, for instance, by using the general deterministic transformer in [3], since in terms of [3], $\mathcal{A}^{ND}$ implements an *elastic behaviour*.

In Sec. 7, we present a protocol *RingDetector* that implements the oracle $\Omega?$ over the family of oriented rings, thus proving the equivalence of $\Omega?$ and $SSLE$ over oriented rings. This result is straightforward to extend to non-oriented rings thanks to the self-stabilizing ring orientation protocol presented in [3]. Finally, as a basic tool for our protocol in graphs of bounded degree, we use the 2-hop coloring self-stabilizing population protocol, denoted $2HC$ and presented in [3]. A 2-hop coloring is a coloring such that all neighbors of the same node have distinct colors.

4 $SSLE$ using $\Omega\$$ over Weakly Connected Graphs

In this section, we show that $SSLE$ can be implemented using $\Omega\$$ over the family of weakly connected graphs. According to Sec. 3.3, it is sufficient to prove the result over strongly connected graphs. The idea of the protocol is simple. A leader moves when it "knows" there are other leaders and does not move when it "knows" it is the unique leader, this information being provided by the oracle. We define the protocol $\mathcal{A}$ as follows. The input alphabet is $\{0,1,2\}$, the state space is $\{\bullet,\circ\}$ where $\bullet$ (resp. $\circ$) stands for leader (resp. non leader). The rules are : (1) $(\circ,0)(\circ,0) \to (\bullet,\circ)$, (2) $(\bullet,2)(\circ,2) \to (\circ,\bullet)$, and (3) $(\bullet,*)(\bullet,*) \to (\bullet,\circ)$. The symbol $*$ means "any possible value". In every other cases, the states are unchanged. Basically, a leader is created whenever the oracle outputs 0 (rule (1)). The leaders keep moving in the graph while the oracle outputs 2 (rule (2)). When two leaders meet, one of them disappears (rule (3)).

Theorem 1. *The protocol $\mathcal{A}$ implements SSLE using $\Omega\$$ over strongly connected graphs.*

Proof Sketch. See Appendix A, Theorem A for details. Given a globally fair execution E with input trace T_{in} (output of $\Omega\$$), there is eventually a configuration with a leader. Otherwise, the trace T_{in} would have a uniform constant suffix 0, and rule (1) would create a leader. In addition, once there is a leader, there is a leader in every subsequent configuration. Hence, T_{in} must have a suffix T'_{in} with values in $\{1,2\}$, which prevents leader creation; hence, the number of leaders is eventually constant equal to $c \geq 1$. If $c \geq 2$, the input trace would have a uniform constant suffix 2, and two leaders may move (rule (2)) to meet and one of them would disappear (rule (3)); whence a contradiction. Hence, $c = 1$ and the input trace has a uniform constant suffix 1, which permanently fixes the unique leader at some place. □

5 $SSLE$ using $W\Omega?$ over Oriented Trees

In this section, we show that $SSLE$ can be implemented using the oracle $W\Omega?$ over the family of oriented trees. We assume that the tree is oriented from the root to the leaves. Note that an agent does not know whether it is a root or a leaf. The idea consists in creating leaders when the oracle outputs 0 and making them migrating towards the root. When two leaders interact, they "merge". From some point on, the root of the tree is a permanent leader. We define the protocol $\mathcal{B}$ as follows. The input alphabet is $\{0,1\}$, the state space is $\{\bullet,\circ\}$ where $\bullet$ (resp. $\circ$) stands for leader (resp. non-leader). The rules are: (1) $(*,0)(*,*) \to (\bullet,\circ)$, (2) $(*,*)(*,0) \to (\circ,\bullet)$, and (3) $(*,*)(\bullet,*) \to (\bullet,\circ)$. The proof of the following theorem is in Appendix B, Theorem B.

Theorem 2. *The protocol $\mathcal{B}$ implements SSLE using $W\Omega?$ over the family of oriented trees.*

6 $SSLE$ using $\Omega?$ over Weakly Connected Graphs with Bounded Degree

In this section, we present a more difficult result, namely that, for every integer d, the behaviour $SSLE$ can be implemented using $\Omega?$ over the family of weakly connected graphs with in/out-degree bounded

above by d. According to Sec. 3.3, it is sufficient to prove the result for the family $\mathcal{F}_d$ of strongly connected graphs with in/out-degree bounded above by d.

