The Weakest Oracle for Symmetric Consensus in Population Protocols

Joffroy Beauquier, Peva Blanchard, Janna Burman, Shay Kutten

The Weakest Oracle for Symmetric Consensus in Population Protocols*

Joffroy Beauquier[1], Peva Blanchard[2], Janna Burman[1], and Shay Kutten[3]

[1] LRI, Paris-South 11 University, Orsay, France, {joffroy.beauquier,janna.burman}@lri.fr
[2] LPD, EPFL, Lausanne, Switzerland, peva.blanchard@epfl.ch
[3] Technion, Haifa, Israel, kutten@ie.technion.ac.il

Abstract. We consider the *symmetric consensus* problem, a version of consensus adapted to *population protocols*, a model for large scale networks of resource-limited mobile sensors. After proving that consensus is impossible in the considered model, we look for *oracles* to circumvent this impossibility. An oracle is an external (to the system) module providing some information allowing to solve the problem. We define a class of oracles adapted to population protocols, and we prove that an oracle in this class, namely $DejaVu$, allows to obtain a solution. Finally, and this is the major contribution of the paper, we prove that $DejaVu$ is the *weakest* oracle for solving the problem.

Keywords: networks of mobile sensors, population protocols, consensus, oracles, weakest oracle

1 Introduction

Consensus is a classical decision problem in distributed computing. In this problem, each process is given initially a value and has to take eventually an irreversible decision (*termination* condition). Processes must decide on a common value depending on the input values, according to *agreement* and *validity (non-triviality)* conditions [3, 17, 25].

Consensus-related problems are relevant to mobile sensor networks in many different contexts like, for example, flocking (see, e.g., [11]), swarm formation control (see, e.g., [26]), distributed sensor fusion (see, e.g., [22]) and attitude alignment (see, e.g., [16]). See also [21, 23, 24] for surveys and references on consensus-related problems in mobile wireless networks.

A fundamental result by Fisher, Lynch and Paterson [14] states that in the classical asynchronous message passing model, no deterministic algorithm for consensus exists, even in the case of a unique possible crash (halting) failure. It is not surprising that the same impossibility holds in the model of population protocols [2]. This model is fundamentally asynchronous, which is also one of the main reasons for the result in [14]. However, some inherent characteristics of population protocols make consensus even more difficult. The *agents*, i.e. the mobile processes in population protocols, are *uniform*, i.e. indistinguishable and executing the same code. They have a constant memory size and thus cannot neither obtain nor store labels or any other information depending on the network size. Agents communicate by asynchronous pair-wise interactions. No broadcast communication is available. Due to all these limitations, the agents are unable to detect which other agents are present

* This work has been partially supported by the Israeli-French Maimonide and the INS2I PEPS JCJC research projects.

but not interacting, even if no crash failure is possible. Hence, in population protocols, even with the assumption of absence of failures, consensus is impossible (Sec. 3).

Like in the message passing model, it seems interesting to study what is missing for solving consensus in population protocols. We adopt the point of view of Chandra and Toueg [10] for defining the possible missing information under the form of oracles, i.e., specific behaviours. Recall that an oracle can be thought as a collection of modules able to provide each process with some information, hopefully useful to solve a given problem. The *failure detectors* [10] are oracles that usually provide each process with failure-related information. In our case, such information seems meaningless since consensus is impossible to solve even without any failure. Moreover, the failure detectors of [10] cannot be used in our case, because they provide lists of process identifiers (estimated to have crashed). As already mentioned, identifiers are absent in population protocols (due to the constant size memory requirement).

Several identity-free oracles exist, though, in the literature. The failure detector introduced in [12] outputs a boolean value at every process, and solves the $(n-1)$-set agreement problem in n-process message passing system. However, this boolean value is mainly used to indicate wether or not the process is "alone", i.e., the other processes have all crashed. Thus, this failure detector does not fit in our case since we do not consider crash failures. Another type of oracles proposed in [19,20] (and used, e.g., in [5,6]) to deal with anonymity, provides information on the number of crashed processes (bounded by $f < n$), and, for the same reason of constant agent state space, cannot be used in the framework of population protocols. A so called "heartbeat" failure detector proposed in [1] requires to maintain unbounded counters at every process, and thus, again, is not suitable in our case. Some other failure detectors used to solve consensus and adapted to anonymous systems, like $A\Omega$ [6], $A\mathcal{L}$ and $A\Sigma'$ [7, 8], provide, roughly speaking, information about the number of correct processes, thus breaking the requirement of constant memory. In addition, in message-passing system, these oracles are used in combination with other powerful model assumptions and capabilities (e.g., "terminating" broadcast, unbounded process memory, etc.) which are unavailable in our case.

Thus, defining oracles in the context of population protocols appears as a real challenge. Several attempts have already been made, like the "eventual leader detector" of [13] (generalized in [4]). This oracle is useful to solve self-stabilizing leader election but, intuitively, is not reliable enough for solving the terminating consensus problem. Another interesting oracle proposed in [18] provides a "cover-time service" in the sense that a state hopping from agents to agents can know when it has covered the whole population. Based on this service, the authors propose an abstract oracle, namely an "absence detector", which is able to notify a specific agent about the absence or presence of some states in the population. This abstraction is adapted to study the computational power[1] of population protocols augmented with a cover-time service. Yet, this formulation is not helpful enough to assess the weakest oracle for a given task.

Due to all the above-mentioned reasons, we introduce a new type of oracles. The constraints we had in mind, when designing these oracles, are basically to make them implementable with minimum external assumptions on the system. Each oracle in the proposed class is distributed: it consists of a collection of local modules mapped onto the agents. Each agent's local module provides information only related to the past schedule, and is independent of the protocol being run (somewhat similarly to the classical failure detectors [10]).

[1] The class of functions computable by a terminating protocol.

To communicate this information, the agent's local module outputs a boolean value (as the failure detector in [12]).

