CORE
Metadata, citation and similar papers at core.ac.uk
Provided by Dagstuhl Research Online Publication Server

On Deterministic Linearizable Set Agreement Objects

Felipe de Azevedo Piovezan
Department of Computer Science, University of Toronto, Canada
felipe@cs.toronto.edu

Vassos Hadzilacos
Department of Computer Science, University of Toronto, Canada
vassos@cs.toronto.edu

Sam Toueg
Department of Computer Science, University of Toronto, Canada
sam@cs.toronto.edu

Abstract

A recent work showed that, for all n and k, there is a *linearizable* (n,k)-set agreement object O_L that is equivalent to the (n,k)-set agreement task [4]: given O_L, it is possible to solve the (n,k)-set agreement task, and given any algorithm that solves the (n,k)-set agreement task (and registers), it is possible to implement O_L. This linearizable object O_L, however, is not deterministic. It turns out that there is also a *deterministic* (n,k)-set agreement object O_D that is equivalent to the (n,k)-set agreement task, but this deterministic object O_D is not linearizable. This raises the question whether there exists a deterministic *and* linearizable (n,k)-set agreement object that is equivalent to the (n,k)-set agreement task. Here we show that in general the answer is no: specifically, we prove that for all $n \geq 4$, every deterministic linearizable $(n,2)$-set agreement object is strictly stronger than the $(n,2)$-set agreement task. We prove this by showing that, for all $n \geq 4$, every deterministic and linearizable $(n,2)$-set agreement object (together with registers) can be used to solve 2-consensus, whereas it is known that the $(n,2)$-set agreement task cannot do so. For a natural subset of $(n,2)$-set agreement objects, we prove that this result holds even for $n = 3$.

2012 ACM Subject Classification Theory of computation → Concurrency; Theory of computation → Parallel computing models; Theory of computation → Distributed computing models

Keywords and phrases Asynchronous shared-memory systems, consensus, set agreement, deterministic objects

Digital Object Identifier 10.4230/LIPIcs.OPODIS.2019.16

Funding This research was partially funded by the Natural Sciences and Engineering Research Council of Canada.

1 Introduction

Consensus is a fundamental problem of distributed computing and set agreement [5] is a well-known generalization of this problem. In the (n,k)-*set agreement task* each of n processes has an input value and must output one of the input values so that there are at most k distinct output values (the special case when $k = 1$ is the n-consensus task) [5, 7, 10]. Researchers have also considered (n,k)-set agreement *objects* [1, 6]. An (n,k)-*set agreement* (SA) *object* is an object that allows up to n processes to invoke a propose operation with some proposal value, such that the following two properties hold: (a) the value returned by the operation has been proposed, and (b) at most k different values are returned. Obviously, every (n,k)-SA object can be used to solve the (n,k)-SA task.

© Felipe de Azevedo Piovezan, Vassos Hadzilacos, and Sam Toueg;
licensed under Creative Commons License CC-BY

Until recently, it was thought that, for $k \geq 2$, (n,k)-SA objects were inherently *not* linearizable,[1] because every *linearizable* (n,k)-SA object imposes constraints that are not required by the (n,k)-SA task. For example, for $k \geq 2$, with any linearizable (n,k)-SA object, the propose operation that is linearized first must return its own proposal value; in contrast, with the (n,k)-set agreement task two processes with distinct inputs are allowed to each output the other's input (i.e., they can "swap" their inputs). In fact, such differences in allowable behaviours between tasks and objects motivated the introduction of generalizations of linearizability [9, 3]. Despite this, it was recently shown that there *does* exist a *linearizable* (n,k)-SA object O_L that is *equivalent to* the (n,k)-set agreement task, in the following sense: (a) given O_L, it is possible to solve the (n,k)-SA task, and (b) given any algorithm that solves the (n,k)-SA task (and registers), it is possible to implement O_L [4].

An object is *deterministic* if, when accessed sequentially, the response of each operation depends uniquely on the sequence of the preceding operations. For $k \geq 2$, the linearizable (n,k)-SA object O_L is *not* deterministic: in a sequential execution where the first operation proposes 1 and the second operation proposes 2, the second operation can return either 1 or 2.

Are (n,k)-SA objects inherently *not* deterministic? The answer is "no". We show that there exists a deterministic (n,k)-SA object O_D that is equivalent to the (n,k)-SA task (see Section 5). For $k \geq 2$, however, O_D is *not* linearizable.

Thus, the (n,k)-SA task has an equivalent linearizable (n,k)-SA object, *and* it also has an equivalent deterministic (n,k)-SA object. This raises the following question: is the (n,k)-SA task equivalent to some (n,k)-SA object that is *both* deterministic and linearizable?

In this paper we explore this question, and answer it in the negative. Specifically, we prove that for $n \geq 4$, no deterministic linearizable $(n,2)$-SA object is equivalent to the $(n,2)$-SA task: we do so by showing that, for $n \geq 4$, every deterministic linearizable $(n,2)$-SA object can be used to solve the 2-consensus task; in contrast, it is known that solving the $(n,2)$-SA task is not sufficient to solve 2-consensus [1]. This implies that, for $n \geq 4$, every deterministic linearizable $(n,2)$-SA object is strictly stronger than the $(n,2)$-SA task. For a natural subset of deterministic linearizable $(n,2)$-SA objects, we prove that this result holds even for $n = 3$.

2 Sketch of the model and basic definitions

In this paper, we consider distributed systems in which asynchronous processes communicate via shared objects. To limit the number of processes that can concurrently access an object, we consider shared objects with *ports*: to apply an operation on an object, a process chooses one of its ports, invokes the operation at that port, and waits for the response of the operation at that port before invoking another operation. No two operations are allowed to be applied simultaneously on the same port, otherwise the behavior of the object is arbitrary. So an object with n ports can be accessed concurrently by at most n processes. An execution of operations applied to an object is *well formed* if each operation invoked at a port of the object returns before another operation is invoked at that port. Note that a sequential execution of operations on an (n,k)-SA object, i.e., an execution where each operation completes before the next operation starts, is necessarily well formed.

