On the Number of Objects with Distinct Power and the Linearizability of Set Agreement Objects

David Yu Cheng Chan[1], Vassos Hadzilacos[2], and Sam Toueg[3]

1 **Department of Computer Science, University of Toronto, Ontario, Canada**
davidchan@cs.toronto.edu
2 **Department of Computer Science, University of Toronto, Ontario, Canada**
vassos@cs.toronto.edu
3 **Department of Computer Science, University of Toronto, Ontario, Canada**
sam@cs.toronto.edu

Abstract

We first prove that there are uncountably many objects with distinct computational powers. More precisely, we show that there is an uncountable set of objects such that for any two of them, at least one cannot be implemented from the other (and registers) in a wait-free manner. We then strengthen this result by showing that there are uncountably many *linearizable* objects with distinct computational powers. To do so, we prove that for all positive integers n and k, there is a linearizable object that is computationally equivalent to the k-set agreement task among n processes. To the best of our knowledge, these are the first linearizable objects proven to be computationally equivalent to set agreement tasks.

1998 ACM Subject Classification C.2.4 Distributed Systems

Keywords and phrases Set Agreement, Asynchronous System, Shared Memory

Digital Object Identifier 10.4230/LIPIcs.DISC.2017.12

1 Introduction

One of the fundamental problems in distributed computing is to determine whether two shared objects are *equivalent*, i.e., whether each can be implemented from the other (and registers) in a wait-free manner.[1] Any two objects that are not equivalent do not have the same computational power, since at least one cannot implement the other.

To address this fundamental problem, Herlihy proposed the following object classification scheme: an object O is in level n of a hierarchy if, together with registers, O can be used to solve consensus among at most n processes [12]. It is clear that objects in different levels of this hierarchy are not equivalent. Unfortunately, the converse is not true: every level $n \in \mathbb{Z}^+$ of this hierarchy contains objects that are not equivalent [2, 7, 11, 16].[2] So Herlihy's hierarchy does not classify objects in a "precise" way. This motivates the search for a *precise* object classification scheme, i.e., one that partitions the universe $\mathcal{U}$ of all shared objects such that the following property holds: two objects are equivalent *if and only if* they are in the same cell of the partition.

In this paper, we first prove that there is an uncountable number of objects that are not equivalent to one another (and so they have distinct computational power). Thus any precise

[1] Throughout this paper, we consider only objects, tasks, and implementations that are *wait free*, so we subsequently omit all references to wait freedom.
[2] We denote by $\mathbb{Z}^+$ the set of all positive integers.

classification scheme of the universe $\mathcal{U}$ of objects contains an *uncountable* number of cells. So the cells of a precise classification scheme cannot be labeled using simple integers (as in Herlihy's hierarchy) or finite sequences of integers.

Our proof of the above result uses the (n, k)*-set agreement task* where each of n processes has a proposal value and must decide on one of the proposed values such that there are at most k distinct decision values [8]. The *set agreement power* of an object O is the infinite sequence $\vec{n} = (n_1, n_2, \ldots, n_k, \ldots)$ where, for all $k \geq 1$, n_k is the largest integer such that instances of O and registers can solve the (n_k, k)-set agreement task, or ∞ if instances of O and registers can solve the (n, k)-set agreement task for every integer n [10].

Let $\mathcal{S}$ denote the set of all infinite sequences of positive integers $\vec{n} = (n_1, n_2, \ldots, n_k, \ldots)$ such that $n_{k+1} \geq 2n_k$ for all $k \geq 1$. We use a result in [10] to prove that:

For all $\vec{n} \in \mathcal{S}$, there is an object $R_{\vec{n}}$ with set agreement power $\vec{n}$. (1)

Note that this is not obvious because it is *not* the case that *every* infinite sequence of positive integers $\vec{n} = (n_1, n_2, \ldots, n_k, \ldots)$ has a corresponding object $R_{\vec{n}}$ with set agreement power $\vec{n}$. Finally, we use a standard diagonalization argument to prove that the set $\mathcal{S}$ is uncountable. Therefore, by (1), there is an uncountable number of objects, namely the $R_{\vec{n}}$ objects, that have distinct set agreement power. Since objects with different set agreement power are not equivalent, we conclude that there are uncountably many objects that are not equivalent (and so they have distinct computational power).

Next, we prove that the above result holds even if we restrict the universe $\mathcal{U}$ of objects to contain only objects that are *linearizable* [13]. This result would be immediate if the $R_{\vec{n}}$ objects used in our proof were such objects. Our $R_{\vec{n}}$ objects, however, are *not* linearizable: this is because they are constructed using (n, k)-set agreement objects which, as described in [5, 9], are simply "*black-boxes*" that solve the (n, k)-set agreement task. In fact, to the best of our knowledge, all the (n, k)-set agreement objects used in the literature to date have *not* been defined as linearizable objects.

To show that there are uncountably many *linearizable* objects with distinct computational power, we proceed as follows:

1. We first prove that for all positive integers n and k, there is a linearizable object, denoted $LSA(n, k)$, that is computationally equivalent to the (n, k)-set agreement task in the following sense: the (n, k)-set agreement task can be solved using the $LSA(n, k)$ object, and the $LSA(n, k)$ object can be implemented using any solution to the (n, k)-set agreement task (and registers). This also implies that the linearizable $LSA(n, k)$ object is equivalent to the (n, k)-set agreement "black-box" object.
2. We then construct linearizable objects, denoted $LR_{\vec{n}}$, that are equivalent to the $R_{\vec{n}}$ objects. Roughly speaking we do so by replacing the (n, k)-set agreement "black-box" objects used to construct $R_{\vec{n}}$ with our linearizable $LSA(n, k)$ objects. Since there is an uncountable number of $R_{\vec{n}}$ objects with distinct computational power, and $R_{\vec{n}}$ is equivalent to $LR_{\vec{n}}$, there is also an uncountable number of linearizable $LR_{\vec{n}}$ objects with distinct computational power.

Proving that there is a linearizable object $LSA(n, k)$ that is *computationally* equivalent to the (n, k)-set agreement task is not obvious because the two are not *behaviourally* equivalent. Indeed, any linearizable object for the (n, k)-set agreement task imposes restrictions that are not inherent to this task [15, 6]. To see this, suppose that all the proposal values are distinct, and two processes propose concurrently. With the (n, k)-set agreement task, each of these two processes could decide the proposal value of the other. But a linearizable (n, k)-set agreement object does not allow this behaviour: whichever of the two processes is linearized first cannot decide the proposal value of the other process. Since the set of behaviours allowed by linearizable (n, k)-set agreement objects is a proper subset of those allowed by the corresponding task, it is conceivable that such objects inherently have *greater computational power* than the task. Our result about the $LSA(n, k)$ objects shows that this is not the case.