Moreover, the proposed oracles are unreliable in the sense that the local modules are not required to provide this information at the precise time when it appears, but only *eventually*, *at least once* and *at least in one agent*. That is, the local modules may be very slow for some agents, and even completely dysfunctional (i.e., providing no information) for some others.

Finally, each oracle in our class is *anonymity-compliant*, in the sense that the information provided by the local modules does not depend on how these modules are mapped onto the agents. Roughly speaking, a permutation of the agents does not affect the possible output of these oracles (see Sec. 2.4 for precise definitions).

Besides this new class of oracles and the appropriately adapted computational model (Sec. 2), the paper presents three results. The first result (Sec. 3) states that consensus is impossible without an oracle. The second contribution (Sec. 4) is the presentation of an oracle in the class, called $DejaVu$, allowing a solution. These two results are relatively easy. The third result is intricate and is the main contribution of the paper (Sec. 5). It states that the proposed oracle is the *weakest* (see Sec. 2.5) for solving a *symmetric* version of consensus, a version adapted to population protocols (see Sec. 2.6). Intuitively, $DejaVu$ being the weakest oracle means that it provides the minimum required information for solving the problem (among all the oracles in the proposed oracle class).

2 Model and Definitions

2.1 Population Protocol

Here, we use the definitions of [2] with some slight modifications. A network is represented by a directed graph $G = (V, \mathcal{E})$ with n vertices and no multi-edges nor self-loops. Each vertex represents a finite-state sensing device called an *agent*, and an edge $(u, v) \in \mathcal{E}$ indicates the possibility of a communication (meeting/interaction) between two distinct nodes u and v in which u plays the role of the *initiator* and v of the *responder*. The orientation of an edge corresponds to this asymmetry in roles. We often write G instead of V to refer to the vertices of G. In this paper, for the sake of simplicity, we consider only *bidirectional complete* graphs: for any two vertices u, v there is an edge from u to v, and an edge from v to u. An edge e *involves* an agent u if u is an endpoint of e. Two edges are *independent* if they involve no common agent. Otherwise, they are *dependent*. To deal with permutation of agents, we also introduce the group $\mathfrak{S}G$ of permutations of the vertices of G.

A *population protocol* $\mathcal{A}(\mathcal{Q}, \mathcal{I}, Init, Input, \delta)$ consists of a finite *state space* $\mathcal{Q}$, a set $\mathcal{I}$ of *initial values*, a set $Input$ of *input values*, an *initialization map* $Init : \mathcal{I} \to \mathcal{Q}$, and a transition function $\delta : (\mathcal{Q} \times Input)^2 \to \mathcal{Q}^2$ that maps any tuple (q_1, v_1, q_2, v_2) to an element $\delta(q_1, v_1, q_2, v_2)$ in $\mathcal{Q}^2$. A *(transition) rule* of the protocol is a tuple $(q_1, v_1, q_2, v_2, q_1', q_2')$ where $(q_1', q_2') = \delta(q_1, v_1, q_2, v_2)$ and is denoted by $(q_1, v_1)(q_2, v_2) \to (q_1', q_2')$. We refer to $(q_1, v_1)(q_2, v_2)$ (resp. (q_1', q_2')) as the left (resp. right) part of the rule. Note that we only consider *deterministic* population protocols (the right part is uniquely determined by the left part of the rule).

Intuitively, the input values in $Input$ are provided to the agents by some external device like an oracle, continuously along the protocol execution. Besides, the initial values will correspond to the initial values to the consensus instance.

At the beginning of an execution, every agent is assigned an initial value from the set $\mathcal{I}$, formalized by the initialization map. In [2] a population protocol is defined to compute

a function of such initial values. In our case, the initial values represent the initial values in the consensus problem. Note that consensus, *a priori*, cannot be represented as a function of input values only (its output depends on the schedule too).

2.2 Schedules and histories

The characterization of the weakest oracle for consensus relies on a tight analysis of the causal order [15]. In contrast to the usual presentation of schedules as words of events, the following definition embeds an explicit formulation of the causal structure of the events. Formally, a *schedule* S is a (possibly infinite) sequence $E_1|E_2|E_3|\dots$ where the *Cartier-Foata condition* [9] holds: *(i)* every E_i is a subset of independent edges, *(ii)* for every i, every edge in E_{i+1} depends on some edge in E_i. The empty schedule is denoted by ϵ. When we write $E_1|E_2|\dots$, it is implicitly assumed that the sequence satisfies the Cartier-Foata condition; otherwise we simply write $E_1, E_2, \dots$. The *support of a schedule* S, denoted by $supp(S)$, is the subset of agents in G involved in the schedule S. Given a permutation $\alpha \in \mathfrak{S}G$, we write $\alpha S = \alpha E_1|\alpha E_2|\dots$ where $\alpha E_i = \{\alpha(e),\ e \in E_i\}$.

A schedule $S = E_1|E_2|\dots$ is a *factor* of a schedule $S' = E'_1|E'_2|\dots$ if there exists $i \geq 1$ such that $E_1 \subseteq E'_i$, $E_2 \subseteq E'_{i+1}$, and so on. We also say that S' *contains* S. If $i = 1$ above, and if S is finite, then S is a *prefix* of S', and S' is an *extension* of S.

An *event in* S is a pair $p = (i, e)$ such that $e \in E_i$. We denote by $\mathbb{P}(S)$ the set of events in S, which is naturally endowed with a partial order $\rightsquigarrow$ defined as the reflexive transitive closure of: $(i, e) \rightsquigarrow (i + 1, e')$ if and only if e, e' are dependent. Intuitively, the relation $\rightsquigarrow$ encodes the causal order of events during S. We often write e without mentioning the index i to refer to the event (i, e) when it is clear from the context.