[1] Intuitively, an object is *linearizable* if it behaves as if *all* operations, including concurrent ones, are applied sequentially: each operation appears to take effect instantaneously at some distinct point between its invocation and response [8].

▶ **Definition 1.** *An (n, k)**-set agreement object** is an n-ported object that allows processes to invoke* ProposeSA *operations, each with some proposal value $v \in I$ (where $|I| \geq n$), and the responses of the* ProposeSA *operations satisfy the following two properties in every well-formed execution:*

- Validity*: the value returned by each operation is either its proposal value or the proposal value of some previously invoked* ProposeSA *operation.*
- k-Agreement*: at most k different values are returned by the invocations of* ProposeSA *operations.*

For the rest of this section, O is an (n, k)-set agreement object. As we mentioned in the introduction, O is not necessarily linearizable, but of course it can be accessed sequentially. A sequential execution of operations on O can be modelled by a sequence of triples as follows:

▶ **Definition 2.** *A finite or infinite sequential execution (of operations) on O is a sequence of the form $(p_1, v_1, u_1)(p_2, v_2, u_2)\ldots$, where the i-th operation on O is a* ProposeSA(v_i) *at port p_i of O that returns u_i.*

▶ **Definition 3.** *The **sequential behaviour** of O is the set of* all *possible sequential executions of operations on O.*

Thus, the sequential behaviour of O is a set of sequences of the form $(p_1, v_1, u_1)(p_2, v_2, u_2)\ldots$.

▶ **Notation 4.**

- *If a sequence S is in the sequential behaviour of O, i.e., S is a sequential execution of operations on O, we write $S \in O$.*
- *If $S \in O$ and the* ProposeSA(v) *operation applied immediately after S on port p_i of O can return the value u, we write $S(p_i, v, u) \in O$.*
- *If $S \in O$ but the* ProposeSA(v) *operation applied immediately after S on port p_i of O can **not** return the value u, we write $S(p_i, v, \neq u) \in O$.*

Intuitively, object O is deterministic if for every finite sequential execution of operations S on O the following holds: if S is executed on O, and then a ProposeSA(v) operation is applied on port p of O, the response of that operation is uniquely determined by S, p, and v. More precisely:

▶ **Definition 5.** *O is **deterministic** if for every finite $S \in O$, every port p, and all values v, u, u', if $S(p, v, u) \in O$ and $S(p, v, u') \in O$, then $u = u'$.*

Intuitively, O is linearizable if each operation on O appears to take effect instantaneously between its invocation and its response (even in the presence of concurrent operations) [8].

▶ **Definition 6.** *O is **linearizable** if in every well-formed execution of operations E on O, each operation has a distinct linearization point, between its invocation and its response, such that E appears to be one of the sequential executions of operations $S \in O$, where the operations of S occur in the order of their linearization points in E.*[2]

Because of the above definition, if O is linearizable we can view any well-formed execution of operations on O as one of the sequences $S \in O$.

[2] For simplicity, this definition assumes that all operations of E are complete, i.e., each one has a response. If E contains incomplete operations, one is allowed to discard some incomplete operations and complete the remaining ones by returning some value, before linearizing the now "complete" E.

▶ **Notation 7.** *If an (n,k)-SA object is both deterministic and linearizable, we say that it is an (n,k)-**DLSA object**.*

In the *2-consensus task*, each of two processes has an input value and must output one of the inputs, such that both processes output the same value. Intuitively, we say that O can be used to solve 2-consensus, if two processes can solve the 2-consensus task using registers and copies of O initialized appropriately. More precisely:

▶ **Definition 8.** *We say that O **can be used to solve 2-consensus** if two processes can solve the 2-consensus task using copies of O (each of which may be initialized by applying some finite sequential execution of operations on O) and registers.*[3]

3 Oblivious $(3,2)$-DLSA objects can be used to solve 2-consensus

In this section, we focus on *oblivious* (n,k)-DLSA objects, that is, (n,k)-DLSA objects that behave the same regardless of the port to which operations are applied. We show that for $n \geq 3$, oblivious $(n,2)$-DLSA objects can be used to solve 2-consensus, and thus they are *not* equivalent to the $(n,2)$-SA task.

We first define oblivious (n,k)-DLSA objects more precisely as follows:

▶ **Definition 9.** *Let O be an (n,k)-DLSA object. O is **oblivious** if for all finite sequential executions of operations $S \in O$, ports p_i, p_j, and values u, v, the following holds: If $S(p_i,u,v) \in O$ then $S(p_j,u,v) \in O$.*

Thus, henceforth we omit ports from sequential executions of operations on oblivious (n,k)-SA objects, e.g., instead of $S = (p_1,v_1,u_1)(p_2,v_2,u_2)\ldots(p_k,v_k,u_k)$, we write $S = (v_1,u_1)(v_2,u_2)\ldots(v_k,u_k)$.

By the validity and the k-set agreement properties, it is clear that the first ProposeSA operation applied to an (n,k)-DLSA object must return its own proposal value and the second must return either its own proposal value (allowed only if $k \geq 2$) or the proposal value of the first. Thus,

▶ **Observation 10.** *Let O be an oblivious (n,k)-DLSA object. For all values a, a', b, b', if $(a,a')(b,b') \in O$ then $a = a'$ and $b' \in \{a,b\}$.*

Suppose an oblivious (n,k)-DLSA object O has a finite sequential execution of operations S such that, after S has been applied to O, a process proposing a and another process proposing b can determine the order in which their propose operations were executed. Then O can be used to solve 2-consensus as we now explain.