In summary, the main contributions of this paper are to show that:

1. For all $n, k \in \mathbb{Z}^+$, there is a linearizable object that is computationally equivalent to the (n, k)-set agreement task.
2. The number of linearizable objects with distinct computational power is uncountable.

2 Some Basic Definitions

Object equivalence. Given any pair of shared memory objects O and O', we denote by $O \succeq O'$ the relation: there exists an implementation of O' from instances of O and registers. We say that O and O' are *equivalent*, denoted $O \equiv O'$, if and only if $O \succeq O'$ and $O' \succeq O$. Furthermore, given any object O and any *collection* of objects $\mathcal{C}$, (i) we denote by $\mathcal{C} \succeq O$ the relation: there exists an implementation of O from instances of objects in $\mathcal{C}$ and registers, and (ii) we denote by $O \succeq \mathcal{C}$ the relation: there exists an implementation of each object in $\mathcal{C}$ from instances of O and registers. We say O and $\mathcal{C}$ are *equivalent*, denoted $O \equiv \mathcal{C}$, if and only if $O \succeq \mathcal{C}$ and $\mathcal{C} \succeq O$.

Redirection objects. Let $\mathbb{Z}^*$ denote the set of all positive integers and the value ∞, i.e., $\mathbb{Z}^* = \mathbb{Z}^+ \cup \{\infty\}$. For all $n, k \in \mathbb{Z}^*$, we denote by $SA(n, k)$ the "black-box" (n, k)*-set agreement object* described in [5, 9]. Then, given any infinite sequence $\vec{n} = (n_1, n_2, \ldots, n_k, \ldots)$ such that $n_k \in \mathbb{Z}^*$ for all $k \in \mathbb{Z}^+$, we define a "redirection" object $R_{\vec{n}}$ that is equivalent to the collection $\mathcal{SA}_{\vec{n}}$ of set agreement objects $\bigcup_{k=1}^{\infty}\{SA(n_k, k)\}$. The object $R_{\vec{n}}$ supports the operation $\text{PROPOSE}(v, k)$, for any value v and any integer $k \in \mathbb{Z}^+$. Intuitively, when an operation $\text{PROPOSE}(v, k)$ is applied to the $R_{\vec{n}}$ object, it is redirected and applied as a $\text{PROPOSE}(v)$ operation on the (n_k, k)-set agreement object $SA(n_k, k)$ in $\mathcal{SA}_{\vec{n}}$ and the response is returned. More precisely, for each $k \in \mathbb{Z}^+$, consider the set of all $\text{PROPOSE}(-, k)$ operations that are applied to the object $R_{\vec{n}}$. The values returned to these operations satisfy the properties of the $SA(n_k, k)$ object: If there are at most n_k such operations, then (a) k*-agreement*: at most k distinct values are returned to these operations, and (b) *validity*: each value returned to these operations was proposed by one of them.

▶ **Observation 1.** *For all infinite sequences $\vec{n} = (n_1, n_2, \ldots, n_k, \ldots)$ such that $n_k \in \mathbb{Z}^*$ for all $k \in \mathbb{Z}^+$, we have:*

(a) *For all $k \in \mathbb{Z}^+$, $R_{\vec{n}} \succeq SA(n_k, k)$; so $R_{\vec{n}} \succeq \mathcal{SA}_{\vec{n}}$.*
(b) *$\mathcal{SA}_{\vec{n}} \succeq R_{\vec{n}}$.*
(c) *Thus $R_{\vec{n}} \equiv \mathcal{SA}_{\vec{n}}$.*

3 Uncountability of Objects with Distinct Power

Let b be a non-negative integer, and $(a_1, a_2, \ldots)$ be non-negative integers such that only finitely many of the a_ℓ's are non-zero. Suppose that for each $\ell \in \mathbb{Z}^+$, we are given a_ℓ copies of $SA(n_\ell, \ell)$ objects. Using these objects, we can solve k-set agreement among n processes where $n \leq b + \sum_{\ell=1}^{\infty} a_\ell n_\ell$, and $k \geq b + \sum_{\ell=1}^{\infty} a_\ell \ell$. To do so, we partition the n processes as follows: for every a_ℓ that is non-zero, we create a_ℓ groups of at most n_ℓ processes each, and we also create one group of at most b processes. In each of the a_ℓ groups of at most n_ℓ processes, every process uses an $SA(n_\ell, \ell)$ object to propose its value and returns the object's response; in the last group of at most b processes, each process returns its proposal value. In this way, the n processes return at most $b + \sum_{\ell=1}^{\infty} a_\ell \ell$ distinct values, each proposed by some process. By extending a result of Chaudhuri and Reiners [9], Delporte *et al.* proved that the ability to partition the n processes in such a manner is also a *necessary* condition to solve k-set agreement among n processes [10]. The following theorem follows from Theorem 2 of [10]:

▶ **Theorem 2** (Extended Set Agreement Partial Order Theorem). *Let $m \in \mathbb{Z}^*$ and $k \in \mathbb{Z}^+$, and let $\vec{n} = (n_1, n_2, \ldots, n_\ell, \ldots)$ be an infinite sequence such that $n_\ell \in \mathbb{Z}^*$ for all $\ell \in \mathbb{Z}^+$. Then $\mathcal{SA}_{\vec{n}} \succeq SA(m,k)$ if and only if there exists an infinite sequence $(a_1, a_2, \ldots)$ of non-negative integers and an integer $b \in \mathbb{N}$ such that:*[3]

- $b + \sum_{\ell=1}^{\infty} a_\ell n_\ell \geq m.$
- $b + \sum_{\ell=1}^{\infty} a_\ell \ell \leq k.$

Recall that $\mathcal{S}$ is the set of all infinite sequences of positive integers $\vec{n} = (n_1, n_2, \ldots, n_k, \ldots)$ such that $n_{k+1} \geq 2n_k$ for all $k \in \mathbb{Z}^+$. We now prove some properties of $\mathcal{S}$ that will be useful for applying Theorem 2.