The *causal past of* $p = (k, e)$ *in* S is the set $Past(p, S) = \{p' \in \mathbb{P}(S),\ p' \rightsquigarrow p\}$. We can equivalently write $Past(p, S) = F_1|\dots|F_k$, where $F_i = \{f,\ (i, f) \rightsquigarrow (k, e)\}$; note that necessarily $F_k = \{e\}$. Since these two presentations are equivalent, we will use the same notation to refer to them. Also, when S is clear from the context, we simply write $Past(p)$.

A finite schedule K is a *past cone* if there exists an event p in K such that all the events in K are in the causal past of p in K. If p involves an agent x, K is said to be a *past cone at* x.

Given a finite schedule S, an event $p = (i, (x, y))$, and an agent $z \neq x$, we say that x *meets indirectly* z at p in S if z is involved in some event in the causal past $past(p, S)$.

An infinite schedule S is *weakly fair* if and only if, for any pair of agents (x, y), there are infinitely many factors K of S such that x meets indirectly y at some event in K. Intuitively, this means that for any pair (x, y) the agent y causally influences the agent x infinitely often. Unless stated otherwise, every infinite schedule in this paper is assumed to be weakly fair.

A *history* $H = (S, h)$ with values in the set $Input$ is a schedule S together with a function h that associates with every event $p = (i, (x, y))$, a pair (v_x, v_y) of values from $Input$. The value v_x (resp. v_y) is the *history value*[1] *at* x *(resp. at* y*) in event* p. We also say that the history H *outputs the value* v_x *(resp.* v_y*) at* x *(resp.* y*) during* p. The schedule S is the *underlying schedule* of the history. The *support of* H is the support of its underlying schedule, $supp(H) = supp(S)$. Given a permutation $\alpha \in \mathfrak{S}G$, we write $\alpha H = (\alpha S, h')$ where $h'(i, \alpha(e)) = h(i, e)$. Finally, given a factor L of the schedule S, we denote by $H|_L = (L, h'')$ the *restriction of* H *to* L, where h'' is the restriction of h to the events occurring in L.

[1] These values will later be provided by an oracle.

2.3 Executions

Consider a (deterministic) population protocol $\mathcal{A}$ with state space $\mathcal{Q}$, input values set $Input$, and initial values set $\mathcal{I}$. A *configuration* γ is a function that associates with every agent x a state $\gamma(x)$ in $\mathcal{Q}$. For every assignment κ of initial values from $\mathcal{I}$ to the agents of the graph, we define the corresponding *initial configuration* $\gamma_\kappa = Init \circ \kappa$, where $Init$ is the initialization map of the protocol.

Let γ, γ' be configurations, E a subset of independent edges, and h a function labeling each edge in E with a pair of values from $Input$. We write $\gamma \xrightarrow{E,h} \gamma'$ when, for every edge $(x, y) \in E$ such that $h(x, y) = (v_x, v_y)$, $(\gamma(x), v_x)(\gamma(y), v_y) \to (\gamma'(x), \gamma'(y))$ is a rule of the protocol. In other words, γ' is the configuration that results from γ by applying the labeled events (E, h). Generally speaking, h represents the values provided by the environment as input to the protocol (the information provided by an oracle, in our case) during the corresponding transitions. We use these values to model the oracle output (Sec. 2.4). Note also that, since the edges are independent, the above definition is consistent.

An *execution* is a sequence $\gamma_0 \xrightarrow{E_0,h_0} \gamma_1 \xrightarrow{E_1,h_1} \dots$ such that the sequence $E_0, E_1, \dots$ satisfy the Cartier-Foata condition (i.e. it forms a schedule). The sequences $S = E_0|E_1 \dots$ and $h_0, h_1, \dots$ naturally yield a history $H = (S, h)$ where $h(i, e) = h_i(e)$. Since we deal with deterministic population protocols, an execution is entirely determined by a history H and an initial configuration γ_0. Hence we can denote an execution by $H[\gamma_0]$. The history H is referred to as the *input history* of the execution $H[\gamma_0]$.

Consider an output map $Out : \mathcal{Q} \to R$. The *output history* of the execution $H[\gamma_0]$ is the history $H' = (S, h')$ (with the same schedule as H) where h' associates with every event $e_i = (x, y) \in E_i$ the pair $(Out(\gamma_i(x)), Out(\gamma_i(y)))$. Intuitively, the output history is the history produced by the protocol during the execution.

2.4 Oracles

In the following, we define oracles having in mind the classical failure detectors of [10]. Informally, we think of an oracle as a collection of local modules, each of them being attached to an agent. These modules may be different and mapped onto the agents arbitrarily. Each module looks for specific predefined patterns of meetings in the causal past of an agent, and notifies the agent accordingly. A module never notifies wrongly an agent, but the notification can be arbitrarily delayed. Some modules may even never deliver their notifications.

More formally, an oracle is a function that associates with each graph G, each permutation $\sigma \in \mathfrak{S}G$ and each weakly fair schedule S on G, a set $O(G, \sigma, S)$ of *legal histories* that take values in $\{0, 1\}$ and all have S as their underlying schedule. These are the legal histories when the local modules are mapped to the agents according to σ.

More precisely, each oracle O is specified by a family of (possibly empty) sets, $Cones(O, G, \sigma, x)$, of finite schedules, for every complete graph G, every permutation σ of the vertices and every agent x in G. The set $Cones(O, G, \sigma, x)$ represents the patterns that will be looked for in the causal past of x. These sets satisfy: *i. (cone)* every schedule K in $Cones(O, G, \sigma, x)$ is a past cone at x; *ii. (anonymity-compliance)* for every permutation $\alpha \in \mathfrak{S}G$ of the agents, $K \in Cones(O, G, \sigma, x)$ if and only if $\alpha K \in Cones(O, G, \alpha\sigma, \alpha(x))$.