▶ **Definition 11.** *Let O be an oblivious (n,k)-DLSA object. Let a, b be two (not necessarily distinct) proposal values, and let $S \in O$. Let a', a'', b' be the unique values such that:*

$$\begin{cases} S(b,b')(a,a') \in O \\ S(a,a'') \in O \end{cases}$$

- *If $a' \neq a''$, we say "a **notices** b **after** S", denoted $a \xrightarrow{S} b$.*
- *If $a' = a''$, we say "a **does not notice** b **after** S", denoted $a \overset{S}{\not\rightarrow} b$.*

[3] Note that this solution is allowed to use registers, in addition to copies of O; allowing the use of registers in solutions and object implementations is standard.

*If S is the empty sequence, we simply say that a **notices** b, denoted $a \to b$, or a **does not notice** b, denoted $a \not\to b$.*

▶ **Observation 12.** *Let O be an oblivious (n,k)-DLSA object. By Observation 10 and Definition 11, we have that for all distinct proposal values a, b:*

$$a \to b \text{ if and only if } (b,b)(a,b) \in O$$
$$a \not\to b \text{ if and only if } (b,b)(a,a) \in O$$

▶ **Definition 13.** *Let O be an oblivious (n,k)-DLSA object. We say that O is **good** if there exist a finite sequential execution $S \in O$ and proposal values a, b such that $a \xrightarrow{S} b$ and $b \xrightarrow{S} a$.*

▶ **Lemma 14.** *Let O be an oblivious (n,k)-DLSA object. If O is good, then O can be used to solve 2-consensus.*

Proof. Let O be an oblivious (n,k)-DLSA object and assume it is good. Since O is good, there exists a finite sequential execution $S \in O$ and two values a, b such that $a \xrightarrow{S} b$ and $b \xrightarrow{S} a$. By the definition of $\xrightarrow{S}$, it follows that there exist a', a'', b', b'' such that $a' \neq a''$, $b' \neq b''$ and:

$$S(a,a')(b,b'') \in O \tag{1}$$
$$S(b,b')(a,a'') \in O \tag{2}$$

To solve 2-consensus, O is first initialized by applying S to it. Two processes, p and q, can now solve 2-consensus using this initialized O and two registers, R_p (written only by p) and R_q (written only by q), by executing the following algorithm. We assume that p and q use *different* ports when accessing O, so their access to O is guaranteed to be well-formed.

```
PROPOSE($v_p$) by process $p$
1  $R_p := v_p$
2  $ret$ := PROPOSESA($a$) on a port of $O$
3  if $ret = a'$
4        decide $R_p$
5  else // $ret = a''$
6        decide $R_q$
```

```
PROPOSE($v_q$) by process $q$
1  $R_q := v_q$
2  $ret$ := PROPOSESA($b$) on a port of $O$
3  if $ret = b'$
4        decide $R_q$
5  else // $ret = b''$
6        decide $R_p$
```

Since O is linearizable (and it is accessed in a well-formed manner), we can regard the PROPOSESA operations of p and q as atomic, and thus the PROPOSESA operations are totally ordered. From the algorithm and (1), (2) it is clear that each process decides the value it reads from the register it writes if it is the first to execute a PROPOSESA operation; otherwise, it decides the value it reads from the register written by the other process. Therefore agreement and validity hold provided that the process that reads the register written by the other process (in its Line 6) does so after the other process has written into that register (in its Line 1). To see that if process p reads register R_q, p does so after process q has written into R_q, note that process p reads R_q if and only if its PROPOSESA operation does not return a'. By (1) and (2), this means that q executed its PROPOSESA operation (in Line 2) before p executes its ProposeSA operation; and therefore q writes into R_q (in its Line 1) before p reads R_q (in its Line 6). A similar argument applies to q. ◀

We will now prove that oblivious $(3,2)$-DLSA objects that are *not* good have some other properties that can also be exploited to solve 2-consensus.

▶ **Lemma 15.** *Let O be an oblivious $(3,2)$-DLSA object. If O is not good, the following holds. For all distinct proposal values u, v, w, if $v \not\to u$ then $w \to u$.*

Proof. Let O be an oblivious $(3,2)$-DLSA object and assume it is not good. Suppose $v \not\to u$. By Observation 12, $(u,u)(v,v) \in O$. Furthermore, if w is proposed after that, the response of this operation cannot be w, by the 2-agreement property (see the left branch of the figure below). So $(u,u)(v,v)(w,\neq w) \in O$.

Assume for contradiction that $w \not\to u$. By Observation 12 $(u,u)(w,w) \in O$. Furthermore, if v is proposed after that, the response of this operation cannot be v, by the 2-agreement property (right branch of the figure below). So $(u,u)(w,w)(v,\neq v) \in O$.

$$\begin{tikzcd}
 & S=(u,u) \arrow[dl, dash] \arrow[dr, dash] & \\
(v,v) \arrow[d, dash] & & (w,w) \arrow[d, dash] \\
(w,\neq w) & & (v,\neq v)
\end{tikzcd}$$

Let $S = (u,u)$. Since $S(v,v) \in O$ and $S(w,w)(v,\neq v) \in O$, $v \xrightarrow{S} w$. Similarly, $w \xrightarrow{S} v$, and so O is good, a contradiction. ◀

▶ **Lemma 16.** *Let O be an oblivious $(3,2)$-DLSA object. If O is not good, the following holds. For all distinct proposal values u, v, w, if $v \not\to u$ then all of the relationships illustrated in the diagram below hold:*

$$\begin{tikzcd}
u \arrow[rr, bend left] \arrow[dr, "\vert" marking] & & v \arrow[ll, "\vert" marking] \arrow[dl, bend left] \\
 & w \arrow[ul, bend left] \arrow[ur, "\vert" marking] &
\end{tikzcd}$$

Proof. Let O be an oblivious $(3,2)$-DLSA object and assume it is not good. Suppose $v \not\to u$. We prove each of the arrows in turn:

- $w \to u$: Since $v \not\to u$, Lemma 15 implies $w \to u$.
- $u \not\to w$: Because O is not good and $w \to u$, it follows that $u \not\to w$.
- $v \to w$: Since $u \not\to w$, Lemma 15 implies $v \to w$.
- $w \not\to v$: Because O is not good and $v \to w$, it follows that $w \not\to v$.
- $u \to v$: Since $w \not\to v$, Lemma 15 implies $u \to v$. ◀

▶ **Lemma 17.** *Let O be an oblivious $(3,2)$-DLSA object. If O cannot be used to solve 2-consensus, the following holds. For all distinct proposal values u, v, w, if $u \to v$ then there exists $x \neq w$ such that:*

$$(v,v)(u,v)(w,x) \in O$$
$$(u,u)(v,v)(w,x) \in O$$

Proof. Let O be an oblivious $(3,2)$-DLSA object and assume it cannot be used to solve 2-consensus. So, by Lemma 14, O is not good.

Assume $u \to v$, thus, by Observation 12, $(v,v)(u,v) \in O$. By the definition of DLSA objects, there exists x such that:

$$(v,v)(u,v)(w,x) \in O \tag{3}$$

Since O is not good, $u \to v$ implies $v \not\to u$. By Observation 12, $(u,u)(v,v) \in O$, and so there exists x' such that:

$$(u,u)(v,v)(w,x') \in O \tag{4}$$

The 2-agreement property implies that $x' \in \{u,v\}$, thus $w \neq x'$.

Because $v \not\to u$, Lemma 16 implies $w \not\to v$. By Observation 12, $(v,v)(w,w) \in O$. Thus, by the 2-agreement property:

$$(v,v)(w,w)(u,\neq u) \in O \tag{5}$$

Our goal is to show that $x = x'$. Suppose, for contradiction, that $x \neq x'$.

Two processes, p and q, can solve 2-consensus using O and two registers, R_p (written only by p) and R_q (written only by q), by executing the following algorithm. We assume that p and q use *different* ports when accessing O, so their access to O is guaranteed to be well-formed.

```
PROPOSE($v_p$) by process p
1   $R_p := v_p$
2   $ret :=$ PROPOSESA($u$) on a port of $O$
3   if $ret = u$
4       decide $R_p$
5   else
6       decide $R_q$
```

```
PROPOSE($v_q$) by process q
1   $R_q := v_q$
2   PROPOSESA($v$) on a port of $O$
3   $ret :=$ PROPOSESA($w$) on a port of $O$
4   if $ret = x'$
5       decide $R_p$
6   else
7       decide $R_q$
```

To see why this algorithm is correct, first notice that the possible interleavings of the three ProposeSA operations by p and q are:

- v, u, w (process q, then p, then q)
- u, v, w (process p, then q, then q)
- v, w, u (process q, then q, then p)

The values returned by O in these three cases are given in (3), (4) and (5), respectively. We now argue that the algorithm is correct. Termination is obvious since the algorithm is wait free.

Validity trivially holds if a process decides the value in the register it writes. So consider the cases where each process decides the value contained in the register of the other.

- Process p decides R_q iff its ProposeSA(u) operation returns a value different from u, which only happens in (3) and (5). In both cases, q has previously proposed v, in which case q has previously written its proposal into R_q.
- Process q decides R_p iff its ProposeSA(w) operation returns x', which only happens in (4). In that case, p has previously proposed u, in which case p has previously written its proposal into R_p.

In both cases, validity holds.

Agreement also follows from (3), (4) and (5). If only one process decides, then agreement holds trivially. So suppose both processes decide.

- If p decides R_p, then its ProposeSA(u) operation returned u, which only happens in (4). So q receives x' as the response of its ProposeSA(w) operation, and thus q also decides R_p.

- If p decides R_q, then its ProposeSA(u) operation returned a value different from u, which only happens in (3) and (5). In those cases, q receives $x \neq x'$ (in (3)) or $w \neq x'$ (in (5)) as the response of its ProposeSA(w) operation. Either way, the value q receives is different from x', and thus q also decides R_q.

In both cases, agreement holds.

We have shown that O can be used to solve 2-consensus, a contradiction. Therefore, $x = x'$, as desired. ◀

▶ **Lemma 18.** *Let O be an oblivious $(3,2)$-DLSA object. If O cannot be used to solve 2-consensus, the following holds. For all distinct proposal values u, v, w, if $u \to v$ then $(v,v)(w,w)(u,v) \in O$.*

Proof. Let O be an oblivious $(3,2)$-DLSA object and assume it cannot be used to solve 2-consensus. So, by Lemma 14, O is not good.

Assume $u \to v$. By Lemma 16, $w \not\to v$, therefore $(v,v)(w,w) \in O$ (Observation 12). Also, there exists x' such that $(v,v)(w,w)(u,x') \in O$. By the 2-agreement property, $x' \in \{v,w\}$. If $x' = v$, we are done, so assume, for contradiction, that $x' = w$, i.e., $(v,v)(w,w)(u,w) \in O(*)$.

Since $u \to v$, by Lemma 17, there exists $x \neq w$ such that $(v,v)(u,v)(w,x) \in O$, in other words, $(v,v)(u,v)(w,\neq w) \in O$ (**). Let $S = (v,v)$; then (*) and (**) imply that $u \xrightarrow{S} w$ and $w \xrightarrow{S} u$. Thus O is good, a contradiction. So it must be that $x' = v$. ◀

We are now ready to prove the following theorem:

▶ **Theorem 19.** *Every oblivious $(3,2)$-DLSA object can be used to solve 2-consensus.*

Proof. Let O be an oblivious $(3,2)$-DLSA object and let a, b, c be three distinct proposal values of O. Suppose, for contradiction, that O cannot be used to solve 2-consensus. By Lemma 14, O is not good.

