Bounded Disagreement

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

A well-known generalization of the consensus problem, namely, *set agreement (SA)*, limits the number of distinct *decision values* that processes decide. In some settings, it may be more important to limit the number of "disagreers". Thus, we introduce another natural generalization of the consensus problem, namely, *bounded disagreement (BD)*, which limits the number of *processes* that decide differently from the plurality. More precisely, in a system with n processes, the (n, ℓ)-BD task has the following requirement: there is a value v such that at most ℓ processes (the disagreers) decide a value other than v. Despite their apparent similarities, the results described below show that bounded disagreement, consensus, and set agreement are in fact fundamentally different problems.

We investigate the relationship between bounded disagreement, consensus, and set agreement. In particular, we determine the *consensus number* [16] for every instance of the BD task. We also determine values of n, ℓ, m, and k such that the (n, ℓ)-BD task can solve the (m, k)-SA task (where m processes can decide at most k distinct values). Using our results and a previously-known impossibility result for set agreement [8], we prove that for all $n \geq 2$, there is a BD task (and a corresponding BD object) that has consensus number n but can *not* be solved using n-consensus and registers. Prior to our paper, the only objects known to have this unusual characteristic for $n \geq 2$ (which shows that the consensus number of an object is not sufficient to fully capture its power) were artificial objects crafted solely for the purpose of exhibiting this behaviour [2, 18].

1998 ACM Subject Classification C.2.4 Distributed Systems

Keywords and phrases Consensus, Set Agreement, Asynchronous System, Distributed Algorithms, Shared Memory

Digital Object Identifier 10.4230/LIPIcs.OPODIS.2016.5

1 Introduction

Reaching agreement among processes is a fundamental problem in fault-tolerant distributed computing. The strongest and most-studied formulation of such a problem is the *consensus* task [12], where all non-faulty processes must decide on *one* of their input values. Another well-known agreement problem is a generalization of consensus, namely, the *set agreement (SA)*[1] task [7], which limits the number of distinct *decision values* that processes decide. In this paper we introduce and study another natural generalization of consensus, one that

[1] Also known as *set consensus*.

limits the number of *processes* that decide differently from the plurality; we call this the *bounded disagreement (BD)* task.

In the consensus, SA and BD tasks, there are n processes, each process has an input value (its *proposal*) and must output a value (its *decision*) such that every correct process eventually outputs a decision value (*Termination*) and each decision is one of the proposals (*Validity*). Additionally, the (n, k)-SA task must also satisfy the following property:

- **k-SA**: There exist at most k distinct decision values.

For $k = 1$, the (n, k)-SA task is simply the n-consensus task; in this case, there is only one decision value and hence no disagreers. For $k = 2$, however, there are up to two decision values, but the number of disagreers (i.e., those deciding differently from the majority) can suddenly jump all to the way to $\lfloor n/2 \rfloor$. In some settings, it may be more important to limit the number of processes that disagree than limiting the number of different decision values. Thus we introduce the (n, ℓ)-BD task that replaces the k-SA property with the following one:

- **ℓ-BD**: There is a value v such that at most ℓ processes (the *disagreers*) decide a value other than v.

For $\ell = 0$, the (n, ℓ)-BD task is just the n-consensus task. Furthermore, having at most ℓ disagreers implies having at most $\ell + 1$ distinct decision values, so the (n, ℓ)-BD task is at least as strong as the $(n, \ell + 1)$-SA task.