The anonymity-compliance condition ensures that the set of cones detectable at agent x in the original permutation σ ($K \in Cones(O, G, \sigma, x)$) is exactly the set of cones detectable at agent $\alpha(x)$ in the new permutation $\alpha\sigma$, up to a relabeling of the agents ($\alpha K \in Cones(O, G, \alpha\sigma, \alpha(x))$).

Then, a history H is a legal history of G with schedule S given the permutation σ (i.e. $H \in O(G, \sigma, S)$) if and only if *i. (safety)* if H outputs 1 at x in some event p in S, then the causal past $Past(p, S)$ contains some schedule from $Cones(O, G, \sigma, x)$; *ii. (liveness)* if the set of agents x, whose causal past contains at some point a schedule from $Cones(O, G, \sigma, x)$, is not empty, then the history H eventually outputs 1 at at least one of these agents in some event during S.

Intuitively, the safety property ensures that if O outputs 1 at x, then the corresponding prefix actually contains a schedule from $Cones(O, G, \sigma, x)$. The liveness property ensures that *at least one* agent is eventually notified about this fact.

Note that the set $Cones(O, G, \sigma, x)$ may be empty, which means that it is possible, *a priori*, for O to permanently output 0 at x. In particular, if all the defined sets $Cones(\dots)$ are empty, then the corresponding oracle always outputs 0 at every agent; which basically amounts to having no oracle at all, since the agents get no useful information.

2.5 Comparison between Oracles

We say that a protocol $\mathcal{A}$ *uses an oracle* O when the only considered executions of $\mathcal{A}$ are those whose input histories are legal histories of O.

Intuitively, an oracle O_1 is weaker than an oracle O_2 if there exists a population protocol that simulates a history of O_1 using O_2. Formally, an oracle O_1 is *weaker* than an oracle O_2 if there exists a population protocol $\mathcal{A}$ and an output map such that, for every execution $H[\gamma_0]$, H being a legal history of O_2, the corresponding output history H' is a legal history of O_1.

Stating that $H = (S, h)$ and $H' = (S, h')$ are legal histories of O_2 and O_1, respectively, means that there exist permutations σ and τ such that $H \in O_2(G, \sigma, S)$ and $H' \in O_1(G, \tau, S)$. However, the definition does not force σ and τ to be equal. Intuitively, it means that the history computed by the emulation provided by $\mathcal{A}$ is a legal history of O_1 up to a permutation of the local modules.

Given a family $\mathcal{F}$ of oracles, a *weakest oracle in* $\mathcal{F}$ is an oracle O that is weaker than every oracle in $\mathcal{F}$. Note that there is no evidence *a priori* that a weakest oracle exists. Note also that all the weakest oracles, if they exist, are equivalent.

2.6 Symmetric Consensus

Consider a population protocol $\mathcal{A}$ with initial values $\mathcal{I}$. We assume that the agents have an instruction *decide* which causes them to decide irreversibly on some value in $\mathcal{I}$.

The population protocol $\mathcal{A}$ (possibly using an oracle) is said to solve the *consensus* problem if, for each complete graph G, for each initial configuration γ, for any legal execution $H[\gamma]$, it satisfies: *i. (termination)* every agent eventually decides in the execution $H[\gamma]$; *ii. (agreement)* two agents cannot decide on different values; *iii. (validity)* if all the agents have the same initial value v, then an agent can only decide on v.

The protocol $\mathcal{A}$ is said to solve the *symmetric consensus* problem if it solves the consensus problem and, in addition, for each complete graph G, it satisfies an additional condition: *iv. (symmetry)* for any legal execution $H[\gamma]$, for any permutation $\alpha \in \mathfrak{S}G$ of the vertices, any agent decides on the same value in the execution $H[\gamma]$ and in $H[\gamma\alpha]$.

Intuitively, in the symmetric consensus, the decision value in an execution does not depend on the distribution of the initial values between the agents. Note that the condition of symmetry is quite natural for population protocols. In the seminal paper by Angluin et al. [2], the same condition applies to the predicates that are computable.

3 Impossibility of Consensus without Oracle

We first show that the consensus problem is impossible without an oracle. In particular, the symmetric consensus problem is also impossible. The proof relies on the well-known partitioning argument.

Before proceeding, we introduce a useful notation. Given two schedules $S = E_1|E_2|\dots$ and $S' = E'_1|E'_2|\dots$ such that each $E_i \cup E'_i$ is a subset of independent edges, we denote by $S \cup S'$ the schedule $E_1 \cup E'_1|E_2 \cup E'_2|\dots$.

Proposition 1. *Under weak fairness, there is no population protocol that solves the consensus problem over complete graphs.*

Proof. Assume that there exists a population protocol $\mathcal{A}$ that solves consensus over all complete graphs. Pick a complete graph G of $2 \cdot n$ agents (vertices), and select two complete subgraphs G_0, G_1 of n agents each. Let γ be the initial configuration of $\mathcal{A}$ corresponding to the agents in G_0 (resp. G_1) having the initial value 0 (resp. 1). Let S_v be a weakly fair schedule over G_v. By the validity condition of the consensus problem, in the execution $S_v[\gamma]$, all agents in G_v decide on the value v. Let S'_v be a finite prefix of S_v such that all the agents in G_v decide (on v) in the finite execution $S'_v[\gamma]$. Let S'' be any weakly fair extension of the schedule $S'_0 \cup S'_1$. This last schedule is well-defined since the two graphs are disjoint. Then, in the execution $S''[\gamma]$, the agents in G_0 decide on 0, and the agents in G_1 decide on 1; whence a contradiction with the agreement condition. □

4 Symmetric Consensus with $DejaVu$

Proposition 1 motivates the use of oracles. We define a particular oracle, called $DejaVu$. Intuitively, oracle $DejaVu$ outputs 1 at some agent x only when x has indirectly met (see definition in Sec. 2.2) every other agent at least once. Formally, a schedule K belongs to $Cones(DejaVu, G, \sigma, x)$ if and only if K is a past cone at x and $supp(K) = G$. The legal histories of $DejaVu$ are then defined according to the oracle rules.