Because O is not good, either $a \not\to b$ or $b \not\to a$. Without loss of generality, assume $b \not\to a$. By Lemma 16, we have:

▶ **Observation 20.** *All of the relationships among a, b, and c illustrated in the diagram below hold:*

$$\begin{tikzcd}
a \arrow[rr, bend left] \arrow[dr, "\shortmid" marking] & & b \arrow[ll, "\shortmid" marking] \arrow[dl, bend left] \\
& c \arrow[ul, bend left] \arrow[ur, "\shortmid" marking] &
\end{tikzcd}$$

By Observation 20, $c \to a$. Applying Lemma 18 with $c = u$, $a = v$ and $b = w$, it follows that $(a,a)(b,b)(c,a) \in O$. By the 2-agreement property, if we then propose b, the return value of this operation cannot be c. Therefore:

$$(a,a)(b,b)(c,a)(b,\neq c) \in O \tag{6}$$

By Observation 20, $b \to c$. Applying Lemma 18 with $b = u$, $c = v$ and $a = w$, it follows that $(c,c)(a,a)(b,c) \in O$ (*). Furthermore, since $c \to a$, by Lemma 17 with $c = u$, $a = v$ and $b = w$, there exists $x \neq b$ such that:

$(a,a)(c,a)(b,x) \in O$
$(c,c)(a,a)(b,x) \in O$

By (*) and the fact that O is deterministic, $x = c$. Thus, $(a,a)(c,a)(b,c) \in O$. Now let y be such that

$$(a,a)(c,a)(b,c)(b,y) \in O$$

By the 2-agreement property, $y \in \{a,c\}$. If $y = a$, for $S = (a,a)(c,a)$, we have $S(b,c)(b,a) \in O$. Therefore $b \xrightarrow{S} b$, contradicting the fact that O is not good. Thus $y = c$ and

$$(a,a)(c,a)(b,c)(b,c) \in O \tag{7}$$

To solve 2-consensus, O is first initialized by applying $S = (a,a)$ to it. Two processes, p and q, can now solve 2-consensus using this initialized O and two registers, R_p (written only by p) and R_q (written only by q), by executing the following algorithm. We assume that p and q use *different* ports when accessing O, so their access to O is guaranteed to be well-formed.

```
PROPOSE($v_p$) by process p
1  $R_p := v_p$
2  $ret :=$ PROPOSESA($b$) on a port of $O$
3  **if** $ret = b$
4      decide $R_p$
5  **else**
6      decide $R_q$
```

```
PROPOSE($v_q$) by process q
1  $R_q := v_q$
2  PROPOSESA($c$) on a port of $O$
3  $ret :=$ PROPOSESA($b$) on a port of $O$
4  **if** $ret \neq c$
5      decide $R_p$
6  **else**
7      decide $R_q$
```

To see why this algorithm is correct, first notice that the possible interleavings of the three PROPOSESA operations by p and q are:

- b, c, b (process p, then q, then q)
- c, b, b (process q, then p, then q)
- c, b, b (process q, then q, then p)

Notice that the second and third interleavings have the same sequence of proposal values. The values returned by O in each of those cases is given in (6) and (7), under the assumption that the object is initialized by applying (a,a) to it. We now argue that the algorithm is correct; the arguments are similar to those of Lemma 17. Termination is obvious since the algorithm is wait free.

Validity trivially holds if a process decides the value in the register it writes. So consider the cases where each process decides the value contained in the register of the other.

- Process p decides R_q iff its PROPOSESA(b) operation returns a value different from b, which only happens in (7). In that case, q has previously proposed c, in which case q has previously written its proposal into R_q.
- Process q decides R_p iff its PROPOSESA(b) operation returns a value different from c, which only happens in (6). In that case, p has previously proposed b, in which case p has previously written its proposal into R_p.

In both cases, validity holds.

Agreement also follows from the properties of O's specification that we derived above. If only one process decides, then agreement holds trivially. So suppose both processes decide.

- If p decides R_p, then its ProposeSA(b) operation returned b, which only happens in (6). So q receives a value different from c as the response of its ProposeSA(b) operation, and thus q also decides R_p.
- If p decides R_q, then its ProposeSA(b) operation returned a value different from b, which only happens in (7). So q receives c as the response of its ProposeSA(b) operation, and thus q also decides R_q.

In both cases, agreement holds.

Under the assumption that O cannot be used to solve 2-consensus, we have shown that O can be used to solve 2-consensus, a contradiction. We conclude that O can indeed be used to solve 2-consensus. ◀

▶ **Corollary 21.** *For all $n \geq 3$, every oblivious $(n, 2)$-DLSA object can be used to solve 2-consensus.*

▶ **Theorem 22.** *For all $n \geq 3$, there is no oblivious $(n, 2)$-DLSA object that is equivalent to the $(n, 2)$-SA task.*

Proof. It is known that it is impossible to solve the 2-consensus task using an arbitrary solution to the $(n, 2)$-SA task (and registers) [1]. The theorem now follows immediately by Corollary 21. ◀

4 $(4, 2)$-DLSA objects can be used to solve 2-consensus

We now prove that for $n \geq 4$, $(n, 2)$-DLSA objects can be used to solve 2-consensus, which implies that they are not equivalent to the $(n, 2)$-SA task.

Recall that $(4, 2)$-DLSA objects accept at least 4 different values as proposal values, and have 4 ports to which ProposeSA operations may be applied. The proof that such objects can be used to solve 2-consensus is simpler than the oblivious $(3, 2)$-DLSA case.

By the validity and the k-set agreement properties, it is clear that the first ProposeSA operation applied to an (n, k)-DLSA object must return its own proposal value and the second must return either its own proposal value (allowed only if $k \geq 2$) or the proposal value of the first. Thus,

▶ **Observation 23.** *Let O be an (n, k)-DLSA object. For any two (not necessarily distinct) ports p_1, p_2 and for all values a, a', b, b', if $(p_1, a, a')(p_2, b, b') \in O$ then $a = a'$ and $b' \in \{a, b\}$.*

Suppose an (n, k)-DLSA object O has a finite sequential execution of operations S such that, after S has been applied to O, a process proposing a to some port p_i and another process proposing b to some port p_j can determine the order in which their propose operations were executed. Then O can be used to solve 2-consensus as we now explain.