In this paper, we study the relationship between the consensus, set agreement and bounded disagreement tasks, and show that despite their similarity, these tasks are fundamentally different. To do so, we first determine values of n, ℓ, m and k such that the (n, ℓ)-BD task and registers can solve the (m, k)-SA task. We then determine the *consensus number* [16] of every instance of the BD task: specifically, we prove that the (n, ℓ)-BD task has consensus number $\max(n - 2\ell, 1)$.[2] Using the above results and a previously-known impossibility result for set agreement [8], we prove that BD tasks have the following uncommon property: for all $j \geq 2$, there is a BD task, namely the $(9j, 4j)$-BD task, that has consensus number j but *cannot* be solved using j-consensus objects and registers. These results imply that there are infinitely many instances of the BD task that are not equivalent to any consensus task or any set agreement task (because for $k \geq 2$ all (m, k)-SA tasks have consensus number 1).[3]

We also define the (n, ℓ)-BD *object* that solves the (n, ℓ)-BD *task*, as follows: intuitively, processes can "propose" any value to this object and it responds to the first n such proposals with values that satisfy the Validity and ℓ-BD properties; all subsequent proposals return arbitrary responses. It turns out that the above results about BD tasks also apply to the corresponding BD objects. In particular, for all $j \geq 2$, the $(9j, 4j)$-BD object has consensus number j, but cannot be implemented using j-consensus and registers. The existence of objects with this unusual property has been the subject of much research [2, 3, 4, 5, 9, 10, 11, 15, 18]. Prior to this paper, however, the only objects known to have this property for $j \geq 2$ were artificial objects crafted solely for the purpose of exhibiting this behaviour [2, 18]. Unlike these objects, however, our BD objects and tasks are natural generalizations of consensus. The BD objects and those in [18] are non-deterministic, while the objects in [2] are deterministic.

[2] The consensus number of a task T is the maximum number of processes for which consensus can be solved using any solution of T and registers.

[3] We say that two tasks are equivalent if any solution to one task, together with registers, can be used to solve the other task. Similarly, we say that two objects are equivalent if instances of either object, together with registers, can be used to implement the other.

Roadmap. In Section 2, we determine values of ℓ, m, k, and n such that the (m, ℓ)-BD task can solve the (n, k)-SA task. In Section 3, we define BD objects that can be used to solve BD tasks, and, in Section 4, we investigate what instances of consensus can *not* be solved using these BD objects. Together, these results allow us to determine the consensus number of every instance of the BD task and BD object. In Section 5, we prove that for all $n \geq 2$, there are instances of the BD task and BD object that have consensus number n, yet cannot be implemented using n-consensus and registers. In Section 6, we conclude with a brief discussion on bounded disagreement and related problems.

2 Using BD Tasks to Solve SA Tasks

Recall that with the SA and BD tasks, each process starts with an input value (its *proposal*) and must output a value (its *decision*), subject to certain restrictions. A solution to such a task is a collection of protocols $\mathcal{P}$, one for each process to execute in order to produce decisions that respect these restrictions.

In this section, we show that for all $k \geq 1$, $n \geq 1$, $\ell \geq 0$ and $m \geq \max(n+\ell+\lfloor \ell/k \rfloor, 2\ell+2)$, the (m, ℓ)-BD task and registers can solve the (n, k)-SA task.[4] For $k = 1$ and $n = m - 2\ell$, this says that the (m, ℓ)-BD task and registers can solve the $(m - 2\ell, 1)$-SA task, i.e., the $(m - 2\ell)$-consensus task.

We prove this result in two steps: we first assume that processes have access to a *Fetch&Increment* object, and then we remove this assumption. Intuitively, a Fetch&Increment object $\mathcal{F}$ is just a counter: each operation on $\mathcal{F}$ returns the current value of the counter and increments it by one [5].

Multisets notation. It is convenient to use a *multiset* to store the outputs of a BD task. Given any multiset M, the *mode of* M, denoted *mode*(M), is the value that appears the most often in M (ties are broken arbitrarily), and $|M|$ is the total number of (non-necessarily distinct) elements in M. The $+$ operator denotes the *disjoint union* of two multisets. This is distinct from the normal union operator $\cup$ in that the elements are added regardless of whether they are distinct from other elements.