The purpose of this section is to show that $DejaVu$ is sufficient to solve symmetric consensus. A simple protocol using $DejaVu$ is presented under the form of pseudo-code (Alg. 1), which is equivalent to the representation using transition rules.

We denote by $\mathcal{I}$ the set of initial values in the consensus problem. Every agent x has the following variables: val_x holding an estimate of the consensus value (set to the initial value of x at initialization), a boolean flag $decided_x$ (initially **false**), and a read-only boolean variable $done_x^{DV}$ holding the input value provided by the local module of $DejaVu$ at agent x. We assume that the set $\mathcal{I}$ is totally ordered. When two agents x and y meet, they both select the minimum of val_x and val_y as a new estimate of the consensus value. An agent x decides on its estimate when either its $DejaVu$'s local module outputs **true** ($done_x^{DV} =$ **true**), or agent x meets an agent y that has already decided ($decided_y =$ **true**); agent x then sets its flag $decided_x$ to **true**.

Lemma 1 (Termination and Validity). *Let $H \in DejaVu(G, \sigma, S)$ be a legal history and γ be an initial configuration. Then, in the execution $H[\gamma]$, every agent eventually decides on some initial value present in γ.*

Proof. Since an agent x can only decide on its estimate val_x, and since every update of val_x assigns a value of some agent, x can only decide on a value present in γ. The liveness

Algorithm 1: Symmetric consensus with $DejaVu$

```
1 done_x^DV : output of the oracle DejaVu at x
2 Initialization:
3    val_x ← a value in I
4    decided_x ← false
5 On a meeting event (x, y) of the agents x and y:
6    val_x ← min(val_x, val_y)
7    if ¬decided_x ∧ (done_x^DV ∨ decided_y) then
8        decide on val_x
9        decided_x ← true
```

property of the oracle $DejaVu$ and weak fairness imply that the oracle eventually outputs **true** at some agent x, which thus decides. Then, thanks to weak fairness, every agent will eventually indirectly meet x, and decide too (if it has not decided already). □

Lemma 2. *Let $H = (S, h)$ be any history with values in $\{0, 1\}$, and γ be an initial configuration. Consider the causal past $Past(p)$ for some event p in S, and let x be an agent involved in p. Then, at the end of the finite execution $H|_{Past(p)}[\gamma]$, the value of val_x at x is equal to the minimum of the initial values of the agents in the support of the causal past of p.*

Proof. For any event p in S, for any agent z involved in p, we denote by $val(p, z)$ the value of val_z right after p. We denote by $val(\perp, z)$ the initial value of the agent z.

Let x, y be the agents involved in the event p. Let p_x (resp. p_y) be the immediate predecessor[1] of p in $Past(p)$ that involves the agent x (resp. the agent y). If such an immediate predecessor does not exist (i.e. p is the first event involving x (resp. y)), then we set $p_x = \perp$ (resp. $p_y = \perp$). By line 6 in Alg. 1, $val(p, x) = \min(val(p_x, x), val(p_y, y))$. By iterating, we get $val(p, x) = \min\{val(\perp, z), z \in supp(Past(p))\}$. □

Lemma 3. *Consider Alg. 1 using $DejaVu$. Let $H \in DejaVu(G, \sigma, S)$ be a legal history of $DejaVu$, and γ an initial configuration. In the execution $H[\gamma]$ of Alg. 1, if some agent x' decides in some event p', then $supp(Past(p')) = G$.*

Proof. When x' decides, it is either because of the meeting with an agent which has already decided, or because the oracle has output 1 at x' (Alg. 1, line 7). Hence, there is an event p (in S) involving some agent x such that $p \rightsquigarrow p'$ and the oracle has output 1 at x during p (note that p and p' may be the same event).

By the safety property of $DejaVu$, $Past(p)$ contains some schedule from $Cones(DejaVu, G, \sigma, x)$. Hence, $supp(Past(p')) = supp(Past(p)) = G$, by the definition of $DejaVu$. □

Proposition 2. *Alg. 1 using $DejaVu$ solves the symmetric consensus.*

Proof. The termination and validity conditions are satisfied thanks to Lemma 1. The agreement and symmetry conditions are satisfied thanks to Lemma 2 and 3. □

[1] immediate means that, if p' involves x and $p_x \rightsquigarrow p' \rightsquigarrow p$, then $p' = p_x$ or $p' = p$.

5 Weakest Oracle for Symmetric Consensus

In this section, we prove an intricate property: any oracle O allowing to solve symmetric consensus can be used to implement $DejaVu$. With the result of Sec. 4, this proves that $DejaVu$ is the weakest oracle to solve symmetric consensus.

The following lemma states that if an oracle O allows to solve consensus then, for any (weakly fair) schedule, the corresponding "zero history", i.e., the history which always outputs 0 at every agent, cannot be a legal history. This shows, in particular, that any legal history of O eventually outputs 1 at at least one agent, which in turn implies that the set $Cones(O, G, \sigma, x)$ is not empty for at least one agent x.

The proof relies on a partitionning argument similar to the one used in the impossibility proof (Proposition 1). The argument though is not exactly the same because, *a priori*, the zero history may be a legal history only on a single specific graph. The partition argument usually consists in building an execution on a twice larger graph. To do so, we have to ensure that the history built on the larger graph is still legal.

Lemma 4. *Let $\mathcal{A}$ be a population protocol that solves the consensus[1] problem using an oracle O. For every graph G, and every permutation $\sigma \in \mathfrak{S}G$, for any weakly fair schedule S, the history H with schedule S which outputs the value 0 in every event is not a legal history, i.e., $H \notin O(G, \sigma, S)$. In particular, there exists an agent x such that $Cones(O, G, \sigma, x) \neq \emptyset$.*

Proof. We assume that there exist some graph G, and a legal history H of O on G which permanently outputs 0 everywhere, and we prove a contradiction. Let γ_0 (resp. γ_1) be the initial configuration where all the agents have initial value 0 (resp. 1). In $H[\gamma_v]$, every agent eventually decides on v. Let L be a prefix of S such that by the end of both executions $H[\gamma_0]$ and $H[\gamma_1]$, all the processes have decided.

