Life Beyond Set Agreement

David Yu Cheng Chan
Department of Computer Science
University of Toronto
10 King's College Road
Toronto, Ontario M5S3G4, Canada
davidchan@cs.toronto.edu

Vassos Hadzilacos
Department of Computer Science
University of Toronto
10 King's College Road
Toronto, Ontario M5S3G4, Canada
vassos@cs.toronto.edu

Sam Toueg
Department of Computer Science
University of Toronto
10 King's College Road
Toronto, Ontario M5S3G4, Canada
sam@cs.toronto.edu

ABSTRACT

The *set agreement power* of a shared object O describes O's ability to solve set agreement problems: it is the sequence $(n_1, n_2, \ldots, n_k, \ldots)$ such that, for every $k \geq 1$, using O and registers one can solve the k-set agreement problem among at most n_k processes. It has been shown that the ability of an object O to implement other objects is not fully characterized by its *consensus number* (the first component of its set agreement power) [1, 3, 14]. This raises the following natural question: is the ability of an object O to implement other objects fully characterized by its *set agreement power*? We prove that the answer is no: every level $n \geq 2$ of Herlihy's consensus hierarchy has two objects that have the same set agreement power but are not equivalent, i.e., at least one cannot implement the other. We also show that every level $n \geq 2$ of the consensus hierarchy contains a *deterministic* object O_n with some set agreement power $(n_1, n_2, \ldots, n_k, \ldots)$ such that being able to solve the k-set agreement problems among n_k processes, for all $k \geq 1$, is not enough to implement O_n.

KEYWORDS

asynchronous system, shared memory, set agreement, consensus hierarchy

1 INTRODUCTION

Background and Motivation. The most widely studied problem in distributed computing is consensus [8] and its natural generalization, k-set agreement [5]. With the k-set agreement problem, each process has a proposal value and must decide on one of the proposed values such that there are at most k distinct decision values.

In this paper we consider distributed systems where asynchronous processes may apply operations to wait-free shared objects and fail by crashing [10]. The *k-set agreement number* of a shared object O is the largest integer n_k such that instances of O and registers can solve the k-set agreement problem among n_k processes, or it is ∞ if instances of O and registers can solve the k-set agreement problem among any number of processes. Since 1-set agreement is just *consensus*, the 1-set agreement number of an object is its *consensus number* [10]. The *set agreement power* of an object O is the infinite sequence $(n_1, n_2, \ldots, n_k, \ldots)$ where n_k is the k-set agreement number of O for all $k \geq 1$.[1] Thus $(n_1, n_2, \ldots, n_k, \ldots)$ describes the object O's ability to solve set agreement problems among various numbers of processes: for all $k \geq 1$, O can solve the k-set agreement problem among n processes if and only if $n \leq n_k$.

Herlihy showed that instances of any object with consensus number n, together with registers, can implement in a wait-free manner any object that can be shared by up to n processes [10]. The consensus number of an object, however, does not fully characterize its ability to implement other objects: there are objects O and O' that have the same consensus number but instances of O, together with registers, cannot implement O' in a wait-free manner [1, 3, 14]. Since the consensus number of an object is only the *first* component of its set agreement power, this raises the following natural question: does the set agreement power of an object fully characterize its ability to implement other objects? To formulate this question more precisely, we define two objects to be *equivalent* if each can be implemented from instances of the other and registers in a wait-free manner. The question then is: are two objects equivalent if and only if they have the same set agreement power?

Our Result. In this paper, we show that the answer is no. In fact, we prove the following stronger result: every level $n \geq 2$ of the well-known consensus hierarchy [10] contains a pair of objects that have the same set agreement power but are not equivalent.[2] We also show that every level $n \geq 2$ of the consensus hierarchy contains a *deterministic* object O_n with some set agreement power $(n_1, n_2, \ldots, n_k, \ldots)$ such that being able to solve the k-set agreement problems among n_k processes, for all $k \geq 1$, is not enough to implement O_n.

2 ROADMAP OF THE PROOF

In Section 3, we introduce the *n-pseudo-abortable consensus (n-PAC)* object, which is a non-abortable and deterministic version of the abortable n-DAC object introduced in [9].

In Section 4, we consider the *n-DAC problem* defined in [9], and prove that for $n \geq 2$ the $(n+1)$-DAC problem *can* be solved using $(n+1)$-PAC objects, but *cannot* be solved using n-consensus objects, registers, and a strong version of 2-set agreement objects,

This work was partially supported by the Natural Sciences and Engineering Research Council of Canada.

PODC'17, , July 25–27, 2017, Washington, DC, USA.

ACM ISBN 978-1-4503-4992-5/17/07…$15.00
http://dx.doi.org/10.1145/3087801.3087822

[1] In [7], k-set agreement number and set agreement power are called *k-set-consensus number* and *set consensus power*.
[2] Recall that an object O is at level n of the consensus hierarchy if instances of the object and registers can solve consensus among n, but not $n+1$, processes, i.e., the consensus number of O is n.

denoted 2-SA objects. This implies:

for $n \geq 2$, the $(n+1)$-PAC object *cannot* be implemented using n-consensus objects, registers, and 2-SA objects. (1)

In Section 5, we introduce a combination of an n-PAC object and an m-consensus object, called the (n, m)-PAC object, and show that for all $m \geq 2$, (n, m)-PAC is at level m of the consensus hierarchy. In particular, for all $n \geq 2$, the $(n+1, n)$-PAC object, denoted O_n, is at level n of this hierarchy. By definition O_n is at least as strong as the $(n+1)$-PAC object, therefore, by (1):

for $n \geq 2$, O_n *cannot* be implemented using n-consensus objects, registers, and 2-SA objects. (2)

Finally, in Section 6, for all $n \geq 2$ we construct an object O'_n such that: (a) O'_n has the same set agreement power as O_n, and (b) O'_n *can* be implemented by n-consensus objects, registers, and 2-SA objects. By (2) and (b):

O_n cannot be implemented by O'_n objects and registers. (3)

So every level $n \geq 2$ of the consensus hierarchy has two objects, namely, O_n and O'_n, that are not equivalent to each other, even though they have the same set agreement power.

Let $(n_1, n_2, \ldots, n_k, \ldots)$ be the set agreement power of O_n. By construction, the set agreement power of O'_n is $(n_1, n_2, \ldots, n_k, \ldots)$ as well. Thus, O'_n and registers can solve the k-set agreement problem among n_k processes, for all $k \geq 1$. By (3), this implies that the deterministic object O_n *cannot* be implemented from arbitrary solutions to the k-set agreement problems among n_k processes, for all $k \geq 1$, and registers.

3 PSEUDO-ABORTABLE CONSENSUS OBJECTS

In [9], Hadzilacos and Toueg introduced n-DAC objects that behave like n-consensus objects except for one key difference: operations can nondeterministically *abort* without taking effect as long as the following conditions are satisfied

- Every operation that aborts is concurrent with another operation.
- No crashed process may cause infinitely many aborts.

In this section, we define *n-pseudo-abortable consensus (n-PAC)* objects that behave similarly to n-DAC objects, but are deterministic and not abortable. Intuitively, for all $n \geq 1$, the n-PAC object simulates an n-DAC object by supporting two operations: PROPOSE(v, i) and DECIDE(i), where i is an integer in $[1..n]$ that we call the *label* of the operation. A process can use these two operations to simulate a PROPOSE(v) operation on an n-DAC object by first applying a PROPOSE(v, i) operation and then applying a DECIDE(i) operation with the same label i on the n-PAC object. Whenever a PROPOSE(v, i) operation is performed on an n-PAC object, the object simulates the invocation of a propose operation with value v on the i-th port of the simulated n-DAC object, and returns **done**.[3] Whenever a DECIDE(i) operation is performed, the n-PAC simulates the completion of the propose operation on the i-th port of the simulated n-DAC object, and returns the response.