▶ **Definition 24.** *Let O be an (n, k)-DLSA object. Let a, b be two (not necessarily distinct) proposal values, let p_i, p_j be two (not necessarily distinct) ports, and let $S \in O$. Let a', a'', b' be the unique values such that:*

$$\begin{cases} S(p_j, b, b')(p_i, a, a') \in O \\ S(p_i, a, a'') \in O \end{cases}$$

- *If $a' \neq a''$, we say that (p_i, a) **notices** (p_j, b) **after** S, denoted $(p_i, a) \xrightarrow{S} (p_j, b)$.*
- *If $a' = a''$, we say that (p_i, a) **does not notice** (p_j, b) **after** S, denoted $(p_i, a) \overset{S}{\not\rightarrow} (p_j, b)$.*

If S is the empty sequence, we simply say (p_i, a) ***notices*** *(p_j, b), denoted $(p_i, a) \rightarrow (p_j, b)$, or (p_i, a)* ***does not notice*** *(p_j, b), denoted $(p_i, a) \not\rightarrow (p_j, b)$.*

▶ **Observation 25.** *Let O be an (n, k)-DLSA object. By Observation 23 and Definition 24, we have that for all (not necessarily distinct) ports p_i, p_j and for all distinct proposal values a, b:*

$$(p_i, a) \rightarrow (p_j, b) \text{ if and only if } (p_j, b, b)(p_i, a, b) \in O$$
$$(p_i, a) \not\rightarrow (p_j, b) \text{ if and only if } (p_j, b, b)(p_i, a, a) \in O$$

▶ **Definition 26.** *Let O be an (n, k)-DLSA object. We say that O is* ***good*** *if there exist a finite sequential execution $S \in O$, two proposal values a, b, and two* distinct *ports p_i, p_j such that $(p_i, a) \xrightarrow{S} (p_j, b)$ and $(p_j, b) \xrightarrow{S} (p_i, a)$.*

▶ **Lemma 27.** *Let O be an (n, k)-DLSA object. If O is good, then O can be used to solve 2-consensus.*

Proof. Let O be a (n, k)-DLSA object and assume it is good. Since O is good, there exists a finite sequential execution $S \in O$ two values a, b and two distinct ports p_i, p_j such that $(p_i, a) \xrightarrow{S} (p_j, b)$ and $(p_j, b) \xrightarrow{S} (p_i, a)$.

By the definition of $\xrightarrow{S}$, it follows that there exist a', a'', b', b'' such that $a' \neq a''$, $b' \neq b''$ and:

$$S(p_i, a, a')(p_j, b, b'') \in O \tag{8}$$
$$S(p_j, b, b')(p_i, a, a'') \in O \tag{9}$$

To solve 2-consensus, O is first initialized by applying S to it. Two processes, p and q, can now solve 2-consensus using this initialized O and two registers, R_p (written only by p) and R_q (written only by q), by executing the following algorithm. Recall that p_i and p_j are distinct, so the two processes apply their PROPOSESA operations to different ports, ensuring that there are no concurrent operations applied to any port.

```
PROPOSE($v_p$) by process $p$
1  $R_p := v_p$
2  $ret :=$ PROPOSESA($a$) on port $p_i$ of $O$
3  if $ret = a'$
4        decide $R_p$
5  else // $ret = a''$
6        decide $R_q$
```

```
PROPOSE($v_q$) by process $q$
1  $R_q := v_q$
2  $ret :=$ PROPOSESA($b$) on port $p_j$ of $O$
3  if $ret = b'$
4        decide $R_q$
5  else // $ret = b''$
6        decide $R_p$
```

From the algorithm and (8), (9) it is clear that each process decides the value it reads from the register it writes if it is the first to execute a PROPOSESA operation; otherwise, it decides the value it reads from the register written by the other process. Therefore agreement and validity hold provided that the process that reads the register written by the other process (in its Line 6) does so after the other process has written into that register (in its Line 1). To see that if process p reads register R_q, p does so after process q has written into R_q, note that process p reads R_q if and only if its PROPOSESA operation does not return a'. By (8) and (9), this means that q executed its PROPOSESA operation (in Line 2) before p executes its ProposeSA operation; and therefore q writes into R_q (in its Line 1) before p reads R_q (in its Line 6). A similar argument applies to q. ◀

We will now prove that $(4,2)$-DLSA objects that are *not* good have some other properties that can also be exploited to solve 2-consensus.

▶ **Lemma 28.** *Let O be a $(4,2)$-DLSA object. If O is not good, the following holds. For all distinct proposal values u, v, w and all ports p_i, p_j, p_k such that $p_j \neq p_k$, if $(p_j, v) \not\to (p_i, u)$ then $(p_k, w) \to (p_i, u)$.*

Proof. Let O be a $(4,2)$-DLSA object and assume that it is not good. Suppose $(p_j, v) \not\to (p_i, u)$. By Observation 25, $(p_i, u, u)(p_j, v, v) \in O$. Furthermore, if w is proposed after that on port p_k, the response of this operation cannot be w, by the 2-agreement property (see the left branch of the figure below). So $(p_i, u, u)(p_j, v, v)(p_k, w, \neq w) \in O$.