▶ **Theorem 1.** *For all $k \geq 1$, $n \geq 1$, $\ell \geq 0$ and $m \geq n + \ell + \lfloor \ell/k \rfloor$, the (m, ℓ)-BD task and registers, together with a Fetch&Increment object, can solve the (n, k)-SA task.*

Proof. Consider any $k \geq 1$, $n \geq 1$, $\ell \geq 0$ and $m \geq n + \ell + \lfloor \ell/k \rfloor$. We present an algorithm that solves the (n, k)-SA task using:

- $\mathcal{P}[1..m]$, a solution for the (m, ℓ)-BD task.
- $\mathcal{X}$, an n-process snapshot object where each field contains a multiset of integers; initially empty. (Note that snapshot objects can be implemented from registers [1].)
- $\mathcal{F}$, a Fetch&Increment object; initialized such that the first operation will have response 1.

Recall that an n-process snapshot object $\mathcal{X}$ is composed of n fields, and provides two types of operations, write and snapshot. A write operation, denoted by $\mathcal{X}[p]$.WRITE(v), replaces the value of field $\mathcal{X}[p]$ by v. A snapshot operation, denoted by $\mathcal{X}$.SNAPSHOT(), returns the values in all n fields at once.

Intuitively, each process p uses $\mathcal{F}$ to determine which protocol(s) in $\mathcal{P}[1..m]$ to execute, and uses the field $\mathcal{X}[p]$ to announce the multiset of all the outputs that p has received from

[4] That is, *any* solution to the (m, ℓ)-BD task can be used, together with registers, to derive a solution to the (n, k)-SA task.

Algorithm 1 Solving the (n, k)-SA task using an (m, ℓ)-BD task solution $\mathcal{P}[1..m]$, an atomic snapshot $\mathcal{X}$, and a Fetch&Increment object $\mathcal{F}$.

```
1: while TRUE do
2:     next ← F.FETCH&INCREMENT()
3:     if next ≤ m then
4:         received ← received + {P[next].execute(v_p)}
5:         X[p].WRITE(received)
6:     else
7:         snap ← X.SNAPSHOT()
8:         decide mode(snap)
9:         halt
```

executing protocols in $\mathcal{P}[1..m]$ so far. In an abuse of notation, we will also use $\mathcal{X}$ to denote the multiset formed by the disjoint union of the multisets in the n fields of $\mathcal{X}$.

In addition to these shared objects, each process p also uses the following local variables:

- *received*: a multiset used to store all outputs that p has received from executing protocols in $\mathcal{P}[1..m]$; initially empty.
- *snap*: a multiset used to store a snapshot of $\mathcal{X}$; initially empty.
- *next*: a positive integer.

We denote by var_p the local variable *var* of process p, and by v_p the input value of p. We now describe the algorithm that each process p executes to solve the (n, k)-SA task:

1. It executes a Fetch&Increment operation on $\mathcal{F}$ to get a positive integer i (line 2).
2. If $i \leq m$, it executes protocol $\mathcal{P}[i]$ with input value v_p, and adds the output to $\mathcal{X}[p]$ (lines 3 to 5).
3. If $i > m$, it takes a snapshot of $\mathcal{X}$, decides mode($\mathcal{X}$), and halts (lines 6 to 9).
4. It repeats from the first step.

For simplicity, let $\mathcal{P}[i].execute(v)$ denote the output from executing protocol $\mathcal{P}[i]$ with input value v. The pseudocode for each process p in shown in Algorithm 1.