▶ **Lemma 3.** *For all $\vec{n} \in \mathcal{S}$ and all $k \in \mathbb{Z}^+$, $\frac{n_{k+1}}{k+1} \geq \frac{n_k}{k}$.*

Proof. By definition, for all $\vec{n} \in \mathcal{S}$ and $k \in \mathbb{Z}^+$, $n_{k+1} \geq 2n_k$. Furthermore, since $k \in \mathbb{Z}^+$, $\frac{2}{k+1} \geq \frac{1}{k}$. Consequently, $\frac{n_{k+1}}{k+1} \geq \frac{2n_k}{k+1} = (\frac{2}{k+1})n_k \geq (\frac{1}{k})n_k = \frac{n_k}{k}$. ◀

▶ **Corollary 4.** *For all $\vec{n} \in \mathcal{S}$ and all $k, k' \in \mathbb{Z}^+$ where $k' \geq k$, $\frac{n'_k}{k'} \geq \frac{n_k}{k}$.*

▶ **Observation 5.** *For all $\vec{n} \in \mathcal{S}$ and all $k \in \mathbb{Z}^+$, $n_k \geq k$.*

▶ **Lemma 6.** *For all $\vec{n} \in \mathcal{S}$ and all $k \in \mathbb{Z}^+$, $\mathcal{SA}_{\vec{n}} \not\succeq SA(n_k + 1, k)$.*

Proof. Let $\vec{n} = (n_1, n_2, \ldots, n_k, \ldots)$ be an infinite sequence in $\mathcal{S}$ and let $k \in \mathbb{Z}^+$. Furthermore, let $\mathcal{A}$ be the set of all infinite sequences of non-negative integers. Then for all $a = (a_1, a_2, \ldots, a_k, \ldots) \in \mathcal{A}$ and $b \in \mathbb{N}$, we define the predicate $\mathcal{P}(a, b, \vec{n}, k)$ to be true if and only if the following inequalities are true:

$$b + \sum_{\ell=1}^{\infty} a_\ell n_\ell \geq n_k + 1 \tag{2}$$

$$b + \sum_{\ell=1}^{\infty} a_\ell \ell \leq k \tag{3}$$

By Theorem 2, it suffices to show that $\mathcal{P}(a, b, \vec{n}, k)$ is false for all $a \in \mathcal{A}$ and $b \in \mathbb{N}$.

Case 1. There exists an integer $k' > k$ such that $a_{k'} > 0$.
Then $b + \sum_{\ell=1}^{\infty} a_\ell \ell \geq a_{k'} k' \geq k' > k$. Thus inequality (3) is false, and so $\mathcal{P}(a, b, \vec{n}, k)$ is false.

Case 2. For all $k' \in \mathbb{Z}^+$ such that $k' > k$, we have $a_{k'} = 0$.

Case 2(a). $b + \sum_{\ell=1}^{k} a_\ell = 0$
Since a is a sequence of non-negative integers and b is a non-negative integer, this implies $b = 0$ and for all $1 \leq \ell \leq k$, $a_\ell = 0$. Then $b + \sum_{\ell=1}^{\infty} a_\ell n_\ell = 0 < n_k + 1$. Thus the inequality (2) is false, and so $\mathcal{P}(a, b, \vec{n}, k)$ is false.

[3] We denote by $\mathbb{N}$ the set of all natural numbers, including 0.

Case 2(b). $b + \sum_{\ell=1}^{k} a_\ell > 0$.
Thus either $a_\ell > 0$ for some $1 \le \ell \le k$, or $b > 0$. We define the function:

$$f(a, b, \vec{n}, k) = k(b + \sum_{\ell=1}^{k} a_\ell n_\ell) - (n_k + 1)(b + \sum_{\ell=1}^{k} a_\ell \ell)$$

$f(a, b, \vec{n}, k)$ is the left side of (2) multiplied by the right side of (3), minus the left side of (3) multiplied by the right side of (2). Thus for all $a \in \mathcal{A}$ and $b \in \mathbb{N}$, if $\mathcal{P}(a, b, \vec{n}, k)$ is true, then $f(a, b, \vec{n}, k) \ge 0$.
By algebra,

$$\begin{aligned} f(a, b, \vec{n}, k) &= k(b + \sum_{\ell=1}^{k} a_\ell n_\ell) - (n_k + 1)(b + \sum_{\ell=1}^{k} a_\ell \ell) \\ &= bk - b(n_k + 1) + k(\sum_{\ell=1}^{k} a_\ell n_\ell) - (n_k + 1)(\sum_{\ell=1}^{k} a_\ell \ell) \\ &= b(k - (n_k + 1)) + (\sum_{\ell=1}^{k} k a_\ell n_\ell) - (\sum_{\ell=1}^{k} (n_k + 1) a_\ell \ell) \\ &= b(k - (n_k + 1)) + \sum_{\ell=1}^{k} (k a_\ell n_\ell - (n_k + 1) a_\ell \ell) \\ &= b(k - (n_k + 1)) + \sum_{\ell=1}^{k} a_\ell (k n_\ell - (n_k + 1)\ell) \end{aligned}$$

By Observation 5, $0 > k - (n_k + 1)$. Thus if $b > 0$, then $b(k - (n_k + 1)) < 0$, whereas if $b = 0$, then $b(k - (n_k + 1)) = 0$. By Corollary 4, for all $1 \le \ell \le k$,

$$\begin{aligned} &\frac{n_k}{k} \ge \frac{n_\ell}{\ell} \\ &\Rightarrow n_k \ell \ge k n_\ell \\ &\Rightarrow 0 \ge k n_\ell - n_k \ell \\ &\Rightarrow 0 > k n_\ell - (n_k + 1)\ell \end{aligned}$$

Thus for all $\ell \in [1..k]$, if $a_\ell > 0$, then $a_\ell(k n_\ell - (n_k + 1)\ell) < 0$, whereas if $a_\ell = 0$, then $a_\ell(k n_\ell - (n_k + 1)\ell) = 0$. Therefore, every term of the sum $b(k - (n_k + 1)) + \sum_{\ell=1}^{k} a_\ell(k n_\ell - (n_k + 1)\ell)$ is either 0 or negative. Furthermore, since $b + \sum_{\ell=1}^{k} a_\ell > 0$, either $a_\ell > 0$ for some $1 \le \ell \le k$, or $b > 0$, so at least one of the terms is negative. Therefore, $f(a, b, \vec{n}, k) < 0$, and so $\mathcal{P}(a, b, \vec{n}, k)$ is false.

So, for all $a \in \mathcal{A}$ and $b \in \mathbb{N}$, $\mathcal{P}(a, b, \vec{n}, k)$ is false. Thus $\mathcal{SA}_{\vec{n}} \not\succeq SA(n_k + 1, k)$. ◀

By Observation 1(b), $\mathcal{SA}_{\vec{n}} \succeq R_{\vec{n}}$, so by Lemma 6,

▶ **Corollary 7.** *For all $\vec{n} \in \mathcal{S}$ and all $k \in \mathbb{Z}^+$, $R_{\vec{n}} \not\succeq SA(n_k + 1, k)$.*

Since for all $\vec{n} \in \mathcal{S}$ and all $k \in \mathbb{Z}^+$, $R_{\vec{n}} \succeq SA(n_k, k)$ by Observation 1(a), by Corollary 7,

▶ **Corollary 8.** *For all $\vec{n} \in \mathcal{S}$, $R_{\vec{n}}$ has set agreement power $\vec{n}$.*

We now prove:

▶ **Lemma 9.** *$\mathcal{S}$ is uncountable.*

Proof. We prove this via a standard diagonalization argument. Assume, for contradiction, that $\mathcal{S}$ is countable. In other words, there exists some enumeration $\mathcal{E}$ of all elements in $\mathcal{S}$. For all $i \in \mathbb{Z}^+$ and $j \in \mathbb{Z}^+$, let $\mathcal{E}[i, j]$ denote the j-th number in the i-th sequence of the enumeration $\mathcal{E}$.