[3]The n-DAC has n ports through which operations are applied; in contrast, our n-PAC object has no ports.

Thus the propose and decide operations form matching pairs, with each decide operation needing a previous propose operation with the same label. If a DECIDE(i) operation is performed without a matching PROPOSE$(-, i)$ operation performed beforehand, the n-PAC object becomes permanently upset. Similarly, if a pair of PROPOSE$(-, i)$ operations are performed without a DECIDE(i) in between, the n-PAC object becomes permanently upset. Intuitively, this is because two operations were invoked concurrently on port i of the n-DAC object.

A (sequential) *history* of an n-PAC object is a sequence of propose and decide operations applied to that object. We say that the history of an n-PAC object is *legal* if for all $i \in [1..n]$, the history of operations with label i performed on the object is either empty or begins with a propose operation and alternates between propose and decide operations. Thus an n-PAC object becomes upset if and only if its history is not legal. For convenience, we say a PROPOSE$(-, i)$ operation on the n-PAC object *decides* a value v' if it has a matching DECIDE(i) operation that returns v'. Once an n-PAC object becomes upset, it returns the special value $\perp$ to all decide operations but still returns **done** to all propose operations.[4]

To simulate the fact that an n-DAC object can abort when operations are concurrent, an n-PAC object that is not upset behaves as follows: if there is an operation between a propose operation and its matching decide operation, then the decide operation returns $\perp$. So intuitively, an n-PAC object returns $\perp$ if it is upset or if it detects a concurrent operation.

We now describe the n-PAC object more precisely. The state of such an object consists of:

- A boolean *upset*, initially **false**.
- An array $V[1..n]$, initially all NIL. Intuitively, $V[i] = v$ if the last operation with label i is a PROPOSE(v, i) operation.
- A variable L, initially NIL. Intuitively, $L = i$ if the last operation is PROPOSE$(-, i)$.
- A variable *val*, initially NIL. Intuitively, *val* contains the consensus value.

The sequential behaviour of the n-PAC object is given by Algorithm 1.

From the above description, it is clear that the sequential specification of n-PAC can be formally given in terms of a set of states, a set of operations, a set of responses, and a state transition relation. For brevity, we omit this formal definition here. The n-PAC object is *linearizable* [11], i.e., the operations PROPOSE$(-, -)$ and DECIDE$(-)$ are *atomic*. Thus, we consider only sequential histories of this object.

An n-PAC object *is upset* if its variable *upset* = **true**.

OBSERVATION 3.1. *If an n-PAC object is upset at some time, it remains upset thereafter.*

In the full paper we show the following properties of n-PAC objects.

LEMMA 3.2. *An n-PAC object is upset at time t if and only if the history of all operations that have been performed on the object by time t is not legal.*

[4]We assume that processes do not propose the special values $\perp$ and NIL.

Algorithm 1 Sequential specification of the n-PAC object

```
1:  procedure PROPOSE(v, i) { 1 ≤ i ≤ n }
2:      if V[i] ≠ NIL then upset ← true
3:      if upset = false then
4:          L ← i
5:          V[i] ← v
6:      return done

7:  procedure DECIDE(i) { 1 ≤ i ≤ n }
8:      if V[i] = NIL then upset ← true
9:      if upset = true then return ⊥
10:     if L ≠ i then
11:         temp ← ⊥
12:     else
13:         if val = NIL then val ← V[i]
14:         temp ← val
15:     L ← NIL
16:     V[i] ← NIL
17:     return temp
```

Lemma 3.3. *If an n-PAC object is not upset at time t, then for all $i \in [1..n]$,*

$$V[i] = \begin{cases} v & \text{if, at time } t\text{, the last operation with} \\ & \text{label } i \text{ is a } \textsc{propose}(v, i) \text{ operation.} \\ \textsc{nil} & \text{otherwise.} \end{cases}$$

Lemma 3.4. *If an n-PAC object is not upset at time t,*

$$L = \begin{cases} i & \text{if, at time } t\text{, the last operation is } \textsc{propose}(-, i). \\ \textsc{nil} & \text{otherwise.} \end{cases}$$

Theorem 3.5. *An n-PAC object satisfies the following properties:*

(a) *Agreement: If a decide operation returns a value $v \neq \perp$, and another decide operation returns a value $v' \neq \perp$, then $v = v'$.*
(b) *Validity: If a decide operation returns a value $v \neq \perp$, then a propose operation proposes v and decides v.*
(c) *Nontriviality: A decide operation op returns $\perp$ if and only if either (i) the n-PAC object is upset before op, or (ii) there is no operation before op, or the last operation before op is not a propose operation with the same label.*

4 THE N-DAC PROBLEM

In [9], Hadzilacos and Toueg introduced the n-DAC problem, where $n \geq 2$ processes start with binary inputs and are supposed to decide on a binary value; in addition, one of the n processes is a distinguished process p that can abort instead of deciding. An algorithm solves the n-DAC problem if every execution of the algorithm satisfies the following properties:

- **Agreement:** If a process q decides v and a process q' decides v', then $v = v'$.
- **Validity:** If a process decides v, then some process that does not abort has input v.
- **Termination:**
  (a) If the distinguished process p takes infinitely many steps, then p eventually decides or aborts.
  (b) If any process $q \neq p$ takes infinitely many steps solo (i.e., not interleaved with steps of other processes), then q eventually decides.
- **Nontriviality:** If p aborts, then some process $q \neq p$ took at least one step.

Algorithm 2 Solving the n-DAC problem using a single n-PAC $\mathcal{D}$

```
    Code executed by the distinguished process p:
1:  D.PROPOSE(v_p, p)
2:  temp ← D.DECIDE(p)
3:  if temp ≠ ⊥ then
4:      decide temp
5:  else abort

    Code executed by each process q ≠ p:
6:  while true do
7:      D.PROPOSE(v_q, q)
8:      temp ← D.DECIDE(q)
9:      if temp ≠ ⊥ then
10:         decide temp
11:         break
```

It is easy to solve the n-DAC problem with an n-PAC object: the processes numbered 1 to n execute Algorithm 2, where p is the distinguished process, and v_q is the initial value of process q ($1 \leq q \leq n$). The proof of the following theorem is straightforward and is given in the full paper.

Theorem 4.1. *For all $n \geq 2$, the n-DAC problem can be solved using a single n-PAC object.*

We now define the *strong 2-set agreement object*, denoted 2-*SA*, that is designed to solve the 2-set agreement problem among any finite number of processes. The state of the 2-SA object is a set STATE, initially empty. The 2-SA object supports the PROPOSE(v) operation, which adds v to STATE if STATE contains fewer than 2 elements, and returns a value arbitrarily selected from STATE. Consequently, a 2-SA object responds to operations with at most two distinct values, corresponding to the *first* two distinct values proposed to the object.[5]

Algorithm 3 gives the sequential behaviour of the 2-SA object.

Algorithm 3 Sequential specification of the 2-SA object

```
1:  procedure PROPOSE(v)
2:      if |STATE| < 2 then STATE ← STATE ∪ {v}
3:      return arbitrary value from STATE
```

From the above, it is clear that the sequential specification of 2-SA can be formally given in terms of a set of states, a set of operations, a set of responses, and a state transition relation. The 2-SA object is linearizable, i.e., its PROPOSE(−) operation is *atomic*.