Consider a graph G' containing two copies G_0 and G_1 of G (with disjoint sets of vertices). Let L_0 (resp. L_1) be the analog of the schedule L applied to G_0 (resp. G_1). Let S' be any weakly fair extension of $L_0 \cup L_1$ (see Sec. 3 for this notation), and $H' \in O(G', \sigma, S')$ be any legal history which outputs the value 0 during $L_0 \cup L_1$. This is possible since the oracle output can be arbitrarily delayed.

Consider now the initial configuration g' on G' in which all agents in G_0 (resp. G_1) have the initial value 0 (resp. 1). Then, by construction, in $H'[g']$, all agents in G_0 (resp. G_1) decide on 0 (resp. 1); whence a contradiction. This proves the first claim.

Moreover, assume that for some graph G, for some σ, for every agent x, $Cones(O, G, \sigma, x) = \emptyset$. Then, by anonymity-compliance, for every agent x, for every permutation τ, $Cones(O, G, \tau, x) = \emptyset$. Thus, the only legal history of O is the one always outputting 0 at every agent; which contradicts the first claim. This proves the second claim. □

The following crucial lemma shows that if O allows to solve symmetric consensus then the sets $Cones(O, \dots)$ defining O are the subsets of those defining $DejaVu$. In other words, the lemma states that the support of each past cone in $Cones(O, \dots)$ is the entire graph G. Informally, this means that, if O is strong enough to solve symmetric consensus, then O cannot notify an agent before this agent has indirectly met every other.

The proof relies on the fact that if an agent x can be notified before having indirectly met every other agent, then we can design a legal history H of O so that agent x decides without ever knowing about the initial value of some other agent y. In other words, the decision value

[1] Not necessarily symmetric, in this lemma.

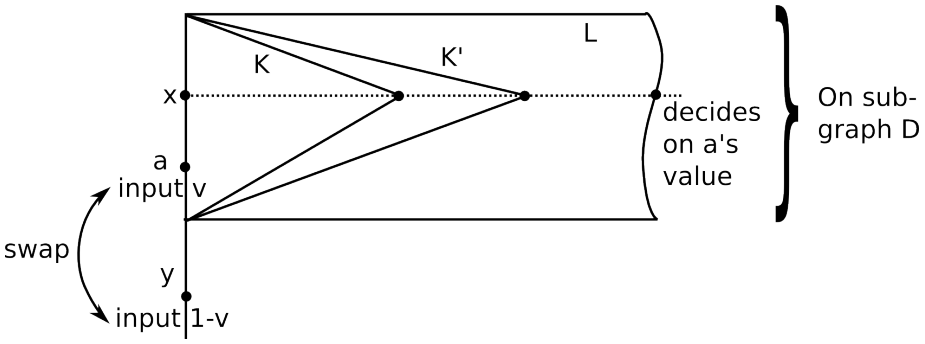

Fig. 1. Illustration to the proof of Lemma 5: x decides on the value v of a, but swapping the values of a and y makes x decide on the value $1-v$, what contradicts the definition of symmetric consensus.

of x is left unchanged if we flip agent y's initial value. Because of the validity condition of consensus, there exist two initial configurations g_0, g_1 which only differ by the initial value at one specific agent a, e.g., $g_0(a) = 1 - g_1(a) = 0$, and such that agent x decides on 0 (resp. 1) in the execution $H[g_0]$ (resp. $H[g_1]$). Obviously, a is not y. Let's assume that the initial value of y in g_0 (and g_1) is 1. By considering the configuration g obtained by swapping the initial values of a and y in the configuration g_0, we see that, on one hand, agent x has to decide on 0 by the symmetry condition of symmetric consensus, and, on the other hand, agent x has to decide on 1, since it cannot distinguish between the configurations g and g_1. This yields a contradiction. The main difficulty of the proof is to build a legal history of the (unknown) oracle O so that x decides without (indirectly) meeting some agent.

Lemma 5. *Let $\mathcal{A}$ be a population protocol that solves the symmetric consensus problem over all complete graphs using an oracle O. Then, for every complete graph G, every permutation σ, and every agent x in G, $Cones(O, G, \sigma, x) \subseteq Cones(DejaVu, G, \sigma, x)$.*

Proof. In this proof, for sake of clarity, we use the same notation for the initial value of an agent, and the corresponding initial state. Fig. 1 illustrates the core idea of the proof.

Assume that there is some schedule $K \in Cones(O, G, \sigma, x)$ that is not in $Cones(DejaVu, G, \sigma, x)$, i.e., K is a past cone at x whose support $D = supp(K)$ is a strict subgraph of G.

By Lemma 4, for some agent w (not necessarily distinct from x), the set $Cones(O, D, \sigma, w) \neq \emptyset$. Let $\alpha \in \mathfrak{S}G$ be the permutation that swaps x and w, and $\beta = \alpha\sigma$. Then, by the anonymity-compliance property of the cone sets (Sec. 2.4), $Cones(O, D, \beta, x) \neq \emptyset$. Thus, there is some $K' \in Cones(O, D, \beta, x)$.

Let S be any weakly fair extension of K on D containing the schedule K' as well. We build a history H with schedule S as follows: the history always outputs 0 everywhere except at x, for which it permanently outputs 1 only after the occurrences of K and K' in S. Since $K' \in Cones(O, D, \beta, x)$, we have $H \in O(D, \beta, S)$, i.e. H is a legal history of O on D.