Now consider the infinite sequence $\vec{n} = (n_1, n_2, \ldots, n_k, \ldots)$ where:

- $n_1 = \mathcal{E}[1, 1] + 1$.
- For all $i \geq 2$, $n_i = \mathcal{E}[i, i] + 2n_{i-1}$.

Since $\mathcal{E}$ is an enumeration of infinite sequences of positive integers, n_i is a positive integer for all $i \in \mathbb{Z}^+$. Furthermore, by construction, n_i is at least twice as large as n_{i-1} for all $i \geq 2$. Consequently, $\vec{n}$ is an infinite sequence of positive integers where each integer is at least twice as large as its predecessor, so $\vec{n} \in \mathcal{S}$. By construction, however, $n_i \neq \mathcal{E}[i, i]$ for all $i \in \mathbb{Z}^+$, so $\vec{n}$ is not any of the infinite sequences in the enumeration $\mathcal{E}$ of $\mathcal{S}$. This is a contradiction, so we conclude that $\mathcal{S}$ is uncountable. ◀

▶ **Theorem 10.** *There are uncountably many objects that are not equivalent to each other.*

Proof. By Lemma 9, $\mathcal{S}$ is uncountable. By Corollary 8, for all $\vec{n} \in \mathcal{S}$, there is an object with set agreement power $\vec{n}$. So there are uncountably many objects with distinct set agreement power. By definition, objects with different set agreement power are not equivalent. ◀

Note that the uncountably many objects that Theorem 10 refers to were constructed using the "black-box" set agreement objects described in [5, 9]. In Section 5, we strengthen this result by proving that it holds even for *linearizable* objects [13]. To do this, we first prove that each set agreement task is equivalent to a linearizable object.

4 Linearizable Set Agreement Objects

Henceforth, we consider objects with *ports.* With such an object, each process can apply any operation to any port $i \in [1..n]$, and must then wait for a response from that port. We assume that accesses to the ports are *well-formed*: no port is accessed concurrently by multiple processes; i.e., while an operation on a port is pending, no process can apply another operation on that port. If accesses to the object are not well-formed, the behaviour of the object is undefined.

For all $n, k \in \mathbb{Z}^+$, we now define $LSA(n, k)$, a simple *linearizable* object that we will prove is *equivalent* to the (n, k)-set agreement task in the following sense: the (n, k)-set agreement task can be solved using the $LSA(n, k)$ object, and the $LSA(n, k)$ object can be implemented using any solution to the (n, k)-set agreement task (and registers).

The behaviour of the $LSA(n, k)$ object when it is accessed sequentially is given by its *sequential specification*, described below. The behaviour of $LSA(n, k)$ when it is accessed concurrently (in a well-formed manner) is *linearizable* [13].

The sequential specification of $LSA(n, k)$ is given by Algorithm 1. $LSA(n, k)$ has n *ports*: each process can apply a PROPOSE(v) operation for any value v to any port $i \in [1..n]$. The state of the $LSA(n, k)$ object consists of:

- The set V_{in} of all values proposed to $LSA(n, k)$; V_{in} is initially empty.
- The set V_{out} of all values returned by $LSA(n, k)$; V_{out} is initially empty.

The sequential specification of $LSA(n, k)$ 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.

From the above definition of $LSA(n, k)$, we have:

Algorithm 1 Sequential specification of the $LSA(n, k)$ object.

```
   Code for port $i \in [1..n]$:
1: procedure PROPOSE($v$)
2:     $V_{in} \leftarrow V_{in} \cup \{v\}$
3:     Let $v'$ be nondeterministically chosen from $V_{in}$ such that $|V_{out} \cup \{v'\}| \leq k$.
4:     $V_{out} \leftarrow V_{out} \cup \{v'\}$
5:     return $v'$
```

▶ **Observation 11.** *For all $n, k \in \mathbb{Z}^+$, if the $LSA(n, k)$ object is accessed in a well-formed manner, it satisfies:*

- *k-**agreement**: There are at most k distinct return values.*
- ***Validity**: If an operation op returns a value v, then v was proposed by op or by an operation linearized before op.*

This implies:

▶ **Observation 12.** *For all $n, k \in \mathbb{Z}^+$, the (n, k)-set agreement task can be solved using an $LSA(n, k)$ object.*

We now show that the converse also holds: for all $n, k \in \mathbb{Z}^+$, given any algorithm that solves the (n, k)-set agreement task, one can implement the linearizable object $LSA(n, k)$. This implementation of $LSA(n, k)$, shown in Algorithm 2, uses:

- $P[1..n]$: any algorithm that solves the (n, k)-set agreement task, where $P[i]$ is the protocol executed by process i.
- $X[1..n]$: an atomic snapshot object with n fields, initially all **nil**; each $X[i]$ stores a value output by $P[i]$.
- $R[1..n]$: an array of registers, initially all **nil**; each $R[i]$ stores the return value of the first operation performed on port i ($R[i]$ is used to have all operations on port i return the same value).

Note that the atomic snapshot object $X[1..n]$ can be implemented using only registers [1].

To perform an operation PROPOSE(v) on port i, a process executes the following steps:

1. It reads $R[i]$ and returns the same value as the previous operation on port i, if it exists (line 3).
2. It executes protocol $P[i]$ with proposal value v, and stores the decided value into a temporary local variable $pval_i$ (line 4).
3. It writes $pval_i$ to the i-th field of the atomic snapshot X (line 5), letting the other processes know that $pval_i$ was decided by protocol $P[i]$.
4. It takes a snapshot of X (line 6) to check whether its *own* proposed value v was decided by any protocol of $P[1..n]$; if so, it writes v to $R[i]$ (line 7), otherwise it writes $pval_i$ to $R[i]$ (line 8).
5. It returns the value in $R[i]$ (line 9).

Note that, for every $i \in [1..n]$, protocol $P[i]$ is only executed by the first operation on port i, and thus it is executed *at most once*.

▶ **Theorem 13.** *For all $n, k \in \mathbb{Z}^+$, Algorithm 2 implements the linearizable object $LSA(n, k)$ using any algorithm $P[1..n]$ that solves the (n, k)-set agreement task and registers.*

Proof. Let H be any history of this implementation of the $LSA(n, k)$ object where accesses to ports are well-formed.