▶ **Claim 2.** *Each protocol of $\mathcal{P}[1..m]$ is executed at most once.*

Proof. Follows immediately from the fact that each protocol $\mathcal{P}[i]$ for all $1 \leq i \leq m$ is executed only when the Fetch&Increment object $\mathcal{F}$ gives the response i to some process. ◀

Thus the outputs of $\mathcal{P}[1..m]$ must satisfy the requirements of the (m, ℓ)-BD task. Let $\texttt{Output}(\mathcal{P})$ be the multiset of all outputs from $\mathcal{P}[1..m]$ so far, and let $\texttt{Output}_p(\mathcal{P})$ be the multiset of all outputs from $\mathcal{P}[1..m]$ given to process p so far. To begin the analysis of this algorithm, first observe that the following invariant always holds:

▶ **Invariant 3.** *For each process p,*
(a) *$\mathcal{X}[p] \subseteq \texttt{Output}_p(\mathcal{P})$, and hence $\mathcal{X} \subseteq \texttt{Output}(\mathcal{P})$.*
(b) *$\mathcal{X}[p]$, and hence $\mathcal{X}$, is monotonically increasing.*

▶ **Claim 4** (Termination)**.** *Every correct process eventually decides exactly one value.*

Proof. Let p be a correct process. Since $next_p$ is derived from repeated operations on the Fetch&Increment object $\mathcal{F}$, it will eventually become greater than m. Then the conditional on line 3 will evaluate to false, causing p to decide exactly one value and halt (line 8). ◀

▶ **Claim 5** (Validity). *Every decision v was proposed by some process, i.e., $v = v_q$ for some process q.*

Proof. Suppose a process p decides a value v. Clearly, $v \in \mathcal{X}$. By Invariant 3, $\mathcal{X}$ is a subset of $\texttt{Output}(\mathcal{P})$, so v is an output from $\mathcal{P}[1..m]$. Since $\mathcal{P}[1..m]$ is a solution for the (m, ℓ)-BD task, by the Validity property of this task, v must be one of the input values for $\mathcal{P}[1..m]$. The claim follows from the fact that each process q only uses v_q as the input value for $\mathcal{P}[1..m]$. ◀

▶ **Claim 6.** *Consider any snapshot of $\mathcal{X}$ that contains at least $h \geq \ell$ elements. The mode of this snapshot appears at least $h - \ell$ times in $\texttt{Output}(\mathcal{P})$.*

Proof. Since $\mathcal{P}[1..m]$ is a solution for the (m, ℓ)-BD task, by the ℓ-BD property of this task, there is a value v such that at most ℓ of the outputs in $\texttt{Output}(\mathcal{P})$ are not v. By Invariant 3, every snapshot of $\mathcal{X}$ is a subset of $\texttt{Output}(\mathcal{P})$. Consequently, each snapshot of $\mathcal{X}$ contains at most ℓ outputs that are not v, and the rest of the outputs must equal v. In other words, if a snapshot of $\mathcal{X}$ contains at least h elements, then v appears at least $h - \ell$ times in this snapshot. Thus the mode of the snapshot of $\mathcal{X}$ appears at least $h - \ell$ times in the snapshot. By Invariant 3, the mode of the snapshot also appears at least $h - \ell$ times in $\texttt{Output}(\mathcal{P})$. ◀

▶ **Claim 7.** *If a process decides a value v, then v appears at least $\lfloor \ell/k \rfloor + 1$ times in $\texttt{Output}(\mathcal{P})$.*

Proof. Suppose a process p decides a value v. Then, by lines 8 and 7, v is the mode of a snapshot S of $\mathcal{X}$, and by the conditional on line 3, the snapshot S is taken after p get a value greater than m from $\mathcal{F}$. Since $\mathcal{F}$ is initialized such that the first operation on $\mathcal{F}$ gets a response of 1, at least $m + 1$ operations were performed on $\mathcal{F}$ before the snapshot S is taken.

Note that each time a process receives a response from $\mathcal{F}$ that is at most m, it cannot perform another operation on $\mathcal{F}$ before executing a protocol of $\mathcal{P}[1..m]$ and adding the output of that execution to the multiset stored by $\mathcal{X}$. Thus, since the n processes perform a total of at least $m + 1$ operations on $\mathcal{F}$ before the snapshot S is taken, at least $m - n + 1$ outputs were added to the multiset stored by $\mathcal{X}$ before the snapshot S is taken. Thus S contains at least $m - n + 1$ outputs.