For any initial configuration γ on D, we have an execution $H[\gamma]$ of $\mathcal{A}$ in which every agent in D decides. By the validity property of the consensus, if all the agents have the same initial value 0 (resp. 1), then all agents decide on 0 (resp. 1). Hence, there exist two initial configurations γ_0 and γ_1 on D such that, for some agent a in D, $\gamma_0(a) = 0$, $\gamma_1(a) = 1$ and for every $z \in D - \{a\}$, $\gamma_0(z) = \gamma_1(z)$, and the agents decide on the value 0 (resp. 1) in the execution $H[\gamma_0]$ (resp. $H[\gamma_1]$).

In particular, x decides on 0 in $H[\gamma_0]$ after some event p_0 in S, and decides on 1 in $H[\gamma_1]$ after some event p_1 in S. Let L be the a prefix of S that contains both $Past(p_0, S)$ and $Past(p_1, S)$. By the end of the finite execution $H|_L[\gamma_0]$ (resp. $H|_L[\gamma_1]$) x decides on the value 0 (resp. 1).

We can extend $H|_L$ to get a weakly fair legal history H' of O on the graph G as follows. Consider any weakly fair extension S' of L on G. In L, the history H' outputs the same values as $H|_L$; and in the complement of L, the history H' outputs 0 everywhere except at x, where it outputs 1. Since $K \in Cones(O, G, \sigma, x)$, we have $H' \in O(G, \sigma, x)$, i.e., H' is a legal history of O on G.

For $v \in \{0, 1\}$, let g_v be the initial configuration on G such that g_v is equal to γ_v on D, and 1 elsewhere. In $H'[g_0]$, agent x decides by the end of L. The support of the causal past of the event preceding its decision, is included in D. Hence, since g_0 and γ_0 are equal on D, x decides on 0 in $H'[g_0]$. For similar reasons, x decides on 1 in $H'[g_1]$. Now pick an agent y in $G - D$, and let g be the initial configuration obtained from g_0 by permuting the values of a and y. In other words, $g(a) = g_0(y) = 1$, $g(y) = g_0(a) = \gamma_0(a) = 0$, and, for every $b \in G - \{a, y\}$, $g(b) = g_0(b)$. The restriction of g to D is equal to γ_1. Hence, in $H'[g]$, the agent x decides on the value 1. On the other hand, since the protocol solves the symmetric consensus, x decides on the value 0; whence a contradiction. □

A consequence of the previous lemma is that, if an oracle O is strong enough to solve symmetric consensus, then every legal history of O is a legal history of $DejaVu$. Then, roughly speaking, by taking the protocol that simply outputs the same information provided to it by O, we show that $DejaVu$ is weaker than O (according to the definitions in Sec. 2.5). The idea is formalized in the following theorem.

Theorem 1 (Weakest Oracle). *The $DejaVu$ oracle is the weakest oracle for solving symmetric consensus in population protocols.*

Proof. Consider an oracle O such that some protocol solves symmetric consensus using O. By Lemma 5, we have $Cones(O, G, \sigma, x) \subseteq Cones(DejaVu, G, \sigma, x)$ for every triple (G, σ, x). We claim that every legal history of O is a legal history of $DejaVu$.

Indeed, let S be some schedule, and $H \in O(G, \sigma, S)$. We first show that H satisfies the safety property (see Sec. 2.4) of $DejaVu$. If H outputs 1 at x during some event p, then, by the safety property of O, the causal past of p in S contains some schedule $K \in Cones(O, G, \sigma, x)$. Since $K \in Cones(DejaVu, G, \sigma, x)$ by Lemma 5, H also satisfies the safety property of $DejaVu$.

Second, we show that H satisfies the liveness property (see Sec. 2.4) of $DejaVu$. Precisely, we have to show that O eventually outputs 1 at *at least one* of the agents whose causal pasts contain some cones defining $DejaVu$. But since the underlying schedule S is weakly fair, every agent x eventually has some cone from $Cones(DejaVu, G, \sigma, x)$ in its causal past. Therefore, we only have to show that O eventually output 1 at *at least one* agent. But this is a consequence of Lemma 4. Therefore, we have proved that every legal history of O is a legal history of $DejaVu$.

We define a protocol $\mathcal{A}$ with a state space $\{0, 1\}$ and an input space $\{0, 1\}$ by the following rules

$$(q_1, v_1)(q_2, v_2) \to (v_1, v_2)$$

There is a unique initial state 0. Intuitively, during a transition, each agent simply copies its input (here provided by the oracle O) into its state. The output map $Out : \{0, 1\} \to \{0, 1\}$ is defined as the identity map.

Let $H[\gamma_0] = \gamma_0 \xrightarrow{E_0,h_0} \gamma_1 \xrightarrow{E_1,h_1} \dots$ be an execution of $\mathcal{A}$ with $H = (S, h)$ a legal history of O. Consider $H' = (S, h')$ be the output history of this execution (see Sec. 2.3). Then

$$\begin{aligned} \forall (x,y) \in E_0,\ h'(0,(x,y)) &= (Out(\gamma_0(x)), Out(\gamma_0(y))) \\ &= (0,0) \\ \forall i \geq 1, \forall (x,y) \in E_i,\ h'(i,(x,y)) &= (Out(\gamma_i(x)), Out(\gamma_i(y))) \\ &= (h(p_{i,x}), h(p_{i,y})) \end{aligned}$$

where $p_{i,x}$ (resp. $p_{i,y}$) is the latest event (different from $(i,(x,y))$) involving agent x (resp. agent y) in the causal past of the event $(i,(x,y))$.

We now prove that H' is a legal history of $DejaVu$. First, H' satisfies the safety property of $DejaVu$. Indeed, if H' outputs 1 at x during some event p, then this implies that the history H outputs 1 at x during some event p_{old} in the causal past of p. Since H is a legal history of $DejaVu$, this implies that the causal past of p_{old} (and thus of p) has a support equal to G. In other words, by event p, agent x has indirectly met with every other agent.