[5] This object is "strong" because the 2-set agreement problem does not have this restriction: with the 2-set agreement problem, processes can decide *any* two of the proposed values.

It is straightforward to see that the 2-SA object solves the 2-set agreement problem among any finite number of processes, as required. A fortiori, the 2-SA object solves the k-set agreement problem among n processes for all $k \geq 2$ and all $n \geq 1$.

In [9], it was shown that for all $n \geq 2$, the $(n+1)$-DAC problem cannot be solved using n-consensus objects and registers. We strengthen this result by showing the following:

Theorem 4.2. *For all $n \geq 2$, the $(n+1)$-DAC problem cannot be solved using n-consensus objects, registers, and 2-SA objects.*[6]

Proof. In the following, we use the bivalency technique introduced by Fischer et al. [8], and we assume the reader is familiar with the associated terminology. In particular, a configuration C is *v-valent* (for $v \in \{0,1\}$) if starting from C no process can decide $\overline{v} = 1 - v$; C is *univalent* if it is either 0-valent or 1-valent; and C is *bivalent* if it is not univalent [8, 10].

Assume, for contradiction, that there exists an algorithm $\mathcal{A}$ that solves the $(n+1)$-DAC problem among $n+1$ processes using only n-consensus objects, registers, and 2-SA objects. Let I be the initial configuration where the input of the distinguished process p is 1 and the input of every other process is 0.

The following standard claim still holds for the $(n+1)$-DAC problem:

Claim 4.2.1. *No configuration reachable from an I is both 0-valent and 1-valent.*

Proof. Suppose a configuration C reachable from an I is both 0-valent and 1-valent. There are $n+1 > 1$ processes, so some process q is not the distinguished process p. By Termination (b) of the $(n+1)$-DAC problem, there exists a finite q-solo history H such that H is applicable to C and q decides some value $v \in \{0,1\}$ in $H(C)$. Then, since q decides v in $H(C)$, $H(C)$ is not $\overline{v}$-valent, where $\overline{v} = 1 - v$. Thus C is also not $\overline{v}$-valent, contradicting our assumption that C is both 0-valent and 1-valent. □ Claim 4.2.1

Claim 4.2.2. *If p aborts in a configuration C that is reachable from initial configuration I, then C is 0-valent.*

Proof. Suppose, for contradiction, that p aborts in a configuration C that is reachable from I, but C is not 0-valent. Then, there is a finite history H, applicable to C, such that some process decides 1 in $H(C)$. Since p aborts in C, p also aborts in $H(C)$, since aborting is irrevocable. Since C is reachable from I, so is $H(C)$. This violates the Validity property of the $(n+1)$-DAC problem, since p is the only process with input 1 in I. □ Claim 4.2.2

The above claim immediately implies:

Observation 4.2.3. *If p aborts or decides in a configuration C that is reachable from initial configuration I, then C is univalent.*

Claim 4.2.4. *I is bivalent.*

Proof. Starting from I, let p run solo until it either decides or aborts, which eventually happens by Termination (a). By Nontriviality, since p is the only process that takes steps, p does not abort. Furthermore, since p is the only process that takes steps, p does not know the input of any other process. Thus by Validity, p decides its own input 1, so I is not 0-valent.

Starting from I, let a process $q \neq p$ run solo until it decides, which eventually happens by Termination (b). Since q is the only process that takes steps, q does not know the input of any other process. Thus by Validity, q decides its own input 0, so I is not 1-valent.

Since I is neither 0-valent nor 1-valent, I is bivalent. □ Claim 4.2.4

Claim 4.2.5. *There is a bivalent configuration C' that is reachable from I such that for every history H that is applicable to C' and ends with a step of the distinguished process p, $H(C')$ is univalent.*

Proof. Suppose, for contradiction, that the claim is false. In other words, for every bivalent configuration C' that is reachable from I, there is a history H such that H is applicable to C', H ends with a step of p, and $H(C')$ is bivalent. Since $H(C')$ is itself a bivalent configuration that is reachable from I, a straightforward induction can be used to construct an infinite history that contains infinitely many steps of p but never reaches a univalent configuration. Thus from Observation 4.2.3, p takes infinitely many steps without deciding or aborting, violating Termination (a). □ Claim 4.2.5

Claim 4.2.6. *There is a configuration C reachable from I, a process $q \neq p$, and steps e_p, e'_p of p and e_q of q, such that e_p and $e_q e'_p$ are applicable to C, $e_p(C)$ is v-valent and $e_q e'_p(C)$ is $\overline{v}$-valent for some $v \in \{0,1\}$.*

Proof. Let C' be a configuration as in Claim 4.2.5. Let f_p be a step of p that is applicable to C'. By Claim 4.2.5, $f_p(C')$ is univalent. Let $v \in \{0,1\}$ be such that $f_p(C')$ is v-valent.

Subclaim 4.2.6.1. *There is a finite p-free history H and a step f'_p of p such that Hf'_p is applicable to C', and $Hf'_p(C')$ is $\overline{v}$-valent.*

Proof. Since C' is bivalent, there is a configuration C'' that is reachable from C' such that some process decides $\overline{v}$ in C'', and so C'' is $\overline{v}$-valent. Let H' be a history such that $C'' = H'(C')$. There are two cases:

Case 1. H' is p-free.

Then let $H = H'$, and f'_p be a step of p applicable to $C'' = H(C')$. Since C'' is $\overline{v}$-valent, so is $f'_p(C'') = Hf'_p(C')$. Thus H is a finite p-free history and f'_p is a step of p such that Hf'_p is applicable to C', and $Hf'_p(C')$ is $\overline{v}$-valent as wanted.

Case 2. H' is not p-free.

Then let H be the longest p-free prefix of H', and let f'_p be the step of p in H' immediately after H. Thus Hf'_p is a prefix of H'. Since f'_p is a step of p and Hf'_p is applicable to C', by Claim 4.2.5, $Hf'_p(C')$ is univalent. Then, since Hf'_p is a prefix of H' and $H'(C') = C''$, C'' is reachable from $Hf'_p(C')$. Finally, since C'' and $Hf'_p(C')$ are both univalent, and C'' is $\overline{v}$-valent and reachable from $Hf'_p(C')$, $Hf'_p(C')$ is also $\overline{v}$-valent.

□ Subclaim 4.2.6.1

Subclaim 4.2.6.2. *The history H of Subclaim 4.2.6.1 is not empty.*

[6] Recall that an n-consensus object solves consensus among n but not $n+1$ processes. A precise specification of the n-consensus object as a deterministic linearizable object, for $n \geq 2$, is given in [12, 13]: for the first n propose operations, the n-consensus object returns the value of the first propose operation, and it returns a special value $\perp$ to any subsequent propose operation.

Proof. Suppose, for contradiction, that H is empty. Thus there are two steps of p, f_p and f'_p, such that $f_p(C')$ is v-valent and $f'_p(C')$ is $\overline{v}$-valent. Since processes are deterministic, f_p and f'_p are steps by p on the same object O. Since $f_p(C')$ and $f'_p(C')$ have opposite valence, O must be nondeterministic. Since the algorithm uses only n-consensus objects, registers, and 2-SA objects, and both n-consensus objects and registers are deterministic, O is a 2-SA object. Thus the steps f_p and f'_p of p are propose operations on the 2-SA object O. Furthermore, since p is deterministic, the value proposed by p in f_p is the same as in f'_p. From the sequential specification of the 2-SA object (Algorithm 3), the state of the 2-SA object only records values that are proposed to it, not values that it returns. Thus the state of O is the same in $f_p(C')$ as in $f'_p(C')$. Consequently, $f_p(C')$ and $f'_p(C')$ differ only in the state of p.