The 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. Then a leader must try to kill possible other leaders, without killing itself. This image comes from Fischer and Jiang, leaders sending bullets for killing other leaders. Although the protocol of Fischer and Jiang is not simple, the ring topology is of great help. For arbitrary graphs, managing bullets is much more complicated, and agents must in some sense keep a trace of them. As the agents are finite-state, graphs of bounded degree are necessary for implementing such a management. Our solution uses the 2-hop coloring $2HC$ self-stabilizing protocol (Sec. 3.3). We note $Colors$ the corresponding set of colors.

(Protocol). We define the population protocol $\mathcal{C}_d$. The input variables (read-only) at each node x are: the *oracle output* $\Omega?_x$ (values in $\{0,1\}$), the *node color* c_x (values in $Colors$). 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 protocol is given in Algorithm 1. 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-neighbor (resp. in-neighbor) of specific color; this information is encoded in the shield and bullet vectors. The purpose of the bullets is to kill leaders, whereas the purpose of the shields is to protect them by absorbing bullets. When a leader is created, it comes with shields for every color, 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. Dually, any time an agent receives a bullet, it reloads bullets for every color. In addition, whenever a leader interacts as an initiator, it loads bullets for every color.

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

```
(Create a leader at x, if needed)
if Ω?_x = 0 then
    leader_x ← 1
    ∀c ∈ Colors, bullet_x[c] ← 1
    ∀c ∈ Colors, shield_x[c] ← 1
end
(Create a leader at y, if needed)
if Ω?_y = 0 then
    leader_y ← 1
    ∀c ∈ Colors, bullet_y[c] ← 1
    ∀c ∈ Colors, shield_y[c] ← 1
end
(Move bullet from y to x, if any)
if bullet_y[c_x] = 1 then
    if shield_x[c_y] = 0 then
        leader_x ← 0
        ∀c ∈ Colors, bullet_x[c] ← 1
    bullet_y[c_x] ← 0
end
(Move shield from x to y, if any)
if shield_x[c_y] = 1 then
    shield_y ← 1_{H_y^+}
    bullet_y[c_x] ← 0
    shield_x[c_y] ← 0
end
(Charge bullets if x is a leader)
if leader_x = 1 then
    ∀c ∈ Colors, bullet_x[c] ← 1
```

(Proofs). 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). Consider a strongly connected graph G of degree (both in and out) less than or equal to d.

A path in G is a sequence of nodes $\pi = x_0 \dots x_r$ such that (x_i, x_{i+1}) is an edge of G. If u is an agent, we note $u \in \pi$ to say that the agent u appears in the path π and we note $ind_\pi(u)$ 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 the graph, 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 node λ and a loop $\pi = x_0 \dots x_{r+1}$ at λ (a path that starts and ends at $x_0 = x_{r+1} = \lambda$). We say that λ is a* leader protected in π *when λ is a leader and there exists $i \in \{0, \dots, r\}$ such that x_i has a shield against x_{i+1} and, if $i \geq 1$, is a non-leader that has no bullet*

against x_{i-1}. In addition, for every $j \in \{1, \ldots, i-1\}$, 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 $x_0 \ldots x_i$ is the* protected zone in π*. The node λ is a* protected leader *if it is protected in every loop at λ.*

Note that a a leader that receives a shield or that has just been created becomes protected since it loads shields for every color.

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

Proof Sketch. For full details, see Appendix C, Lemma 1. It is sufficient to show that, for any input assignment α and any configuration C' such that $(C, \alpha) \to C'$, the configuration C' has a protected leader. We note (u, v) the pair of agents involved, and λ a protected leader in C. When a leader is created, it is already protected. Thus, we examine the other cases. Consider a loop π at λ. If u and v are not in the protected zone, then after the transition, the states of the agents in the protected zone are not modified; hence λ is still protected in π. Now, assume first that the path uv is not in π. If u is in the protected zone, u may only receive a bullet from v which would threaten λ. However, such a case implies the existence of a loop (the one that goes from λ to u, then v to λ along any path) in which λ is not protected. If v is in the protected zone, v may only receive a shield which does not threaten λ. In both cases, the leader λ is still protected in π. Assume now $uv \in \pi$. If u is not the protector, then the same arguments above apply. If u is the protector, it might transfer its shield to v while v is a leader. In that case, in C', λ is not protected, but v is. □

Lemma 2. *If no configuration in IRC_E has a leader, then all input assignments in SE 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 SE 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 $\Omega?$. □

The proofs of the following lemmas are given in Appendix C, Lemma C, D and E.