Second, H' satisfies the liveness property of $DejaVu$. Indeed, we have shown above that H eventually outputs 1 at at least one agent, say, x during some event p. Therefore, H' outputs 1 at x during the next event involving x (which eventually occurs since the schedule is weakly fair).

Thus, H' is a legal history of $DejaVu$, and we have proved that $DejaVu$ is the weakest oracle for solving symmetric consensus (see Sec. 2.5). □

6 Conclusion and Perspectives

Designing oracles and searching for the weakest among them in the model of population protocols is especially hard and challenging for several reasons. Anonymity is the first reason. Although oracles have already been studied in anonymous networks, this was done mostly assuming a point-to-point communication model. There, a process can distinguish between two messages arriving from two different communication links. In population protocols there are no links. A second reason is that, for population protocols, the size of the network is unknown, because the available memory for an agent is uniformly bounded. Being unaware of the total number of agents is an important restriction that leads to oracles completely different from those already existing in the literature. Moreover, we want to highlight that in this work we introduce oracles whose output depends only on the past interactions and that are protocol independent, in contrast with previously proposed oracles for population protocols. Thus, exhibiting oracles and the weakest between them, in this context, is especially challenging, because no known technique can be used.

We note that all the results of the paper can be extended to every family of graphs where the partitioning argument is valid (e.g., bounded degree graphs or trees). Moreover, all the results are easily extended also to other fairness conditions, e.g., to the classical, for population protocols, local and global fairness [2, 13].

Many difficult problems remain open though. For instance, we did not introduce crash failures, because, even without such failures, consensus is impossible for population protocols. Yet, is it possible to define a variant of $DejaVu$ to solve consensus with crash failures? Would this variant still be the weakest? The case of Byzantine failures seems even more problematic. On the other hand, we have focused on the symmetric version of consensus that better suits the model of population protocols. Still, one may want to search for other oracles allowing to solve non symmetric consensus, and look for the existence of a weakest oracle.

References

1. M. K. Aguilera, W. Chen, and S. Toueg. Using the heartbeat failure detector for quiescent reliable communication and consensus in partitionable networks. *Theor. Comput. Sci.*, 220(1):3–30, 1999.
2. 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.
3. H. Attiya and J. Welch. *Distributed Computing*. McGraw-Hill, 1998.
4. J. Beauquier, P. Blanchard, and J. Burman. Self-stabilizing leader election in population protocols over arbitrary communication graphs. In *OPODIS*, pages 38–52, 2013.
5. F. Bonnet and M. Raynal. The price of anonymity: Optimal consensus despite asynchrony, crash and anonymity. In *DISC*, pages 341–355, 2009.
6. F. Bonnet and M. Raynal. Anonymous asynchronous systems: The case of failure detectors. In *DISC*, pages 206–220, 2010.
7. Z. Bouzid and C. Travers. Anonymity, Failures, Detectors and Consensus. Technical report, 2012.
8. Z. Bouzid and C. Travers. Brief announcement: Anonymity, failures, detectors and consensus. In *DISC*, pages 427–428, 2012.
9. P. Cartier and D. Foata. Problèmes combinatoire de commutation et réarrangements. *Lecture Notes in Mathematics*, (85), 1969.
10. T. D. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems. *J. ACM*, 43(2):225–267, 1996.
11. J. Cortés, S. Martínez, T. Karatas, and F. Bullo. Coverage control for mobile sensing networks. *IEEE T. Robotics and Automation*, 20(2):243–255, 2004.
12. C. Delporte-Gallet, H. Fauconnier, R. Guerraoui, and A. Tielmann. The weakest failure detector for message passing set-agreement. In *DISC*, pages 109–120, 2008.
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. L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7):558–565, July 1978.
16. J. R. Lawton and R. W. Beard. Synchronized multiple spacecraft rotations. *Automatica*, 38(8):1359–1364, 2002.
17. N. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.
18. O. Michail and P. G. Spirakis. Terminating population protocols via some minimal global knowledge assumptions. *Journal of Parallel and Distributed Computing*, 81-82:1–10, 2015.
19. A. Mostéfaoui, S. Rajsbaum, M. Raynal, and C. Travers. The combined power of conditions and information on failures to solve asynchronous set agreement. *SIAM J. Comput.*, 38(4):1574–1601, 2008.
20. A. Mostéfaoui, S. Rajsbaum, M. Raynal, and C. Travers. On the computability power and the robustness of set agreement-oriented failure detector classes. *Distributed Computing*, 21(3):201–222, 2008.
21. R. Olfati-Saber, J. A. Fax, and R. M. Murray. Reply to "comments on "consensus and cooperation in networked multi-agent systems"". *Proceedings of the IEEE*, 98(7):1354–1355, 2010.
22. R. Olfati-Saber and J. S. Shamma. Consensus Filters for Sensor Networks and Distributed Sensor Fusion. *44th IEEE Conf. Decision and Control, 2005, and 2005 Eur. Control Conf. (CDC-ECC'05)*, pages 6698–6703, Dec. 2005.
23. R. Oshman. *Distributed Computation in Wireless and Dynamic Networks*. PhD thesis, MIT, Department of Electrical Engineering and Computer Science, 2012.
24. W. Ren, R. W. Beard, and E. M. Atkins. A survey of consensus problems in multi-agent coordination. In *American Control Conference*, pages 1859–1864, 2005.
25. G. Tel. *Introduction to Distributed Algorithms (2nd ed.)*. Cambridge University Press, 2000.

26. W. Xi, X. Tan, and J. S. Baras. A stochastic algorithm for self-organization of autonomous swarms. In *Proc. 44th IEEE Conf. Decision and Control, 2005 and 2005 Eur. Control Conf. (CDC-ECC'05)*, pages 765–770, 2005.