Now, recall that $m \geq n + \ell + \lfloor \ell/k \rfloor$, so $m - n + 1 \geq \ell + \lfloor \ell/k \rfloor + 1$. Thus the snapshot S of $\mathcal{X}$ contains at least $\ell + \lfloor \ell/k \rfloor + 1$ outputs. By Claim 6, the mode v of S appears at least $\lfloor \ell/k \rfloor + 1$ times in $\texttt{Output}(\mathcal{P})$. ◀

▶ **Claim 8** (k-SA). *There are at most k distinct decision values.*

Proof. Assume, for contradiction, that there are at least $k + 1$ distinct decision values. By Claim 7, each decision value appears at least $\lfloor \ell/k \rfloor + 1$ times in $\texttt{Output}(\mathcal{P})$. In other words, $\texttt{Output}(\mathcal{P})$ contains at least $k+1$ distinct output values, each appearing at least $\lfloor \ell/k \rfloor + 1$ times. This implies that for *every* value v, at least $k(\lfloor \ell/k \rfloor + 1) > \ell$ of the outputs given by $\mathcal{P}[1..m]$ are not v. This violates the ℓ-BD property of the (m, ℓ)-BD task, contradicting the fact that $\mathcal{P}[1..m]$ solves the (m, ℓ)-BD task. ◀

Thus the Termination (Claim 4), Validity (Claim 5), and k-SA (Claim 8) properties of the (n, k)-SA task all hold. So Algorithm 1 solves the (n, k)-SA task. ◀

▶ **Theorem 9.** *For all $\ell \geq 0$ and $m \geq 2\ell + 2$, the (m, ℓ)-BD task and registers can solve the 2-consensus task.*

Algorithm 2 Solving consensus among 2 processes using an (m, ℓ)-BD task solution $\mathcal{P}[1..m]$ and an atomic snapshot $\mathcal{X}$.

```
1:  while TRUE do
2:      snap ← X.SNAPSHOT()
3:      if |snap| ≤ m − 2 then
4:          if p = 1 then
5:              received ← received + {P[|received| + 1].execute(v_p)}
6:          else
7:              received ← received + {P[m − |received|].execute(v_p)}
8:          X[p].WRITE(received)
9:      else
10:         decide mode(snap)
11:         halt
```

Proof. Consider any $\ell \geq 0$ and $m \geq 2\ell + 2$. We present a modification of Algorithm 1 that solves the 2-consensus task without using a Fetch&Increment object. In addition to a solution $\mathcal{P}[1..m]$ for the (m, ℓ)-BD task, the new algorithm uses only a (shared) 2-process snapshot object $\mathcal{X}$, and local variables *received* and *snap* as described in Algorithm 1.

Recall that the Fetch&Increment object $\mathcal{F}$ used in Algorithm 1 serves two roles:

- It ensures each protocol of $\mathcal{P}[1..m]$ is executed at most once.
- It tells processes when to take a snapshot of $\mathcal{X}$ and decide the mode.

Roughly speaking, processes can decide as soon as sufficiently many outputs have been written into $\mathcal{X}$. Thus, the second role can be replaced simply by having processes take a snapshot of $\mathcal{X}$ and counting the current number of outputs. For the first role, we can ensure that each protocol of $\mathcal{P}[1..m]$ is executed at most once by using the fact that there are only two processes involved in the 2-consensus task. Intuitively, this is achieved by starting the two processes at opposite ends of the range $[1..m]$, and having them sequentially execute the protocols of $\mathcal{P}[1..m]$ until they meet each other.

As before, let v_p denote p's input value. Let the two processes be numbered 1 and 2. We now describe the algorithm that each process p executes to solve the 2-consensus task:

1. It takes a snapshot S of $\mathcal{X}$.
2. If S contains more than $m - 2$ outputs, it decides the mode.
3. If p is process 1, it executes protocol $\mathcal{P}[i_1 + 1]$ with input value v_p, where i_1 is the number of protocols process 1 has executed.
4. If p is process 2, it executes protocol $\mathcal{P}[m - i_2]$ with input value v_p, where i_2 is the number of protocols process 2 has executed.
5. It adds the output to $\mathcal{X}[p]$.
6. It repeats from the first step.