Algorithm 2 Implementing the linearizable object $LSA(n,k)$ using any algorithm $P[1..n]$ that solves the (n,k)-set agreement task.

```
1: Code for port $i \in [1..n]$:
2: procedure PROPOSE($v$)
3:     if $R[i]$.READ() $\neq$ nil then return $R[i]$.READ()
4:     $pval_i \leftarrow P[i]$.EXECUTE($v$)
5:     $X[i] \leftarrow pval_i$
6:     $X' \leftarrow X$.SCAN()
7:     if $v \in X'$ then $R[i]$.WRITE($v$)
8:     else $R[i]$.WRITE($pval_i$)
9:     return $R[i]$.READ()
```

▶ **Claim 14** (k-agreement)**.** *In H, there are at most k distinct return values.*

Proof. A value v is returned by an operation on port i only if $v \neq \textbf{nil}$ has been written into $R[i]$ (line 3 or 9). A value v is written into $R[i]$ only if v was returned by X.SCAN() (lines 6 to 7) or v has been written into $pval_i$ (line 8). For $v \neq \textbf{nil}$ to be returned by X.SCAN(), v must have been written into X by an operation on some port i' (line 5), and therefore previously written into $pval_{i'}$. For all $j \in [1..n]$, v is assigned to $pval_j$ only if v was returned by the protocol $P[j]$ (line 4). Thus any value returned by the object must have previously been returned by some protocol of the (n,k)-set agreement task solution. Since each protocol $P[j]$ for $j \in [1..n]$ is executed at most once, at most k distinct values are returned by the protocols and hence the object. ◀

We now construct a completion H' of H as follows: for port i with an incomplete operation *op*, if *op* is the first operation on port i and some other port has a complete operation *op*$'$ in H that returns the value proposed by *op*, complete *op* immediately after it is invoked by returning its own proposal value (i.e., the same value that *op*$'$ returns); otherwise remove *op*. Next, we construct a linearization L of H' as follows:

1. Linearize every operation that returns its own proposal value at the point it is invoked.
2. Linearize every operation that returns via line 3 at the point it is invoked.
3. Linearize every remaining operation at the point when it takes a snapshot of X (line 6).

From the above, it is clear that every operation is linearized at some point during its execution interval.

▶ **Claim 15** (Validity)**.** *In H', if an operation op returns a value v, then v was proposed by op or by an operation linearized before op in the linearization L of H'.*

Proof. Suppose an operation *op* on port i proposes v' and returns v. If $v' = v$, the claim holds. Now suppose $v' \neq v$. Recall that when constructing the completion H' of H, incomplete operations are only completed by returning their own proposal values. Thus *op* is a complete operation in H. There are two cases:

Case 1: *op* is the first operation on port i.

Then, since *op* proposes v' and returns $v \neq v'$, from the code of Algorithm 2 and the way we linearize operations, it is clear that:

(1) *op* obtained $pval_i = v$ from executing $P[i]$.EXECUTE(v') in line 4,

(2) *op* wrote $pval_i = v$ in X in line 5, and

(3) *op* is linearized the instant it takes a snapshot of X in line 6.

Since $P[1..n]$ is a solution for the (n, k)-set agreement task, by (1), at least one port i' has an operation op' that starts executing $P[i']$.EXECUTE(v) *before or at the same time* as op obtains v in line 4. Clearly, op' proposes v and is invoked *before* it starts executing $P[i']$.EXECUTE(v), and so *before* op obtains v in line 4. We now show that op' is linearized before op, and therefore v was proposed by an operation linearized before op. There are three subcases:

Case 1(a): op' is incomplete in H.
First, recall that when constructing the completion H' of H, an incomplete operation in H is completed with its own proposal value if it is the first operation on its port, and its proposal value is decided by some complete operation in H. Since op' executes $P[i']$.EXECUTE(v), it is the first operation on port i'. Furthermore, a complete operation in H, namely op, returns the value v that is proposed by op', so op' is completed in H' by returning v immediately after it is invoked. Then, since op' returns its own proposal value, op' is linearized at the moment it is invoked. Thus op' is linearized before op obtains v in line 4. Therefore op' is linearized before op executes line 6, and thus before op is linearized.

Case 1(b): op' is complete in H and it returns the value v that it proposed.
Then op' is linearized at the moment op' is invoked. Thus op' is linearized before before op obtains v in line 4. Therefore op' is linearized before op executes line 6, and thus before op is linearized.

Case 1(c): op' is complete in H and it returns a value different from the value v that it proposed.
Since op' executes $P[i']$.EXECUTE(v), it is the first operation on port i'. Thus op' is linearized when it takes a snapshot of X in line 6, and this snapshot does not contain v (otherwise op' would return v). Since the snapshot does not contain v, it occurs *before* op writes v in X in line 5 (see (2) above). Therefore op' is linearized before op executes line 6, and thus before op is linearized.

Case 2: op is not the first operation on port i.
Then the first operation op' on port i also returns the same value v. Thus from case 1, v was proposed by op' or by an operation linearized before op'. Since the history H is well-formed, i.e., operations on port i are not concurrent, op' is linearized before op. So v was proposed by an operation linearized before op. ◀

Let H_L be the *sequential history* obtained by ordering all the operations in the complete history H' by their linearization points in L. Since the completion of H to H', and the linearization of H' to H_L does not introduce new return values, from Claim 14 we have:

▶ **Observation 16** (k-agreement)**.** *In H_L, there are at most k distinct return values.*

By the definition of H_L and Claim 15, we have:

▶ **Observation 17** (Validity)**.** *In H_L, if an operation op returns a value v, then v was proposed by op or by an operation before op.*

To prove that Algorithm 2 implements the linearizable object $LSA(n, k)$, it suffices to show that H_L satisfies the sequential specification of $LSA(n, k)$.

Suppose, for contradiction, that H_L violates the sequential specification of $LSA(n, k)$, and let op be the first operation in H_L that does so. Let v be the value proposed by op, and v' be the value returned by op. According to the sequential specification of $LSA(n, k)$, v' should be such that (i) v' is in $V_{in} \cup \{v\}$, and (ii) $|V_{out} \cup \{v'\}| \leq k$. Thus, since op violates the sequential specification of $LSA(n, k)$, either (i) v' is not in $V_{in} \cup \{v\}$, or (ii) $|V_{out} \cup \{v'\}| > k$. We consider these two cases below:

Case 1: v' is not in $V_{in} \cup \{v\}$.