Now, let q be a process other than p. By Termination (b), there exists a finite q-solo history $\hat{H}$ such that $\hat{H}$ is applicable to $f_p(C')$ and q decides in $\hat{H}$. Since $f_p(C')$ and $f'_p(C')$ differ only in the state of p, $\hat{H}$ is also applicable to $f'_p(C')$. Thus q decides the same value in $f_p\hat{H}(C')$ as in $f'_p\hat{H}(C')$, contradicting the fact that $f_p(C')$ and $f'_p(C')$ are univalent configurations with opposite valence. □ Subclaim 4.2.6.2

Let H be a history as in Subclaim 4.2.6.1. By Subclaim 4.2.6.2, it is a non-empty history $H = h_1h_2\ldots h_k$, where $h_1, h_2, \ldots, h_k$ are steps of processes that are not p. Let $C_0, C_1, \ldots, C_k$ be the configurations such that $C_0 = C'$ and $C_i = h_i(C_{i-1})$ for all i, $1 \leq i \leq k$. For each i, $0 \leq i \leq k$, we now define a step $h_{p,i}$ of p applicable to C_i. Let $h_{p,0} = f_p$, $h_{p,k} = f'_p$, and for $0 < i < k$, let $h_{p,i}$ be any step of p that is applicable to C_i.

By Claim 4.2.5, $h_{p,i}(C_i)$ is univalent for all i, $0 \leq i \leq k$. Since $h_{p,0}(C_0) = f_p(C')$, $h_{p,0}(C_0)$ is v-valent. Then, since $h_{p,k}(C_k) = Hf'_p(C')$, $h_{p,k}(C_k)$ is $\overline{v}$-valent. Thus, there is some i, $0 \leq i < k$, such that $h_{p,i}(C_i)$ is v-valent and $h_{p,i+1}(C_{i+1})$ is $\overline{v}$-valent. Let $C = C_i$, and let $q \neq p$ be the process that executes step h_{i+1}, $e_q = h_{i+1}$, $e_p = h_{p,i}$, and $e'_p = h_{p,i+1}$. Then we have that e_p and $e_qe'_p$ are applicable to C, $e_p(C)$ is v-valent, and $e_qe'_p(C)$ is $\overline{v}$-valent, as wanted. □ Claim 4.2.6

Let configuration C, process q, steps e_p, e'_p, and e_q, and $v \in \{0,1\}$ be as in Claim 4.2.6. Thus,

$$e_p(C) \text{ is } v\text{-valent, while } e_qe'_p(C) \text{ is } \overline{v}\text{-valent.} \quad (4)$$

Since processes are deterministic, e_p and e'_p access the same object X and e_q accesses some object Y.

Claim 4.2.7. *$X = Y$.*

Proof. Suppose, for contradiction, that $X \neq Y$. Then the step e_q of q that is applicable to C is also applicable to $e_p(C)$. There are two cases:

Case 1. $e_p = e'_p$.
Then e_p and e_q commute at C; i.e., e_pe_q and e_qe_p are both applicable to C, and $e_pe_q(C) = e_qe_p(C) = e_qe'_p(C)$. Since $e_pe_q(C)$ has the same valence as $e_p(C)$, this contradicts (4).

Case 2. $e_p \neq e'_p$.
Since $e_p \neq e'_p$, X is a nondeterministic object, i.e., a 2-SA object. Furthermore, since processes are deterministic, the value proposed by p to the 2-SA object X is the same in e_p as in e'_p. Then, since e_q is an operation on $Y \neq X$, the state of X is the same in $e_pe_q(C)$ as in $e_qe'_p(C)$. Consequently, $e_pe_q(C)$ and $e_qe'_p(C)$ differ only in the state of p.
By Termination (b), there exists a finite q-solo history $\hat{H}$ such that $\hat{H}$ is applicable to $e_pe_q(C)$ and q decides in $e_pe_q\hat{H}(C)$. Since $e_pe_q(C)$ and $e_qe'_p(C)$ differ only in the state of p, $\hat{H}$ is also applicable to $e_qe'_p(C)$. Thus q decides the same value in $e_pe_q\hat{H}(C)$ as in $e_qe'_p\hat{H}(C)$ contradicting (4). □ Claim 4.2.7

Claim 4.2.8. *X is not a register.*

Proof. Suppose, for contradiction, that X is a register. There are two cases to consider:

Case 1. e_p is a read step.
Since processes are deterministic, e'_p is also a read step. Since e_p is a read of X, the state of register X is the same in C as in $e_p(C)$. Thus, since e_q is applicable to C, it is also applicable to $e_p(C)$. Thus the configurations $e_pe_q(C)$ and $e_qe'_p(C)$ are identical except possibly in the state of p. By Termination (b), there is a finite q-solo history H such that H is applicable to $e_pe_q(C)$ and q decides in $e_pe_qH(C)$.
Since $e_pe_q(C)$ and $e_qe'_p(C)$ are identical except possibly in the state of p, and H contains no steps of p, H is also applicable to $e_qe'_p(C)$. Therefore q decides the same value in $e_pe_qH(C)$ as in $e_qe'_pH(C)$. This contradicts (4).

Case 2. e_p is a write step.
Since processes are deterministic, $e'_p = e_p$. Thus both e_p and e'_p write the same value into register X, and so configurations $e_p(C)$ and $e_qe'_p(C)$ are identical, except possibly in the state of q. By Termination (a), there is a finite p-solo history H such that p aborts or decides some value $u \in \{0,1\}$ in $e_pH(C)$.
Since $e_p(C)$ and $e_qe'_p(C)$ are identical, except possibly in the state of q, and H contains no steps of q, H is also applicable to $e_qe'_p(C)$. Therefore p aborts in both $e_pH(C)$ and $e_qe'_pH(C)$, or p decides the same value $u \in \{0,1\}$ in both $e_pH(C)$ and $e_qe'_pH(C)$. If p aborts, then, by Claim 4.2.2, both $e_pH(C)$ and $e_qe'_pH(C)$ are 0-valent; if p decides $u \in \{0,1\}$, then both $e_pH(C)$ and $e_qe'_pH(C)$ are u-valent. This contradicts (4).
□ Claim 4.2.8

Claim 4.2.9. *X is not an n-consensus object.*

Proof. Suppose, for contradiction, that X is an n-consensus object. Let e'_q be any step of q that is applicable to $e_p(C)$. From (4), the configuration $C_v = e_pe'_q(C)$ and $C_{\overline{v}} = e_qe'_p(C)$ have valence v and $\overline{v}$, respectively. Note that the consensus object X is the only object whose state in C_v may differ from its state in $C_{\overline{v}}$. Furthermore, since C_v and $C_{\overline{v}}$ are configurations reached from C via steps of p and q on X, each process that is not p or q has the same state in C_v as in $C_{\overline{v}}$.

Let q' be a process that is not p or q; q' exists, since there are $n+1 > 2$ processes. Let R be the set of $n-2$ processes that are not p, q or q' (if $n = 2$ the set R is empty). We construct a history H that is applicable to C_v as follows. Starting from C_v, let each process $r \in R$ (in some arbitrary order), run solo until *it is about to*

perform an operation on X: this must happen because $r \neq p$, so by Termination(b), when r runs solo it must eventually decide some value, but r cannot distinguish between the univalent configurations C_v and $C_{\overline{v}}$ with opposite valence without accessing X. Note that in history H, no process performs an operation on X, which is the only object whose state in C_v may differ from its state in $C_{\overline{v}}$. Thus H is also applicable to $C_{\overline{v}}$, each process $r \in R$ has the same state in $H(C_v)$ as in $H(C_{\overline{v}})$, and X is the only object whose state in $H(C_v)$ may differ from its state in $H(C_{\overline{v}})$.