Now assume, for contradiction, that $(p_k, w) \not\to (p_i, u)$. Then, by Observation 25, $(p_i, u, u)(p_k, w, w) \in O$. Furthermore, if v is proposed after that on port p_j, the response of this operation cannot be v, by the 2-agreement property (right branch of the figure below). So $(p_i, u, u)(p_k, w, w)(p_j, v, \neq v) \in O$.

$$\begin{tikzcd}
 & S = (p_i, u, u) \arrow[dl, dash] \arrow[dr, dash] & \\
(p_j, v, v) \arrow[d, dash] & & (p_k, w, w) \arrow[d, dash] \\
(p_k, w, \neq w) & & (p_j, v, \neq v)
\end{tikzcd}$$

Let $S = (p_i, u, u)$. Since $S(p_j, v, v) \in O$ and $S(p_k, w, w)(p_j, v, \neq v) \in O$, $(p_j, v) \xrightarrow{S} (p_k, w)$. Similarly, $(p_k, w) \xrightarrow{S} (p_j, v)$, and so O is good, a contradiction. ◀

We are now ready to prove the following theorem:

▶ **Theorem 29.** *Every $(4,2)$-DLSA object can be used to solve 2-consensus.*

Proof. Let O be a $(4,2)$-DLSA object, let p_1, p_2, p_3, p_4 be distinct ports and let a, b, c, d be distinct proposal values. Suppose, for contradiction, that O cannot be used to solve 2-consensus. By Lemma 27, O is not good.

Because O is not good, either $(p_2, b) \not\to (p_1, a)$ or $(p_1, a) \not\to (p_2, b)$. Without loss of generality, assume that $(p_2, b) \not\to (p_1, a)$ (*).

▶ **Claim 29.1.** *All of the relationships among (p_1, a), (p_2, b), and (p_3, c) illustrated in the diagram below hold:*

$$\begin{tikzcd}
(p_1, a) \arrow[rr, bend left] \arrow[dr, "\shortmid" marking] & & (p_2, b) \arrow[ll, "\shortmid" marking] \arrow[dl, bend left] \\
 & (p_3, c) \arrow[ul, bend left] \arrow[ur, "\shortmid" marking] &
\end{tikzcd}$$

Proof. We prove each of the relationships in turn:

- $(p_2, b) \not\to (p_1, a)$: this is (*).
- $(p_3, c) \to (p_1, a)$: since $(p_2, b) \not\to (p_1, a)$, Lemma 28 implies $(p_3, c) \to (p_1, a)$.
- $(p_1, a) \not\to (p_3, c)$: since O is not good and $(p_3, c) \to (p_1, a)$, we have $(p_1, a) \not\to (p_3, c)$.
- $(p_2, b) \to (p_3, c)$: since $(p_1, a) \not\to (p_3, c)$, Lemma 28 implies $(p_2, b) \to (p_3, c)$.

- $(p_3, c) \not\to (p_2, b)$: since O is not good and $(p_2, b) \to (p_3, c)$, we have $(p_3, c) \not\to (p_2, b)$.
- $(p_1, a) \to (p_2, b)$: since $(p_3, c) \not\to (p_2, b)$, Lemma 28 implies $(p_1, a) \to (p_2, b)$. ◀

Now consider what happens to port p_4 and value d:

$$\begin{tikzcd}
 & (p_4, d) \arrow[dl, bend right] \arrow[dr, bend left] & \\
(p_1, a) \arrow[ur] \arrow[rr, bend left] \arrow[dr, "|" marking] & & (p_2, b) \arrow[ul, "|" marking] \arrow[ll, "|" marking] \arrow[dl, bend left] \\
 & (p_3, c) \arrow[ul, bend left] \arrow[ur, "|" marking] &
\end{tikzcd}$$

- $(p_4, d) \to (p_2, b)$: by Claim 29.1, $(p_3, c) \not\to (p_2, b)$. Then, by Lemma 28, $(p_4, d) \to (p_2, b)$.
- $(p_2, b) \not\to (p_4, d)$: because $(p_4, d) \to (p_2, b)$ and because O is not good, it must be the case that $(p_2, b) \not\to (p_4, d)$.
- $(p_1, a) \to (p_4, d)$: because $(p_2, b) \not\to (p_4, d)$, by Lemma 28 it follows that $(p_1, a) \to (p_4, d)$.
- $(p_4, d) \to (p_1, a)$: by Claim 29.1, $(p_2, b) \not\to (p_1, a)$. Then, by Lemma 28, $(p_4, d) \to (p_1, a)$.

Since $(p_1, a) \to (p_4, d)$ and $(p_4, d) \to (p_1, a)$, O is good, a contradiction. Therefore O can be used to solve 2-consensus, as desired. ◀

▶ **Corollary 30.** *For all $n \geq 4$, every $(n, 2)$-DLSA object can be used to solve 2-consensus.*

▶ **Theorem 31.** *For all $n \geq 4$, there is no $(n, 2)$-DLSA object that is equivalent to the $(n, 2)$-SA task.*

Proof. It is known that it is impossible to solve the 2-consensus task using an arbitrary solution to the $(n, 2)$-SA task (and registers) [1]. The theorem now follows immediately by Corollary 30. ◀

5 Existence of a deterministic (n, k)-SA object equivalent to the (n, k)-SA task

In this section we prove that for all n and k, there is a deterministic (n, k)-set agreement object O_D that is equivalent to the (n, k)-set agreement task. This object is not linearizable, but its behaviour is deterministic when it is accessed sequentially (so concurrency is its only source of non-determinism).

Fix any (n, k)-SA object that is equivalent to the (n, k)-SA task. As we mentioned in the introduction, such objects exist [1, 6]; for concreteness we choose here the linearizable (n, k)-SA object O_L defined in [4]. Since O_L is equivalent to the (n, k)-SA task: (a) given O_L, it is possible to solve the (n, k)-SA task, and (b) given any algorithm that solves the (n, k)-SA task (and registers), it is possible to implement O_L. The simple algorithm of Figure 1 shows how to use O_L and a shared register R (initialized to $\perp$) to implement a *deterministic* (n, k)-set agreement object O_D:

It is easy to see that O_D behaves as follows:

- When accessed sequentially, O_D behaves deterministically: every PROPOSESA operation on O_D returns the proposal value of the *first* PROPOSESA operation on O_D. In other words, in sequential executions, O_D behaves like a consensus object.
- In all executions, O_D respects validity and k-agreement.