By the sequential specification of $LSA(n, k)$, $V_{in} \cup \{v\}$ is the set of all values proposed by *op* and the operations before *op* in H_L. Thus *op* returns a value v' that was *not* proposed by *op* or by an operation that is before *op* in H_L. So H_L violates Observation 17.

Case 2: $|V_{out} \cup \{v'\}| > k$.

By the sequential specification of $LSA(n, k)$, $V_{out} \cup \{v'\}$ is the set of all values returned by *op* and the operations before *op* in H_L. Thus the operations in H_L return more than k distinct values. So H_L violates Observation 16. ◀

▶ **Theorem 18.** *For all $n, k \in \mathbb{Z}^+$, the linearizable object $LSA(n, k)$ is equivalent to the (n, k)-set agreement task, that is:*

(a) *The (n, k)-set agreement task can be solved using $LSA(n, k)$, and*

(b) *$LSA(n, k)$ can be implemented using any algorithm that solves the (n, k)-set agreement task (and registers).*

Proof. Part (a) follows by Observation 12, and Part (b) is immediate from Theorem 13. ◀

5 Uncountability of Linearizable Objects with Distinct Power

In this section, we prove that there are uncountably many linearizable objects with distinct computational power.

For every infinite sequence $\vec{n} = (n_1, n_2, \ldots, n_k, \ldots)$ such that $n_k \in \mathbb{Z}^+$ for all $k \in \mathbb{Z}^+$, we define a *linearizable* "redirection" object $LR_{\vec{n}}$ that is equivalent to the collection $\mathcal{LSA}_{\vec{n}}$ of linearizable objects $\bigcup_{k=1}^{\infty}\{LSA(n_k, k)\}$. The object $LR_{\vec{n}}$ has a port i for every integer $i \in \mathbb{Z}^+$, each process can apply a PROPOSE(v, k) operation for any value v and any integer $k \in \mathbb{Z}^+$ to any port $i \in \mathbb{Z}^+$. Intuitively, when an operation PROPOSE(v, k) is applied on a port i of the $LR_{\vec{n}}$ object, the operation PROPOSE(v) is applied to port i of the $LSA(n_k, k)$ object in $\mathcal{LSA}_{\vec{n}}$ and its response is returned. If no such port exists (because $i > n_k$), the operation simply returns $\perp$ without changing the state.

The behaviour of the $LR_{\vec{n}}$ object when it is accessed sequentially is given by its *sequential specification*, described below. The behaviour of $LR_{\vec{n}}$ when it is accessed concurrently (in a well-formed manner) is *linearizable* [13].

The sequential specification of $LR_{\vec{n}}$ is given by Algorithm 3. The state of the $LR_{\vec{n}}$ object consists of the following:

- For all $k \in \mathbb{Z}^+$, the set V^k_{in} of all the values proposed by PROPOSE$(-, k)$ operations on ports 1 to n_k of $LR_{\vec{n}}$; V^k_{in} is initially empty.
- For all $k \in \mathbb{Z}^+$, the set V^k_{out} of all the values returned by PROPOSE$(-, k)$ operations on ports 1 to n_k of $LR_{\vec{n}}$; V^k_{out} is initially empty.

From the above, it is clear that the sequential specification of $LR_{\vec{n}}$ 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.

▶ **Observation 19.** *For all infinite sequences $\vec{n} = (n_1, n_2, \ldots, n_k, \ldots)$ such that $n_k \in \mathbb{Z}^+$ for all $k \in \mathbb{Z}^+$, we have that:*

(a) *For all $k \in \mathbb{Z}^+$, $LR_{\vec{n}} \succeq LSA(n_k, k)$; so $LR_{\vec{n}} \succeq \mathcal{LSA}_{\vec{n}}$.*

(b) *$\mathcal{LSA}_{\vec{n}} \succeq LR_{\vec{n}}$.*

(c) *Thus $LR_{\vec{n}} \equiv \mathcal{LSA}_{\vec{n}}$.*

Algorithm 3 Sequential specification of the $LR_{\vec{n}}$ object.

```
   Code for port $i \in \mathbb{Z}^+$:
1: procedure PROPOSE$(v, k)$
2:     if $i > n_k$ then return $\perp$
3:     $V^k_{in} \leftarrow V^k_{in} \cup \{v\}$
4:     Let $v'$ be nondeterministically chosen from $V^k_{in}$ such that $|V^k_{out} \cup \{v'\}| \leq k$.
5:     $V^k_{out} \leftarrow V^k_{out} \cup \{v'\}$
6:     return $v'$
```

▶ **Lemma 20.** *For all infinite sequences $\vec{n} = (n_1, n_2, \ldots, n_k, \ldots)$ such that $n_k \in \mathbb{Z}^+$ for all $k \in \mathbb{Z}^+$, we have that the linearizable object $LR_{\vec{n}}$ is equivalent to the object $R_{\vec{n}}$.*

Proof. By Observation 1(c), $R_{\vec{n}} \equiv \mathcal{SA}_{\vec{n}}$, where $\mathcal{SA}_{\vec{n}}$ is the collection of "black-box" set agreement objects: $\bigcup_{k=1}^{\infty}\{SA(n_k, k)\}$. By Observation 19(c), $LR_{\vec{n}} \equiv \mathcal{LSA}_{\vec{n}}$, where $\mathcal{LSA}_{\vec{n}}$ is the collection of linearizable set agreement objects: $\bigcup_{k=1}^{\infty}\{LSA(n_k, k)\}$.

By definition, for all $k \in \mathbb{Z}^+$, the "black-box" set agreement object $SA(n_k, k)$ is equivalent to the (n_k, k)-set agreement task. By Theorem 18, for all $k \in \mathbb{Z}^+$, $LSA(n_k, k)$ is equivalent to the (n_k, k)-set agreement task. Thus, for all $k \in \mathbb{Z}^+$, $SA(n_k, k) \equiv LSA(n_k, k)$. Therefore, $\bigcup_{k=1}^{\infty}\{SA(n_k, k)\} \equiv \bigcup_{k=1}^{\infty}\{LSA(n_k, k)\}$, i.e., $\mathcal{SA}_{\vec{n}} \equiv \mathcal{LSA}_{\vec{n}}$. So, by transitivity, $R_{\vec{n}} \equiv LR_{\vec{n}}$. ◀

Consequently, by Corollary 8 and Lemma 20,

▶ **Corollary 21.** *For all $\vec{n} \in \mathcal{S}$, $LR_{\vec{n}}$ has set agreement power $\vec{n}$.*

▶ **Theorem 22.** *There are uncountably many linearizable objects that are not equivalent to each other.*

Proof. By Lemma 9, $\mathcal{S}$ is uncountable. By Corollary 21, for all $\vec{n} \in \mathcal{S}$, there is a linearizable object with set agreement power $\vec{n}$. So there are uncountably many linearizable objects with distinct set agreement power. By definition, objects with different set agreement power are not equivalent. ◀

Thus, there are uncountably many linearizable objects with distinct computational power.