Starting from $H(C_v)$, let each process $r \in R$ take one step (in some arbitrary order), and let C'_v be the resulting configuration. Similarly, starting from $H(C_{\overline{v}})$, let each process $r \in R$ take one step (in some arbitrary order), and let $C'_{\overline{v}}$ be the resulting configuration. Note that from the construction of H, every process $r \in R$ is about to perform an operation on X at $H(C_v)$ and $H(C_{\overline{v}})$. Thus X is the only object whose state in C'_v may differ from its state in $C'_{\overline{v}}$. Furthermore, in both C'_v and $C'_{\overline{v}}$, at least n operations have been performed on X (because every process, except possibly q', has performed at least one operation on X). Since X is an n-consensus object, after n operations have been performed on it, X is no longer useful in differentiating between configurations C'_v or $C'_{\overline{v}}$: more precisely, any operation that is applied to X after reaching configurations C'_v or $C'_{\overline{v}}$ returns the special value $\perp$. Note also that C'_v and $C'_{\overline{v}}$ are configurations with opposite valence.

Starting from C'_v, let q' take steps solo until it decides some value $u \in \{0, 1\}$ (since $q' \neq p$, q' must decide by Termination(b)), and let $H_{q'}$ be this solo history of q'. Since: (a) X is the only object whose state may differ between C'_v and $C'_{\overline{v}}$, and (b) after reaching configurations C'_v or $C'_{\overline{v}}$, the n-consensus object X returns $\perp$ to every operation, the solo history $H_{q'}$ of q' is also applicable to $C'_{\overline{v}}$. Thus q decides u in both $H_{q'}(C'_v)$ and $H_{q'}(C'_{\overline{v}})$, contradicting the fact that C'_v and $C'_{\overline{v}}$ have opposite valence. □ Claim 4.2.9

Claim 4.2.10. *X is not a 2-SA object.*

Proof. Suppose, for contradiction, that X is a 2-SA object. Thus p and q are both invoking propose operations on X. Let v_p and v_q be their respective proposal values. Since processes are deterministic, e'_p also proposes v_p.

Subclaim 4.2.10.1. *There exists a value $\hat{v}$ such that $\hat{v}$ is in X.state in both $e_p(C)$ and $e_q e'_p(C)$.*

Proof. If X.state is not empty at C, we are done, since values are never removed from X.state. Otherwise, X.state contains v_p in both $e_p(C)$ and $e_q e'_p(C)$. □ Subclaim 4.2.10.1

Now, let q' be a process other than p and q; q' exists, since there are $n + 1 > 2$ processes. Let $\hat{v}$ be a value as in Subclaim 4.2.10.1, and let $\hat{H}$ be the finite history where, starting from $e_p(C)$, q' runs solo while X responds to all operations with $\hat{v}$, until q' decides, which eventually happens by Termination (b). Since the state of q' is the same in $e_p(C)$ as in $e_q e'_p(C)$, and $\hat{v}$ is in X.state in $e_q e'_p(C)$, this q'-solo history $\hat{H}$ is also applicable to $e_q e'_p(C)$. Thus q' decides the same value in both $e_p\hat{H}(C)$ and $e_q e'_p\hat{H}(C)$. This contradicts (4). □ Claim 4.2.10

By Claims 4.2.8, 4.2.9, and 4.2.10, X is not an n-consensus object, register, or 2-SA object, contradicting our assumption that Algorithm $\mathcal{A}$ uses only such objects. □ Theorem 4.2

Theorems 4.1 and 4.2 imply:

Theorem 4.3. *For all $n \geq 2$, $(n + 1)$-PAC objects cannot be implemented using n-consensus objects, registers, and 2-SA objects.*

5 (N, M)-PAC IS AT LEVEL M OF THE CONSENSUS HIERARCHY

In this section, we first define a "boosted" version of the n-PAC object: for all $n \geq 1$ and $m \geq 1$, the (n, m)-PAC object is a combination of an n-PAC object P and an m-consensus object C. The (n, m)-PAC object supports three operations:

- proposeC(v), which redirects the operation propose(v) to C and returns the response.
- proposeP(v, i), which redirects the operation propose(v, i) to P and returns the response.
- decideP(i), which redirects the operation decide(i) to P and returns the response.

Note that for all $n \geq 1$ and $m \geq 1$, (n, m)-PAC objects are deterministic, since both n-PAC objects and m-consensus objects are deterministic.

Observation 5.1. *For all $n \geq 1$ and $m \geq 1$,*

(a) *An (n, m)-PAC object can be implemented using an n-PAC object and an m-consensus object.*
(b) *An (n, m)-PAC object can implement an n-PAC object.*
(c) *An (n, m)-PAC object can implement an m-consensus object.*

We now show that for $m \geq 2$, (n, m)-PAC objects are at level m of the consensus hierarchy.

Theorem 5.2. *For all $n \geq 1$ and $m \geq 2$, (n, m)-PAC objects and registers cannot be used to solve the $(m + 1)$-consensus problem.*

Proof. By Observation 5.1(a), the (n, m)-PAC object can be implemented using an n-PAC object and an m-consensus object. Thus it suffices to show that, for all $n \geq 1$ and $m \geq 2$, the $(m + 1)$-consensus problem cannot be solved using only n-PAC objects, m-consensus objects, and registers.

As in the proof of Theorem 4.2, we use the bivalency technique and terminology introduced in [8]. Assume, for contradiction, that there exists a wait-free algorithm $\mathcal{A}$ that solves binary consensus among $m + 1$ processes using only m-consensus objects, registers, and n-PAC objects. We omit the proofs of Claims 5.2.1 to 5.2.5 since they are standard.

Claim 5.2.1. *The algorithm has an initial bivalent configuration C_{init}.*

Claim 5.2.2. *There is a bivalent configuration C_{bi}, reachable from C_{init}, such that if any process takes a step, the resulting configuration is univalent.*

Claim 5.2.3. *At C_{bi}, all $m + 1$ processes are about to perform an operation on the same object $\mathcal{D}$.*

Claim 5.2.4. *$\mathcal{D}$ is not a register.*

Claim 5.2.5. *$\mathcal{D}$ is not an m-consensus object.*

It remains to show that $\mathcal{D}$ is not an n-PAC object. To do so, we use the following two claims.

Claim 5.2.6. *There does not exist a pair of reachable univalent configurations C and C' with opposite valence such that (i) there is a process that has the same state in C as in C', (ii) there is an n-PAC object D that is upset in both C and C', and (iii) every object other than D has the same state in C as in C'.*

Proof. Suppose, for contradiction, that there exists such a pair of configurations C and C'. Let p denote the process with the same state in C as in C', and let D denote the n-PAC object that is upset in both C and C'. Since C and C' have opposite valence, process p must be undecided in both C and C'. Consider a run R starting from C where p takes steps solo until it decides (this must eventually happen by the Termination property of the consensus problem). Since D is upset in C, all decide operations by p on D in R return $\perp$. Furthermore, all propose operations by p return **done**. Since p has the same state in C as in C', and D is upset in both C and C', it is clear that there is an identical p-solo run R' that starts from C'. Thus p decides the same value in both R and R', contradicting that C and C' have opposite valence. □ Claim 5.2.6

Claim 5.2.7. *There does not exist a pair of reachable univalent configurations C and C' with opposite valence such that (i) there are two processes that have the same state in C as in C', and (ii) with the possible exception of a single n-PAC object D, every object has the same state in C as in C'.*

Proof. Suppose, for contradiction, that such a pair exists. Let q_0 and q_1 denote the two processes that have the same state in C as in C'. Since C and C' have opposite valence, processes q_0 and q_1 must be undecided in both C and C'. Let D denote the n-PAC object whose state may be different in C than in C'.