Lemma 3. *Every configuration in IRC_E has at least one leader, and every input assignment in SE is equal to an input assignment α^E that assigns 1 to every variable $\Omega?_x$ and yields a 2-hop coloring.*

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

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

Theorem 3. *The protocol $\mathcal{C}_d$ solves the problem $SSLE$ using $\Omega?$ over strongly connected graphs with degree less than or equal to d.*

Proof Sketch. See Appendix C, Theorem C for full details. Any configuration in IRC_E has the same number l of (protected) leaders. Assume that $l \geq 2$, take two protected leaders λ_1, λ_2 and consider the loop π made 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 bullet, it is possible to kill λ_1; whence a contradiction. Thus, there is a unique leader. □

It is important to notice that $W\Omega?$ is not sufficient. Indeed, consider a ring and the scenario depicted in Fig. 3, and assume we use $W\Omega?$ instead of $\Omega?$. In this scenario, there are initially only two protected leaders (1) and (2), and $W\Omega?$ outputs 1 everywhere. Then the shield protecting (2) moves behind (1), (2) becomes unprotected, and (1) fires a bullet that kills the leader (2). Then $W\Omega?$ punctually outputs 0 at (2), which creates a new protected leader. Then the shield protecting (1) moves behind (2), (1) becomes unprotected and (2) fires a bullet that kills the leader (1). Then $W\Omega?$ punctually outputs 0 at (1), which creates a new protected leader. If we repeat this scenario, in any configuration, there is at least one leader, but there is no permanent leaders. In that case, we have no control on the output of the oracle $W\Omega?$. On the contrary, $\Omega?$ is forced to eventually permanently output 1 everywhere, which prevents the infinite repetition of this scenario.

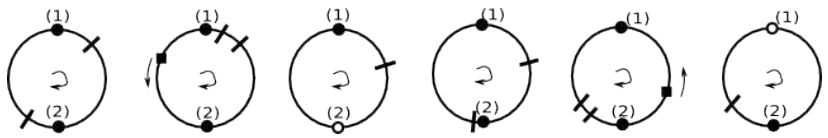

Figure 3: Scenario for which $W\Omega?$ is not sufficient.

7 $SSLE$ and $\Omega?$ are Equivalent over Rings

This equivalence is the second difficult result of the paper. According to Sec. 3.3, it is sufficient to focus on oriented rings. In [13], the authors show that $SSLE$ can be implemented using $\Omega?$ over oriented rings. In this section, we show that $\Omega?$ can be implemented given $SSLE$ over oriented rings. In other words, $\Omega?$ and $SSLE$ are equivalent over oriented rings. We emphasize that the protocol performing the implementation is self-stabilizing.

For the sake of clarity, the unique leader provided by $SSLE$ 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 network, that is to mimick $\Omega?$. We define the population protocol *RingDetector*. The input variables (read-only) at node x are: the *master bit* $master_x$ (values in $\{0,1\}$, from $SSLE$), the *leader bit* $leader_x$ (values in $\{0,1\}$). The working variables are: the *probe field* $probe_x$ ($\perp$, no probe, 0, white probe, 1, black probe), the *bullet field* ($\perp$, no bullet, 0, white probe, 1, black probe), the *flag bit* $flag_x$ (values in $\{0,1\}$) and the *output bit* (values in $\{0,1\}$). The protocol is given in Algorithm 2.

Each time an agent has its leader bit set to 1, it raises its flag. The master loads a white probe each time it is the responder of an interaction. The probes move counter-clockwise, and their purpose is to detect bullets already present in the network. When a probe meets a bullet, the probe becomes black. When two probes meet, they merge into a black probe if one of them was black, into a white probe otherwise. The master loads a bullet colored with its flag only when it receives a white probe. Bullets move clockwise. Each time a bullet meets an agent with its flag raised, the bullet becomes black and the flag is cleared. Two meeting bullets merge into a black bullet if one of them is black, and into a white bullet otherwise. When the master receives a bullet, it outputs 0 if the bullet is white, and 1 otherwise. In any interaction, the responder copies the output of the initiator, unless the responder is the master. In the sequel, 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 node λ (that depends on E) such that $\alpha_i(\lambda).master = 1$ for all i.