6 Concluding remark

In this paper, we used Theorem 2 to prove that there are uncountably many objects with distinct computational power. We can use the same theorem to prove an interesting result about the *robustness* [14] of classifications of certain objects. Consider the subset of shared objects $\mathcal{U}_C \in \mathcal{U}$ that are equivalent to their set agreement power, namely:

$$\mathcal{U}_C = \{O \mid O \equiv \bigcup_{k=1}^{\infty}\{SA(n_k, k)\} \text{ where } \vec{n} = (n_1, n_2, \ldots) \text{ is the set agreement power of } O\}$$

In some sense, $\mathcal{U}_C$ is a generalization of the family of objects known as *Common2* [3, 4], which is the set of objects that are equivalent to the 2-consensus object. Thus $\mathcal{U}_C$ contains every object in Common2, which includes several common objects such as *stack*, *swap*, *fetch&add*, and *test&set* [3, 4].

If we restrict Herlihy's [12] consensus hierarchy to $\mathcal{U}_C$, we can prove the resulting hierarchy is *robust* [14] in the following sense: in $\mathcal{U}_C$, for all $n \in \mathbb{Z}^+$, any non-empty set of objects with consensus number at most n cannot be used to implement any object with consensus number $n' > n$. In fact, we can prove a more general result, as we now describe.

To describe our robustness result, we first define what it means for a sequence to *dominate* another. Given any pair of sequences $\vec{v} = (v_1, v_2, \ldots, v_\ell)$ and $\vec{v}' = (v'_1, v'_2, \ldots, v'_\ell)$ of the same length $\ell \in \mathbb{Z}^+$, we say that v *dominates* v', denoted $\vec{v} \geq \vec{v}'$, if for all $k \in [1..\ell]$, $v_k \geq v'_k$; similarly, we say that v *strictly dominates*, denoted $\vec{v} > \vec{v}'$, if $\vec{v} \geq \vec{v}'$ and $\vec{v} \neq \vec{v}'$.

Consider the set of objects $\mathcal{U}_C$ that are equivalent to their set agreement power. For any integer $\ell \in \mathbb{Z}^+$, we can partition $\mathcal{U}_C$ into equivalence classes such two objects are in the same class if and only if the first ℓ components of their set agreement powers are the same; we call this an *ℓ-partition of $\mathcal{U}_C$*, and denote it $\mathcal{P}_\ell$ (note that the 1-partition of $\mathcal{U}_C$ is simply Herlihy's consensus hierarchy restricted to the objects in $\mathcal{U}_C$ [12]). Let C be any equivalence class of $\mathcal{P}_\ell$. By definition, the first ℓ components of the set agreement power of every object in C is some sequence $\vec{v} = (v_1, v_2, \ldots, v_\ell)$; this sequence is the *label* of C. If C and C' are equivalence classes of $\mathcal{P}_\ell$ with labels $\vec{v}$ and $\vec{v}'$ respectively, we say that C *dominates* C' if $\vec{v} \geq \vec{v}'$, and C *strictly dominates* C' if $\vec{v} > \vec{v}'$.

Our generalized robustness result for can now be stated as follows: Consider the ℓ-partition $\mathcal{P}_\ell$ of $\mathcal{U}_C$, and let C be any equivalence class of $\mathcal{P}_\ell$. Objects in equivalence classes that are dominated by C cannot implement objects in equivalence classes that strictly dominate C. A proof of this result is given in the appendix.

References

1 Yehuda Afek, Hagit Attiya, Danny Dolev, Eli Gafni, Michael Merritt, and Nir Shavit. Atomic snapshots of shared memory. *J. ACM*, 40(4):873–890, Sep 1993. doi:10.1145/153724.153741.

2 Yehuda Afek, Faith Ellen, and Eli Gafni. Deterministic objects: Life beyond consensus. In *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing*, PODC'16, pages 97–106, New York, NY, USA, 2016. ACM. doi:10.1145/2933057.2933116.

3 Yehuda Afek, Eli Gafni, and Adam Morrison. Common2 extended to stacks and unbounded concurrency. *Distributed Computing*, 20(4):239–252, 2007. doi:10.1007/s00446-007-0023-3.

4 Yehuda Afek, Adam Morrison, and Guy Wertheim. From bounded to unbounded concurrency objects and back. In *Proceedings of the 30th Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, PODC'11, pages 119–128, New York, NY, USA, 2011. ACM. doi:10.1145/1993806.1993823.

5 Elizabeth Borowsky and Eli Gafni. The implication of the Borowsky-Gafni simulation on the set-consensus hierarchy. Technical Report 930021, UCLA Computer Science Dept., July 1993.

6 Armando Castañeda, Sergio Rajsbaum, and Michel Raynal. Specifying concurrent problems: Beyond linearizability and up to tasks. In Yoram Moses, editor, *Distributed Computing: 29th International Symposium, DISC 2015, Tokyo, Japan, October 7-9, 2015, Proceedings*, pages 420–435, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg. doi:10.1007/978-3-662-48653-5_28.

7 David Yu Cheng Chan, Vassos Hadzilacos, and Sam Toueg. Bounded disagreement. In Panagiota Fatourou, Ernesto Jiménez, and Fernando Pedone, editors, *20th International Conference on Principles of Distributed Systems (OPODIS 2016)*, volume 70 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 5:1–5:16, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.OPODIS.2016.5.

8 Soma Chaudhuri. More choices allow more faults: Set consensus problems in totally asynchronous systems. *Information and Computation*, 105(1):132–158, 1993. doi:http://dx.doi.org/10.1006/inco.1993.1043.

9 Soma Chaudhuri and Paul Reiners. Understanding the set consensus partial order using the Borowsky-Gafni simulation. In *Distributed Algorithms*, volume 1151 of *Lecture Notes in Computer Science*, pages 362–379. Springer Berlin Heidelberg, 1996. doi:10.1007/3-540-61769-8_23.

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

11 Maurice Herlihy. Impossibility results for asynchronous PRAM (extended abstract). In *Proceedings of the Third Annual ACM Symposium on Parallel Algorithms and Architectures*, SPAA'91, pages 327–336, New York, NY, USA, 1991. ACM. doi:10.1145/113379.113409.

12 Maurice Herlihy. Wait-free synchronization. *ACM Trans. Program. Lang. Syst.*, 11(1):124–149, Jan 1991. doi:10.1145/114005.102808.

13 Maurice P. Herlihy and Jeannette M. Wing. Linearizability: A correctness condition for concurrent objects. *ACM Trans. Program. Lang. Syst.*, 12(3):463–492, jul 1990. doi:10.1145/78969.78972.