Construct a run R starting from C as follows:

(1) First let q_0 take steps solo until it is about to perform a DECIDE(i) operation on D for some label i.
This must eventually happen, since D is the only object whose state in C is not the same as its state in C', and all PROPOSE($-,-$) operations on D return the value **done** regardless of the state of D.
(2) Then let q_1 take steps solo until it is about to perform a DECIDE(i') operation on D for some label i'.
By the same argument, this must eventually happen.

Since q_0 and q_1 have the same state in C as in C', and propose operations on D always return **done**, it is clear that there is an identical run R' that starts from C'.

Let C_{clash} and C'_{clash} denote the configurations resulting from R and R' respectively. Since R is identical to R', (i) q_0 and q_1 have the same state in C_{clash} as in C'_{clash}, and (ii) all objects except possibly D have states that are the same in C_{clash} as in C'_{clash}. Note that q_0 and q_1 are about to perform a DECIDE(i) operation and a DECIDE(i') operation on D, respectively, in both C_{clash} and C'_{clash}. Furthermore, since they began from C and C' respectively, C_{clash} and C'_{clash} are univalent configurations with opposite valence. There are two cases:

Case 1. $i = i'$.

Consider the following configurations:

- The configuration $\overline{C}$ reached from C_{clash} by letting first q_0 and then q_1 perform their DECIDE(i) operations.
- The configuration $\overline{C}'$ reached from C'_{clash} by letting first q_0 and then q_1 perform their DECIDE(i) operations.

Since C_{clash} and C'_{clash} are univalent configurations with opposite valence, $\overline{C}$ and $\overline{C}'$ are univalent configurations with opposite valence. However, since two DECIDE(i) operations were performed consecutively on the n-PAC object D, D is upset in both $\overline{C}$ and $\overline{C}'$. Furthermore, when going from C_{clash} to $\overline{C}$ and also when going from C'_{clash} to $\overline{C}'$, q_1 received $\perp$ for its DECIDE(i) operation because either this decide operation upsets D or D was already upset. Consequently, q_1 has the same state in $\overline{C}$ as in $\overline{C}'$. In other words, $\overline{C}$ and $\overline{C}'$ are a pair of reachable univalent configurations with opposite valence such that (i) there is a process, namely q_1, that has the same state in $\overline{C}$ as in $\overline{C}'$, (ii) there is an n-PAC object, namely $\mathcal{D}$, that is upset in both $\overline{C}$ and $\overline{C}'$, and (iii) every object other than $\mathcal{D}$ has the same state in $\overline{C}$ as in $\overline{C}'$. This contradicts Claim 5.2.6.

Case 2. $i \neq i'$.

From the sequential specification of an n-PAC object D (given in Algorithm 1), it is clear that a DECIDE(i) operation on D returns $\perp$ if either D is upset, or i is not equal to the value of the variable $D.L$ of the state of D.

Consider the state of D in configuration C_{clash}. If D is upset, then both q_0 and q_1 can perform their decide operations on D and receive $\perp$. Now suppose that D is not upset in C_{clash}, and consider the value of $D.L$ in C_{clash}. Since $i \neq i'$, either $D.L \neq i$ or $D.L \neq i'$. Without loss of generality, suppose that $D.L \neq i$. Let q_0 perform DECIDE(i). Since $D.L \neq i$, q_0 receives the reply $\perp$ from D. Moreover, from the sequential specification of D shown in Algorithm 1, q_0's decide operation either causes D to become upset, or it sets $D.L$ to NIL. Now let q_1 perform DECIDE(i') on D. Since either D is upset or $D.L$ = NIL $\neq i'$, q_1 also receives the reply $\perp$ from D.

From the above, we conclude that starting from configuration C_{clash}, there is an order in which q_0 and q_1 can each perform their decide operation such that they both receive $\perp$ from D. Let $\overline{C}$ be the resulting configuration.

By a symmetric argument, starting from C'_{clash}, there is an order in which q_0 and q_1 can each perform their decide operation such that they both receive $\perp$ from D. Let $\overline{C}'$ be the resulting configuration.

Since C_{clash} and C'_{clash} are univalent configurations with opposite valence, $\overline{C}$ and $\overline{C}'$ are univalent configurations with opposite valence. Furthermore, since q_0 and q_1 have the same state in C_{clash} as in C'_{clash}, and both q_0 and q_1 receive the same response in going from C_{clash} to $\overline{C}$ as in going from C'_{clash} to $\overline{C}'$, they also have the same state in $\overline{C}$ as in $\overline{C}'$. In other words, $\overline{C}$ and $\overline{C}'$ are a pair of reachable univalent configurations with opposite valence such that (i) there are two processes, namely q_0 and q_1, that have the same state in $\overline{C}$ as in $\overline{C}'$, and (ii) with the possible exception of a single

n-PAC object, namely $\mathcal{D}$, every object has the same state in $\overline{C}$ as in $\overline{C}'$.
Let $C_1 = C$ and $C'_1 = C'$. Since we have proven that Case 1 never occurs, a straightforward induction on Case 2 allows us to construct for all $j > 1$ a pair of reachable univalent configurations C_j and C'_j with opposite valence such that (i) there are two processes, namely q_0 and q_1, that have the same state in C_j as in C'_j (so they are both undecided in C_j and in C'_j), (ii) with the possible exception of a single n-PAC object, namely $\mathcal{D}$, every object has the same state in C_j as in C'_j, and (iii) C_j and C'_j are reached from C_{j-1} and C'_{j-1}, respectively, via runs in which both q_0 and q_1 take at least one step. Thus, this induction constructs runs in which both q_0 and q_1 take infinitely many steps without ever deciding — a violation of the Termination property of consensus. □ Claim 5.2.7

Claim 5.2.8. *$\mathcal{D}$ is not an n-PAC object.*

Proof. Suppose, for contradiction, $\mathcal{D}$ is an n-PAC object. Recall that by Claim 5.2.3, at C_{bi}, all $m+1$ processes are about to perform an operation on $\mathcal{D}$.

Subclaim 5.2.8.1. *At C_{bi}, every process is about to perform a decide operation on $\mathcal{D}$.*

Proof. Suppose, for contradiction, that some process p_0 is about to perform a propose operation on $\mathcal{D}$ at C_{bi}. Without loss of generality, suppose the resulting configuration C_0 is 0-valent. By Claim 5.2.2, there must exist another process p_1 such that if p_1 takes a step at C_{bi}, the resulting configuration C_1 is 1-valent. Consider the 1-valent configuration C'_1 reached by having p_0 take a step at C_1. Since propose operations always return **done**, p_0 has the same state in C_0 as in C'_1. Furthermore, since there are at least $m+1 > 2$ processes, there exists a process p_2 that is neither p_0 nor p_1, and since p_2 has not taken any steps since C_{bi}, p_2 has the same state in C_0 as in C'_1. Thus C_0 and C'_1 are a pair of reachable univalent configurations with opposite valence such that: (i) there are two processes, namely p_0 and p_2, that have the same state in C_0 as in C'_1, and (ii) with the possible exception of a single n-PAC object, namely $\mathcal{D}$, every object has the same state in C_0 as in C'_1. This contradicts Claim 5.2.7. □ Subclaim 5.2.8.1