Lemma 6. *For any execution E, in any configuration $C \in IRC_E$, there is exactly one bullet (white or black) in C, i.e., there exists a unique node x such that $C(x).bullet \neq \perp$.*

Proof Sketch. See Appendix D, Lemma F for details. If there are no bullets in the system, then the master can fire a white probe that will return to the master without meeting any bullet, thus staying white and making the master fires a bullet. Once there is a bullet, there always is at least one bullet. Since bullets move clockwise and probes counter-clockwise, the master only receives black probes and thus permanently stops firing bullets. By global fairness, all the bullets eventually merge into a single bullet that never disappears. □

We now know that in the suffix SE there is a unique bullet moving clockwise. We divide the suffix in rounds defined as follows. A *round* begins with an interaction in which the master holds the bullet and is the initiator; the round ends with the first event in which the master is the responder and the initiator holds the bullet. In other words, a round corresponds to the bullet traveling through the whole ring before returning to the master.

Lemma 7. *Let R be a round in SE. We note $(C_0,\alpha_0)\ldots(C_r,\alpha_r)$ the corresponding sequence of configurations and input assignment. Case (a) If there are no leaders in the round, i.e., for every $0 \leq i \leq r$, and every agent x, we have $\alpha_i(x).leader = 0$, then after the last step, all the agents have their flags cleared. Case (b) If there are no leaders in the round, and if all the agents have their flags cleared at the*

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

```
1  (if the master is the responder, create a white probe)
2  if master_y = 1 then probe_y ← 0
3
4  (raise flags if needed)
5  if leader_x = 1 then flag_x ← 1
6
7  if leader_y = 1 then flag_y ← 1
8
9  (move probe from y to x)
10 if probe_y ≠ ⊥ then
11     (the probe becomes black when meeting a bullet)
12     if bullet_x ≠ ⊥ then probe_x ← 1
13
14     (and keeps the same color otherwise)
15     else probe_x ← probe_y
16
17     probe_y ← ⊥
18 end
19 (if the master has received a white probe, it loads a bullet)
20 if master_x = 1 and probe_x = 0 then bullet_x ← flag_x
21
22 (move bullet from x to y)
23 if bullet_x ≠ ⊥ then
24     (the bullet becomes black when meeting a flag)
25     if flag_y = 1 then bullet_y ← 1
26
27     (and keeps the same color otherwise)
28     else bullet_y ← bullet_x
29
30     (the flag is cleared)
31     flag_y ← 0
32     bullet_x ← ⊥
33 end
34 (if the master has received a bullet, it changes its output
   and whiten the bullet)
35 if master_y = 1 and bullet_y ≠ ⊥ then
36     out_y ← bullet_y
37     bullet_y ← 0
38 (for non-masters, the responder copies the output of the
   initator)
39 if master_y = 0 then out_y ← out_x
40
```

beginning of the round, then after the last step of the round, the master outputs 0 *and all the agents have their flags cleared. Case (c) If there is at least one leader at each step, 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 step of the round, the master outputs* 1*.*

Proof Sketch. We only describe the case (c) as the proof relies on the compatibility of the input trace with the schedule. For full details, see Appendix D, Lemma 7. Assume that there is a leader at each step. Let μ be an agent that holds a leader in assignment α_0. During the round, there must be some step i, such that $\mu = v_i$ is the responder and the initiator u_i holds the bullet. If μ holds a leader in assignment α_i, then after the transition, the bullet 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 step $j < i$ 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 bullet, so the flag cannot be cleared during the remaining steps until i. Hence, at step i, the bullet turns black when the bullet moves from the initiator u_i to the responder $v_i = \mu$. □

Theorem 4. *The protocol RingDetector is a self-stabilizing implementation of* $\Omega?$ *using SSLE over oriented rings.*

Proof Sketch. See Appendix D, Theorem D for full details. Consider a globally fair execution E and focus on the suffix SE. For the sake of simplicity, we assume that there is a unique master from the beginning. We divide the execution in rounds as defined above. If there are no leader forever, then Lemma 7 ensures that after a finite number of rounds, the master permanently outputs 0. If there is a leader at each step, then Lemma 7 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. □

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, M. J. Fischer, and H. Jiang. Self-stabilizing population protocols. *ACM Trans. Auton. Adapt. Syst.*, 3(4), 2008.

[4] 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.

[5] J. Beauquier, J. Burman, J. Clement, and S. Kutten. On utilizing speed in networks of mobile agents. In *PODC*, pages 305–314. ACM, 2010.

[6] 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.

[7] D. Canepa and M. G. Potop-Butucaru. Self-stabilizing tiny interaction protocols. In *WRAS*, pages 10:1–10:6, 2010.