14 Prasad Jayanti. On the robustness of herlihy's hierarchy. In *Proceedings of the Twelfth Annual ACM Symposium on Principles of Distributed Computing*, PODC'93, pages 145–157, New York, NY, USA, 1993. ACM. doi:10.1145/164051.164070.

15 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.

16 Ophir Rachman. Anomalies in the wait-free hierarchy. In *Proceedings of the 8th International Workshop on Distributed Algorithms*, WDAG'94, pages 156–163, London, UK, UK, 1994. Springer-Verlag. URL: http://dl.acm.org/citation.cfm?id=645951.675479.

A Appendix

▶ **Theorem 23.** *Consider the ℓ-partition $\mathcal{P}_\ell$ of $\mathcal{U}_C$, and let C be any equivalence class of $\mathcal{P}_\ell$. Objects in equivalence classes that are dominated by C cannot implement objects in equivalence classes that strictly dominate C.*

Proof. Assume, for contradiction, that there is an equivalence class C of $\mathcal{P}_\ell$ and a set of objects $\mathcal{O}$ such that:

(a) every object in $\mathcal{O}$ is in an equivalence class that is dominated by C;

(b) $\mathcal{O}$ implements an object O', i.e., $\mathcal{O} \succeq O'$; and

(c) O' is in an equivalence class C' that strictly dominates C.

Let $\vec{v} = (v_1, v_2, \ldots, v_\ell)$ and $\vec{v}' = (v'_1, v'_2, \ldots, v'_\ell)$ be the labels of C and C', respectively. Since C' dominates C, $\vec{v}' > \vec{v}$, and so there is a $d \in [1..\ell]$ such that $v'_d > v_d$. Since O' is in C', and the label of C' is $\vec{v}' = (v'_1, v'_2, \ldots, v'_\ell)$, O' implements the (v'_k, k)-set agreement object $SA(v'_k, k)$ for each $k \in [1..\ell]$. In particular, O' implements $SA(v'_d, d)$, i.e., $O' \succeq SA(v'_d, d)$.

Henceforth, O^i is an arbitrary object in $\mathcal{O}$. Let $\vec{n}^i = (n^i_1, n^i_2, \ldots, n^i_k, \ldots)$ be the set agreement power of O^i. Since every object in $\mathcal{O}$ (including O^i) is in an equivalence class that is dominated by C, and C has label $\vec{v} = (v_1, v_2, \ldots, v_\ell)$, we have that for all $k \in [1..\ell]$,

$n^i_k \leq v_k$. Let $\mathcal{SA}_{\vec{n}^i}$ be the collection of set agreement objects $\bigcup_{k=1}^{\infty}\{SA(n^i_k, k)\}$. Since $O^i \in \mathcal{U}_C$, by definition of $\mathcal{U}_C$, $O^i \equiv \mathcal{SA}_{\vec{n}^i}$.

Let $\vec{n} = (n_1, n_2, \ldots, n_k, \ldots)$ be the infinite sequence such that for all $k \in [1..\ell]$, $n_k = v_k$, and for all $k > \ell$, $n_k = \infty$. Thus for all $k \in \mathbb{Z}^+$, $n_k \geq n^i_k$. Let $\mathcal{SA}_{\vec{n}}$ be the collection of set agreement objects $\bigcup_{k=1}^{\infty}\{SA(n_k, k)\}$. Since for all $k \in \mathbb{Z}^+$, $n_k \geq n^i_k$, we have $\mathcal{SA}_{\vec{n}} \succeq \mathcal{SA}_{\vec{n}^i}$. Since $O^i \equiv \mathcal{SA}_{\vec{n}^i}$, $\mathcal{SA}_{\vec{n}} \succeq O^i$. Recall that O^i is an arbitrary object in $\mathcal{O}$, so $\mathcal{SA}_{\vec{n}}$ implements every object in $\mathcal{O}$, i.e., $\mathcal{SA}_{\vec{n}} \succeq \mathcal{O}$. Since $\mathcal{O} \succeq O'$ and $O' \succeq SA(v'_d, d)$, we have $\mathcal{SA}_{\vec{n}} \succeq SA(v'_d, d)$.

Since $\mathcal{SA}_{\vec{n}} = \bigcup_{k=1}^{\infty}\{SA(n_k, k)\}$ implements $SA(v'_d, d)$, by Theorem 2, there is an infinite sequence $(a_1, a_2, \ldots)$ of non-negative integers and an integer $b \in \mathbb{N}$ such that:

$$b + \sum_{k=1}^{\infty} a_k n_k \geq v'_d$$
$$b + \sum_{k=1}^{\infty} a_k k \leq d$$

Note that for all $k > d$, $a_k = 0$, otherwise $b + \sum_{k=1}^{\infty} a_k k > d$. Thus we have:

$$b + \sum_{k=1}^{d} a_k n_k \geq v'_d$$
$$b + \sum_{k=1}^{d} a_k k \leq d$$

Let O^* be an arbitrary object in the equivalence class C, and $\vec{n}^* = (n^*_1, n^*_2, \ldots, n^*_k, \ldots)$ be the set agreement power of O^*. Since O^* is in C and the label of C is $\vec{v} = (v_1, v_2, \ldots, v_\ell)$, for all $k \in [1..\ell]$, $n^*_k = v_k = n_k$. Thus, since $d \in [1..\ell]$, we have:

$$b + \sum_{k=1}^{d} a_k n^*_k \geq v'_d$$
$$b + \sum_{k=1}^{d} a_k k \leq d$$

Since for all $k > d$, $a_k = 0$, we have:

$$b + \sum_{k=1}^{\infty} a_k n^*_k \geq v'_d$$
$$b + \sum_{k=1}^{\infty} a_k k \leq d$$

Let $\mathcal{SA}_{\vec{n}^*}$ be the collection of set agreement objects $\bigcup_{k=1}^{\infty}\{SA(n^*_k, k)\}$. By the above equations and Theorem 2, $\mathcal{SA}_{\vec{n}^*} \succeq SA(v'_d, d)$. Since the set agreement power of O^* is $\vec{n}^* = (n^*_1, n^*_2, \ldots, n^*_k, \ldots)$ and $O^* \in \mathcal{U}_C$, by definition of $\mathcal{U}_C$, we have that $O^* \equiv \mathcal{SA}_{\vec{n}^*}$. Thus $O^* \succeq SA(v'_d, d)$. Hence the d-set agreement number of O^* is at least $v'_d > v_d$. However, recall that for all $k \in [1..\ell]$, $n^*_k = v_k$, so in particular $n^*_d = v_d$. Therefore the d-set agreement number of O^* is v_d — a contradiction. ◀