Subclaim 5.2.8.2. *If a process performs a decide operation on $\mathcal{D}$ at C_{bi}, the reply of $\mathcal{D}$ is not $\perp$.*

Proof. Suppose, for contradiction, that a process p_0 performs a decide operation on $\mathcal{D}$ at C_{bi} and receives $\perp$ from $\mathcal{D}$. Let C_0 be the resulting configuration. Without loss of generality, suppose that C_0 is 0-valent. By Claim 5.2.2, there exists another process p_1 such that if p_1 takes a step at C_{bi}, the resulting configuration C_1 is 1-valent. By Subclaim 5.2.8.1, this step of p_1 is a decide operation on $\mathcal{D}$, so either it sets $\mathcal{D}.L \leftarrow$ nil, or $\mathcal{D}$ is upset in C_1. Consider the 1-valent configuration C'_1 reached by having p_0 perform its decide operation at C_1. Since either $\mathcal{D}.L =$ nil, or $\mathcal{D}$ is upset in C_1, process p_0 receives $\perp$ for this decide operation. Thus p_0 has the same state in C_0 as in C'_1. The rest of the proof is now as in Subclaim 5.2.8.1. □ Subclaim 5.2.8.2

Recall that a decide(i) operation on $\mathcal{D}$ returns $\perp$ if either $\mathcal{D}$ is upset, or i is not equal to the value of the variable $\mathcal{D}.L$ of the state of $\mathcal{D}$. Consequently, from Subclaim 5.2.8.1 and Subclaim 5.2.8.2, every process is about to perform a decide(i) operation at C_{bi}, such that $i = \mathcal{D}.L$ in C_{bi}. So, from Claim 5.2.2, there exist two processes p_0 and p_1 such that if p_0 performs decide(i) at C_{bi}, the resulting configuration C_0 is 0-valent, and if p_1 performs decide(i) at C_{bi}, the resulting configuration C_1 is 1-valent. Consider the following configurations:

- The 0-valent configuration C'_0 reached from C_{bi} by first letting p_0 and then p_1 perform their decide(i) operations.
- The 1-valent configuration C'_1 reached from C_{bi} by first letting p_1 and then p_0 perform their decide(i) operations.

Since two consecutive decide operations with the same label i were performed, $\mathcal{D}$ is upset in both C'_0 and C'_1. Furthermore, since there are $m+1 > 2$ processes, there exists a process p_2 that is neither p_0 nor p_1, and since p_2 has not taken any steps since C_{bi}, p_2 has the same state in C'_0 as in C'_1. Thus C'_0 and C'_1 are a pair of reachable univalent configurations with opposite valence such that (i) there is a process, namely p_2, that has the same state in C'_0 as in C'_1, (ii) there is an n-PAC object, namely $\mathcal{D}$, that is upset in both C'_0 and C'_1, and (iii) every object other than $\mathcal{D}$ has the same state in C'_0 as in C'_1. This contradicts Claim 5.2.6. □ Claim 5.2.8

By Claim 5.2.4, Claim 5.2.5, and Claim 5.2.8, $\mathcal{D}$ is neither an m-consensus object, nor an n-PAC object, nor a register. This contradicts that algorithm $\mathcal{A}$ uses only such objects. □ Theorem 5.2

By Observation 5.1(c) and Theorem 5.2, we have:

Theorem 5.3. *For all $n \geq 1$, $m \geq 2$, (n, m)-PAC objects are at level m of the consensus hierarchy.*

6 SET AGREEMENT POWER DOES NOT DETERMINE OBJECT EQUIVALENCE

We now prove the main result of this paper. This proof relies on (n, k)-*SA objects*, which allow up to n processes to solve the k-set agreement problem [2, 6]. With an (n, k)-SA object, each of n processes can apply a propose(v) operation and receive a value that satisfies the (n, k)-set agreement problem requirements.

Definition 6.1. For each $n \geq 2$, let O_n be the $(n+1, n)$-PAC object.

We now define an object O'_n that "embodies" the set agreement power of O_n. Let $n \geq 2$, and $(n_1, n_2, \ldots, n_k, \ldots)$ be the set agreement power of O_n. Let $\mathcal{C}_n$ be the collection of (n_k, k)-SA objects for all $k \geq 1$, i.e., $\mathcal{C}_n = \bigcup_{k=1}^{\infty} \{(n_k, k)\text{-SA}\}$. The object O'_n combines all the set agreement objects in $\mathcal{C}_n$ as follows: for all $k \geq 1$, O'_n supports the propose(v, k) operation, which redirects the operation propose(v) to the (n_k, k)-SA object of $\mathcal{C}_n$, and returns that object's response. Note that, by construction and the definition of set agreement power, O'_n has the same set agreement power as O_n, namely, $(n_1, n_2, \ldots, n_k, \ldots)$.

By Theorem 5.3, we have:

Observation 6.2. *For each $n \geq 2$, O_n has consensus number n.*

From Theorem 4.3 and Observation 5.1(b), we also have:

Observation 6.3. *For each $n \geq 2$, O_n cannot be implemented by n-consensus objects, registers, and 2-SA objects.*

Lemma 6.4. *For each $n \geq 2$, O'_n can be implemented by n-consensus objects and 2-SA objects.*

Proof. Let $n \geq 2$ and $(n_1, n_2, \ldots, n_k, \ldots)$ be the set agreement power of O_n. Recall that O'_n can be implemented by the objects in $C_n = \bigcup_{k=1}^{\infty}\{(n_k, k)\text{-SA}\}$. By Observation 6.2, O_n has consensus number n, so $n_1 = n$. Thus the $(n_1, 1)$-SA object in C_n can be implemented by an n-consensus object. In addition, for all $k \geq 2$, the (n_k, k)-SA object in C_n can be implemented by the 2-SA object. So every object in the collection $C_n = \bigcup_{k=1}^{\infty}\{(n_k, k)\text{-SA}\}$ can be implemented by an n-consensus object or a 2-SA object. Thus O'_n can be implemented by n-consensus objects and 2-SA objects. □ Lemma 6.4

Theorem 6.5. *For all $n \geq 2$, O_n cannot be implemented by O'_n objects and registers.*

Proof. Let $n \geq 2$. Suppose, for contradiction, that O_n can be implemented by O'_n objects and registers. So, by Lemma 6.4, O_n can be implemented by n-consensus objects, 2-SA objects, and registers — contradicting Observation 6.3. □ Theorem 6.5

By Observation 6.2, O_n is at level n of the consensus hierarchy. Since O'_n has the same set agreement power as O_n, it is also at level n of the consensus hierarchy. So by Theorem 6.5, we have:

Corollary 6.6. *In every level $n \geq 2$ of the consensus hierarchy, there is a pair of objects, namely O_n and O'_n, that have the same set agreement power but are not equivalent.*

Corollary 6.7. *In every level $n \geq 2$ of the consensus hierarchy, there is a* deterministic *object, namely O_n, that has set agreement power $(n_1, n_2, \ldots, n_k, \ldots)$ but can*not *be implemented from arbitrary solutions to the k-set agreement problems among n_k processes, for all $k \geq 1$, and registers.*

Proof. Let $n \geq 2$ and consider the objects O_n and O'_n. Let $(n_1, n_2, \ldots, n_k, \ldots)$ be the set agreement power of O_n, and therefore of O'_n as well. So O'_n objects and registers can solve the k-set agreement problem among n_k processes for all $k \geq 1$. Suppose, for contradiction, that O_n can be implemented from arbitrary solutions of the k-set agreement problems among n_k processes, for all $k \geq 1$, and registers. Thus O_n can be implemented from O'_n objects and registers — contradicting Theorem 6.5. □ Corollary 6.7