[8] T. D. Chandra, V. Hadzilacos, and S. Toueg. The weakest failure detector for solving consensus. *J. ACM*, 43(4):685–722, 1996.

[9] T. D. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems. *J. ACM*, 43(2):225–267, 1996.

[10] B. Charron-Bost, M. Hutle, and J. Widder. In search of lost time. *Inf. Process. Lett.*, 110(21):928–933, 2010.

[11] A. Cornejo, N. A. Lynch, and S. Sastry. Asynchronous failure detectors. In *PODC*, pages 243–252, 2012.

[12] E. W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Commun. of the ACM*, 17(11):643–644, Nov. 1974.

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

[14] 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.

[15] O. Michail, I. Chatzigiannakis, and P. G. Spirakis. Mediated population protocols. *Theor. Comput. Sci.*, 412(22):2434–2450, 2011.

[16] 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 $SSLE$ using $\Omega\$$ over Weakly Connected Graphs

Theorem A. *The protocol $\mathcal{A}$ implements SSLE using $\Omega\$$ over strongly connected graphs.*

Proof. Consider a strongly connected graph G and consider a globally fair execution E of the protocol. Assume that every configuration C in IRC_E lacks a leader. The definition of $\Omega\$$ implies that every input assignment that occurs in SE assigns 0 to every agent. But, by rule (1), C can reach a configuration C' with a leader, and the global fairness ensures that $C' \in IRC_E$; whence a contradiction. Thus, there exists a configuration $C \in IRC_E$ that has a leader. The rule (3) (the only rule to kill a leader) implies that, for any input assignment α and for any configuration C' such that $(C, \alpha) \to C'$, C' has a leader. Now, consider any $C'' \in IRC_E$. By definition of SE, there must be a sequence of steps from (C, α) to (C'', α'') during SE, and the previous argument shows that every configuration during this sequence has a leader; in particular C''.

Thus, every configuration in IRC_E has at least one leader. The definition of $\Omega\$$ implies that any input assignment in SE does not assign 0 to any agent. Therefore, no leaders are created during SE. If there were two configurations in IRC_E with different number of leaders, then there would be a step in SE during which a leader is created; this is impossible. Hence, every configuration in IRC_E has the same number c of leaders. If $c \geq 2$, then the definition of the oracle implies that every input assignment in SE assigns 2 to every one. Since the graph is strongly connected, from any configuration $C \in IRC_E$ with $c \geq 2$ leaders, it is possible (via rule (2)) to move the two leaders to two neighbor nodes and to kill one of them (via rule (3)), thus reaching a configuration $C' \in IRC_E$ with less than c leaders; whence a contradiction. Hence, $c = 1$, i.e. there is a unique leader in every configuration in IRC_E. Then the definition of the oracle implies that every input assignment assigns 1 everywhere. Thus, during SE, the three rules of the protocol are disabled, and the unique leader is permanently located at some node. □

B $SSLE$ using $W\Omega?$ over Oriented Trees

Theorem B. *The protocol $\mathcal{B}$ implements SSLE using $W\Omega?$ over the family of oriented trees.*

Proof. Consider a globally fair execution. Let $SC = C_0C_1\ldots$ be the sequence of configurations and T_{in} the input trace of $\mathcal{B}$. If every configuration contains no leader, then the definition of the oracle implies that T_{in} has a suffix equal to the uniform constant trace 0. But then, the first two rules of the protocol $\mathcal{B}$ are enabled, and the global fairness ensures that a configuration with at least one leader is reached.

Note that the rules are such that no leaders are killed, except when two meeting leaders merge. This implies that once there is at least one leader, there is always at least one leader in the network. Furthermore, since the leaders keep migrating towards the root (via the last rule), at some point the root becomes a leader and stays so forever. In other words, the sequence SC has a suffix within which there is fixed permanent leader located at the root of the tree. Then, the definition of $W\Omega?$ implies that T_{in} has a suffix equal to the uniform constant trace 1. From that point on, no new leaders are ever created. The remaining leaders keep migrating towards the root, and eventually there is permanently a unique leader located at the root. □

C $SSLE$ using $\Omega?$ over Weakly Connected Graphs with Bounded Degree

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 SE, 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 SE. 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 bullets.