Thus, O_D is indeed a deterministic (n, k)-set agreement object.

```
PROPOSESA($v$) on port $p_i$ of $O_D$
1  $temp := R$
2  if $temp = \bot$
3      $temp :=$ PROPOSESA($v$) on port $p_i$ of $O_L$
4      $R := temp$
5  decide $temp$
```

Figure 1 Implementation of O_D using O_L and registers.

▶ **Theorem 32.** *For all $n \geq k \geq 1$, the deterministic (n, k)-SA object O_D is equivalent to the (n, k)-SA task.*

Proof. Clearly O_D can be used to solve the (n, k)-SA task. It remains to show that given any algorithm $\mathcal{A}_{n,k}$ that solves the (n, k)-SA task (and registers), it is possible to implement O_D. Since O_L is equivalent to the (n, k)-SA task, it can be implemented using $\mathcal{A}_{n,k}$ (and registers). Plugging this implementation of O_L in the algorithm of Figure 1, gives an implementation of O_D that uses $\mathcal{A}_{n,k}$ (and registers). ◀

Note that for $k \geq 2$, the deterministic object O_D is *not* linearizable, even though the object O_L is: a concurrent execution of two PROPOSESA operations on O_D with different proposal values may result in each operation returning its own proposal, but this behaviour is not possible in any *sequential* execution of these operations on O_D.

6 Remark on object initialization

To obtain our results, we gave several algorithms that solve 2-consensus using (a single copy of) some (n, k)-DLSA object O that was initialized by applying some sequence of operations. The reader may ask whether these results still apply if algorithms solving 2-consensus are required to use only *un*initialized copies of O. The answer is yes. This is because, by a result of [2], the ability to solve 2-consensus using an object does not depend on whether this object can be initialized to a specific state or not.

7 Conclusion and open problems

In this paper, we proved that for all $n \geq 4$ there is no deterministic *and* linearizable $(n, 2)$-set agreement object that is equivalent to the $(n, 2)$-set agreement task, and this is because any such object can be used to solve 2-consensus. We conjecture that for all $n > k > 2$ there is no deterministic linearizable (n, k)-set agreement object that is equivalent to the (n, k)-set agreement task, and this is because any such object can be used to solve some (n', k')-set agreement task that is strictly stronger than the (n, k)-set agreement task according to the partial order of set agreement tasks shown by Chaudhuri and Reiners [6]. The techniques we used in this paper to prove Theorems 19 and 29, however, do not seem appropriate to approach this conjecture: As n and k increase, the number of sequential behaviours for the first k accesses of an (n, k)-DLSA object increases exponentially with k, and this would be overwhelming for $k \geq 3$.

We also proved that there is no *oblivious* deterministic and linearizable $(3, 2)$-set agreement object that is equivalent to the $(3, 2)$-set agreement task; the case of (not necessarily oblivious) $(3, 2)$-set agreement objects is still open.

References

1 Elizabeth Borowsky and Eli Gafni. *The implication of the Borowsky-Gafni simulation on the set-consensus hierarchy*. CSD (Series). UCLA Computer Science Department, 1993. URL: https://books.google.ca/books?id=gpNeGwAACAAJ.

2 Elizabeth Borowsky, Eli Gafni, and Yehuda Afek. Consensus Power Makes (Some) Sense! (Extended Abstract). In *Proceedings of the Thirteenth Annual ACM Symposium on Principles of Distributed Computing*, PODC '94, pages 363–372, New York, NY, USA, 1994. ACM. doi:10.1145/197917.198126.

3 Armando Castañeda, Michel Raynal, and Sergio Rajsbaum. Specifying Concurrent Problems: Beyond Linearizability. *CoRR*, abs/1507.00073, 2015. arXiv:1507.00073.

4 David Yu Cheng Chan, Vassos Hadzilacos, and Sam Toueg. On the Number of Objects with Distinct Power and the Linearizability of Set Agreement Objects. In Andréa W. Richa, editor, *31st International Symposium on Distributed Computing (DISC 2017)*, volume 91 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 12:1–12:14, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.DISC.2017.12.

5 Soma Chaudhuri. More Choices Allow More Faults: Set Consensus Problems in Totally Asynchronous Systems. *Inf. Comput.*, 105(1):132–158, July 1993. doi:10.1006/inco.1993.1043.

6 Soma Chaudhuri and Paul Reiners. Understanding the Set Consensus Partial Order Using the Borowsky-Gafni Simulation (Extended Abstract). In *Proceedings of the 10th International Workshop on Distributed Algorithms*, WDAG '96, pages 362–379, London, UK, UK, 1996. Springer-Verlag. URL: http://dl.acm.org/citation.cfm?id=645953.675630.

7 Maurice Herlihy and Nir Shavit. The Topological Structure of Asynchronous Computability. *J. ACM*, 46(6):858–923, November 1999. doi:10.1145/331524.331529.

8 Maurice Herlihy and Jeannette Wing. Linearizability: A Correctness Condition for Concurrent Objects. *ACM Trans. Program. Lang. Syst.*, 12(3):463–492, July 1990. doi:10.1145/78969.78972.

9 Gil Neiger. Set-Linearizability. In *Proceedings of the Thirteenth Annual ACM Symposium on Principles of Distributed Computing*, PODC '94, pages 396–, New York, NY, USA, 1994. ACM. doi:10.1145/197917.198176.

10 Michael Saks and Fotios Zaharoglou. Wait-free K-set Agreement is Impossible: The Topology of Public Knowledge. In *Proceedings of the Twenty-fifth Annual ACM Symposium on Theory of Computing*, STOC '93, pages 101–110, New York, NY, USA, 1993. ACM. doi:10.1145/167088.167122.