7 A RELATED RESULT

Qadri posed the following question in [13]: can $(m + 1)$-consensus objects and registers implement every deterministic object at level m of the consensus hierarchy? Using PAC objects, we show that the answer is no. In fact, we show the following more general result:

Theorem 7.1. *For all $m \geq 2$ and $n \geq m+1$, level m of the consensus hierarchy contains a deterministic object, namely the $(n + 1, m)$-PAC object, that cannot be implemented using n-consensus objects and registers.*

Proof. Let $m \geq 2$ and $n \geq m + 1$. Consider the deterministic object $(n + 1, m)$-PAC, which by Theorem 5.3, is at level m of the consensus hierarchy. We claim that the $(n+1, m)$-PAC object cannot be implemented using n-consensus objects and registers. Suppose, for contradiction, that the $(n+1, m)$-PAC object can be implemented using n-consensus objects and registers. By Observation 5.1(b), the $(n + 1, m)$-PAC object can solve the $(n + 1)$-DAC problem, thus the $(n + 1)$-DAC problem can be solved using n-consensus objects and registers — contradicting Theorem 4.2. □ Theorem 7.1

8 CONCLUSION

This paper proves that the set agreement power of an object does not fully characterize its ability to implement other objects: every level $n \geq 2$ of the consensus hierarchy has objects that have the same set agreement power but are not equivalent (Corollary 6.6). The paper also shows that every level $n \geq 2$ of the consensus hierarchy has a *deterministic* object, namely O_n, with set agreement power $(n_1, n_2, \ldots, n_k, \ldots)$ such that being able to solve the k-set agreement problems among n_k processes, for all $k \geq 1$, is not enough to implement O_n (Corollary 6.7). This result seems to refute a conjecture mentioned in [7], namely that the "computational power" of a deterministic object can be captured by its set agreement power.

We say that an object O is *equivalent to its set agreement power* if O is equivalent to the collection of objects $C = \bigcup_{k=1}^{\infty}\{(n_k, k)\text{-SA}\}$, where $(n_1, n_2, \ldots, n_k, \ldots)$ is the set agreement power of O. Corollary 6.6 implies that there are objects that are *not* equivalent to their set agreement power. Consider now the *subset* $\mathcal{R}$ of objects that *are* equivalent to their set agreement power. In other work [4], we use a result of [7] to prove that the restriction of Herlihy's hierarchy to $\mathcal{R}$ is *robust* in the following sense [12]: in $\mathcal{R}$, for every positive integer n, any set of objects with consensus number at most n cannot be used to implement any object with consensus number $n' > n$. In fact, we prove the following a more general result: roughly speaking, for any integer $\ell \geq 1$, if we classify objects in $\mathcal{R}$ based on the first ℓ components of their set agreement power sequence (i.e., if we place in the same class all the objects whose set agreement powers are identical for the first ℓ components), we obtain a classification that is also robust.

In [4] we also prove that the universe $\mathcal{U}$ of all shared-memory objects contains *uncountably* many objects with distinct computational power: the equivalence relation between objects (defined in Section 1) partitions $\mathcal{U}$ into uncountably many equivalence classes. Thus, the computational power of objects cannot be fully described by a countable scheme such as a finite vector of integers.

ACKNOWLEDGMENTS

We thank the anonymous referees for their comments and suggestions. This work was partially supported by the Natural Sciences and Engineering Research Council of Canada.

REFERENCES

[1] Yehuda Afek, Faith Ellen, and Eli Gafni. 2016. Deterministic Objects: Life Beyond Consensus. In *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing (PODC '16)*. ACM, New York, NY, USA, 97–106. https://doi.org/10.1145/2933057.2933116

[2] Elizabeth Borowsky and Eli Gafni. 1993. *The Implication of the Borowsky-Gafni Simulation on the Set-Consensus Hierarchy.* Technical Report 930021. UCLA Computer Science Dept.

[3] David Yu Cheng Chan, Vassos Hadzilacos, and Sam Toueg. 2017. Bounded Disagreement. In *20th International Conference on Principles of Distributed Systems (OPODIS 2016) (Leibniz International Proceedings in Informatics (LIPIcs))*, Panagiota Fatourou, Ernesto Jiménez, and Fernando Pedone (Eds.), Vol. 70. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, Article 5, 16 pages. https://doi.org/10.4230/LIPIcs.OPODIS.2016.5

[4] David Yu Cheng Chan, Vassos Hadzilacos, and Sam Toueg. 2017. On the Number of Objects with Distinct Power and the Linearizability of Set Agreement Objects. Under submission.

[5] Soma Chaudhuri. 1993. More Choices Allow More Faults: Set Consensus Problems in Totally Asynchronous Systems. *Information and Computation* 105, 1 (1993), 132 – 158. https://doi.org/10.1006/inco.1993.1043

[6] Soma Chaudhuri and Paul Reiners. 1996. 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)*. Springer-Verlag, London, UK, UK, 362–379. http://dl.acm.org/citation.cfm?id=645953.675630

[7] Carole Delporte-Gallet, Hugues Fauconnier, Eli Gafni, and Petr Kuznetsov. 2017. Set-Consensus Collections are Decidable. In *20th International Conference on Principles of Distributed Systems (OPODIS 2016) (Leibniz International Proceedings in Informatics (LIPIcs))*, Panagiota Fatourou, Ernesto Jiménez, and Fernando Pedone (Eds.), Vol. 70. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 7:1–7:15. https://doi.org/10.4230/LIPIcs.OPODIS.2016.7

[8] Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. 1985. Impossibility of Distributed Consensus with One Faulty Process. *J. ACM* 32, 2 (Apr 1985), 374–382. https://doi.org/10.1145/3149.214121

[9] Vassos Hadzilacos and Sam Toueg. 2013. On Deterministic Abortable Objects. In *Proceedings of the 2013 ACM Symposium on Principles of Distributed Computing (PODC '13)*. ACM, New York, NY, USA, 4–12. https://doi.org/10.1145/2484239.2484241

[10] Maurice Herlihy. 1991. Wait-free Synchronization. *ACM Trans. Program. Lang. Syst.* 11, 1 (Jan 1991), 124–149. https://doi.org/10.1145/114005.102808

[11] Maurice P. Herlihy and Jeannette M. Wing. 1990. Linearizability: A Correctness Condition for Concurrent Objects. *ACM Trans. Program. Lang. Syst.* 12, 3 (jul 1990), 463–492. https://doi.org/10.1145/78969.78972

[12] Prasad Jayanti. 1993. On the Robustness of Herlihy's Hierarchy. In *Proceedings of the Twelfth Annual ACM Symposium on Principles of Distributed Computing (PODC '93)*. ACM, New York, NY, USA, 145–157. https://doi.org/10.1145/164051.164070

[13] Ammar Qadri. 2017. m-Consensus Objects Are Pretty Powerful. In *20th International Conference on Principles of Distributed Systems (OPODIS 2016) (Leibniz International Proceedings in Informatics (LIPIcs))*, Panagiota Fatourou, Ernesto Jiménez, and Fernando Pedone (Eds.), Vol. 70. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 27:1–27:14. https://doi.org/10.4230/LIPIcs.OPODIS.2016.27

[14] Ophir Rachman. 1994. Anomalies in the Wait-free Hierarchy. In *Distributed Algorithms: 8th International Workshop, WDAG '1994 Terschelling, The Netherlands, September 29 – October 1, 1994 Proceedings*, Gerard Tel and Paul Vitányi (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 156–163. https://doi.org/10.1007/BFb0020431