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 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 in protected in C; whence 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 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 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 steps from (C, α) to (C'', α'') during SE, 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 SE 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 SE 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 $\Omega?$ and the assumption on $2HC$. □

Lemma C. *Every configuration in IRC_E has at least one leader, and every input assignment in SE 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 SE assigns 0 to every $\Omega?_x$. Hence, in SE, 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 steps from (C, α) to (C', α') for some input assignments α, α', since both C and C' occur infinitely often in SE. According to the protocol, during

one of the steps, 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; whence 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 SE assigns 1 to every variable $\Omega?_x$. Thus no leader is created during SE. 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 steps in SE 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; whence 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 SE and α^E assigns 1 to every variable $\Omega?_x$. We describe a sequence of steps 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 bullet against x_{r-1}. If x_r is a leader, any transition where x_r is a initiator makes x_r creating a bullet against x_{r-1}. Then by moving (backward) the bullet 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 C. *The protocol $\mathcal{C}_d$ solves the problem $SSLE$ 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 step 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 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. □

D $SSLE$ and $\Omega?$ are Equivalent over Rings

Lemma F. *For any execution E, in any configuration $C \in IRC_E$, there is exactly one bullet (white or black) in C, i.e., there exists a unique node x such that $C(x).bullet \neq \perp$.*

Proof. Consider a configuration $C \in IRC_E$. We first prove that C contains at least one bullet. On the contrary, assume that, for every node x, $C(x).bullet = \perp$. The following scenario will produce a bullet. 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 nodes and return to λ again. Since there are no bullets in the network, the white probe will not turn black. Then, the white probe arriving at λ will make λ produce a bullet. This scenario does not depend on the possibly present leaders. Hence, we have shown that there exists a configuration C' with at least one bullet such that $C \xrightarrow{*} C'$, whatever the input trace is during this sequence. By the 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 bullet in the network, there is always at least one bullet in the network in any subsequent configuration. Thus C cannot occur infinitely often; whence a contradiction. Hence C has at least one bullet.

Assume now that C has at least two bullets. Since two meeting bullets merge into one bullet, there is a configuration C' with exactly one bullet 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 bullet is by having the master receive a white probe, this means that the master receives infinitely many white probes during SE. But once there is a bullet in the network, since the bullets move clockwise and the probes counter-clockwise, any probe arriving at the master must be black; whence a contradiction. Therefore, C has exactly one bullet. □

Lemma G. *Let R be a round in SE. We note $(C_0, \alpha_0) \dots (C_r, \alpha_r)$ the corresponding sequence of configurations and input assignment. Case (a) If there are no leaders in the round, i.e., for every $0 \leq i \leq r$, and every agent x, we have $\alpha_i(x).leader = 0$, then after the last step of the round, all the agents have their flags cleared. Case (b) If there are no leaders in the round, and if all the agents have their flags cleared at the beginning of the round, then after the last step of the round, the master outputs 0 and all the agents have their flags cleared. Case (c) If there is at least one leader at each step, 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 step of the round, the master outputs 1.*

Proof. Case (a). Assume there are no leaders in the round. Since the bullet moves clockwise from the master to the master, and since a bullet clears any flag it encounters, after the last step of the round, the bullet 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 step, the master holds the bullet 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 bullet meets no raised flags and stays white. At the end of the round, the master receives a white bullet and outputs 0. *Case (c).* Assume that there is a leader at each step. 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 step i, such that $\mu = v_i$ is the responder and the initiator u_i holds the bullet. If μ holds a leader in assignment α_i, then after the transition, the bullet 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 step $j < i$ 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 bullet, so the flag cannot be cleared during the remaining steps until i. Hence, at step i, the bullet turns black when the bullet moves from the initiator u_i to the responder $v_i = \mu$. In all cases, the master receives a black bullet at the end of the round, and thus outputs 1. □

Theorem D. *The protocol RingDetector is a self-stabilizing implementation of $\Omega?$ using SSLE over oriented rings.*

Proof. Consider a globally fair execution E and focus on the suffix SE. For the sake of simplicity, we assume that there is a unique master from the beginning. By Lemma 6, we know that in SE there is a unique bullet moving clockwise. Without loss of generality, we assume that SE begins with the bullet being hold by the master. We then write $SE = R_1 R_2 \dots$ where each R_i is a round.

Consider first the case where the input trace $T = \alpha_0\alpha_1\ldots$ in SE permanently assigns no leaders everywhere, i.e., for all i, for every agent x, $\alpha_i(x).leader = 0$. By Lemma 7, 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 bullet, and since this happens exactly at the end of a round, we know that 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 the agents permanently output 0.

Assume now that the input trace is such that there is at least one leader at every step. By Lemma 7, at the end of each R_1, 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. □