The pseudocode for each process p in shown in Algorithm 2.

▶ **Claim 10.** *Each protocol of $\mathcal{P}[1..m]$ is executed at most once.*

Proof. Assume for contradiction, that for some $1 \leq i \leq m$, protocol $\mathcal{P}[i]$ is executed more than once. Observe that in Algorithm 2, this can only happen if both processes execute protocol $\mathcal{P}[i]$, since each process sequentially traverses the range $[1..m]$. This means both processes took a snapshot of $\mathcal{X}$ and found at most $m - 2$ outputs right before executing protocol $\mathcal{P}[i]$. However, in order for both processes to reach protocol $\mathcal{P}[i]$, they must first

execute all other protocols in $\mathcal{P}[1..m]$ and write their outputs into $\mathcal{X}$. Hence one of the two processes must have found more than $m-2$ outputs in its snapshot, a contradiction. ◀

Since each protocol is executed at most once, their outputs must satisfy the requirements of the (m,ℓ)-BD task. As before, let $\texttt{Output}(\mathcal{P})$ be the multiset of all outputs from $\mathcal{P}[1..m]$ so far, and let $\texttt{Output}_p(\mathcal{P})$ be the multiset of all outputs from $\mathcal{P}[1..m]$ given to process p so far. First, note that Invariant 3 still holds for this modified algorithm. Furthermore, Claim 5 and Claim 6 also hold here: their proofs are exactly as before. The remaining claims also have similar proofs:

▶ **Claim 11** (Termination). *Every correct process eventually decides exactly one value.*

Proof. Let p be a correct process. Observe that each time the conditional on line 3 evaluates to true, process p will execute a protocol of $\mathcal{P}[1..m]$, and write its output to $\mathcal{X}$, increasing the number of outputs in $\mathcal{X}$ by one. By Invariant 3, the number of outputs in $\mathcal{X}$ is monotonically increasing. Thus it will eventually become greater than $m-2$. Then the conditional on line 3 will evaluate to false, causing p to decide exactly one value and halt (lines 10). ◀

▶ **Claim 12.** *If a process decides a value v, then v appears at least $\ell+1$ times in $\texttt{Output}(\mathcal{P})$.*

Proof. Suppose a process p decides a value v. Then, by lines 10 and 2, v is the mode of a snapshot S of $\mathcal{X}$, and by the conditional on line 3, the snapshot S contains at least $m-1$ outputs. Now, recall that $m \geq 2\ell+2$, so $m-1 \geq 2\ell+1$. Thus the snapshot S of $\mathcal{X}$ contains at least $2\ell+1$ outputs. By Claim 6, the mode v of S appears at least $\ell+1$ times in $\texttt{Output}(\mathcal{P})$. ◀

▶ **Claim 13** (Agreement). *All decisions have the same value.*

Proof. Assume, for contradiction, that there are at least 2 distinct decision values. By Claim 12, each decision value appears at least $\ell+1$ times in $\texttt{Output}(\mathcal{P})$. Thus $\texttt{Output}(\mathcal{P})$ contains at least 2 distinct output values, each appearing at least $\ell+1$ times. This implies that for *every* value v, at least $\ell+1 > \ell$ of the outputs given by $\mathcal{P}[1..m]$ are not v. This violates the ℓ-BD property of the (m,ℓ)-BD task, contradicting the fact that $\mathcal{P}[1..m]$ solves the (m,ℓ)-BD task. ◀

Thus the Termination (Claim 11), Validity (Claim 5), and Agreement (Claim 13) properties of the 2-consensus task all hold. So Algorithm 2 solves the 2-consensus task. ◀

We can now prove the main result of this section:

▶ **Theorem 14.** *For all $k \geq 1$, $n \geq 1$, $\ell \geq 0$ and $m \geq \max(n+\ell+\lfloor \ell/k \rfloor, 2\ell+2)$, the (m,ℓ)-BD task and registers can solve the (n,k)-SA task.*

Proof. In previous work, Afek *et al.* [5] proved that (any solution to) the 2-consensus task, together with registers, can be used to implement a Fetch&Increment object for any number of processes. Thus from Theorem 9, for all $\ell \geq 0$ and $m \geq 2\ell+2$, the (m,ℓ)-BD task and registers can implement a Fetch&Increment object for any number of processes. The result now follows from Theorem 1. ◀

▶ **Corollary 15.** *For all $\ell \geq 0$ and $m > 2\ell$, the (m,ℓ)-BD task and registers can solve the $(m-2\ell)$-consensus task.*

Proof. Let $\ell \geq 0$. For $m = 2\ell+1$, the $(m-2\ell)$-consensus task is the trivial 1-consensus task. For $m \geq 2\ell+2$, by setting $k=1$ and $n = m-2\ell$ in Theorem 14, we get that the (m,ℓ)-BD task and registers can solve the $(m-2\ell,1)$-SA task, i.e., the $(m-2\ell)$-consensus task. ◀

3 BD Objects

Intuitively, with an (m, ℓ)-BD *object*, processes can propose any value to the object, and the object responds to the first m such proposals with values that satisfy the Validity and ℓ-BD properties; the $(m+1)$-th proposal permanently *upsets* the (m, ℓ)-BD object, causing it to nondeterministically respond with an arbitrary value to this proposal and all the subsequent ones. Thus, as long as the BD object is not upset, every response is one of the proposals (Validity) and satisfies the following property:

- **ℓ-BD**: There exists a value v such that at most ℓ responses are not v.

Note that m processes can trivially use a single (m, ℓ)-BD object to solve the (m, ℓ)-BD task: each process just proposes its input value to the object and decides the object's response. However, since the (m, ℓ)-BD object gets upset if more than m proposal operations are applied, it cannot be used in this trivial way by $m' > m$ processes to solve the (m', ℓ)-BD task.

We now give a more precise specification of the (m, ℓ)-BD object. A linearizable shared memory object is specified by a tuple $(OP, RES, Q, s_0, \delta)$, where OP is a set of operations, RES is a set of response values, Q is a set of states, $s_0 \in Q$ is the initial state, and $\delta \subseteq Q \times OP \times Q \times RES$ is a state transition relation such that:

- A tuple $(s, op, s', res) \in \delta$ means that if the object is in state s when operation $op \in OP$ is invoked, then the object can change its state to s' and return res as the response.
- δ is total: For every reachable state $s \in Q$ and every operation $op \in OP$, there exists at least one $s' \in Q$ and $res \in RES$ such that $(s, op, s', res) \in \delta$.

The state of an (m, ℓ)-BD object $\mathcal{D}$ is either a pair (S_{OP}, M_{RES}), where S_{OP} is the *set* of all the values proposed to $\mathcal{D}$ and M_{RES} is the *multiset* of all responses given by $\mathcal{D}$, or the upset state $\perp$. Note that the state of $\mathcal{D}$ does *not* record information about the *order* of operations and responses. If the state of $\mathcal{D}$ is a pair (S_{OP}, M_{RES}), we denote by $|M_{RES}|$ the size of M_{RES}; note that $|M_{RES}|$ is also the number of operations performed on $\mathcal{D}$ so far. We say that the state (S_{OP}, M_{RES}) is *full* if $|M_{RES}| = m$, since the next operation performed on $\mathcal{D}$ will cause $\mathcal{D}$ to permanently enter the upset state $\perp$.

For simplicity, we define the set of operations to be $\mathbb{Z}$ (the set of integers), where operation $i \in \mathbb{Z}$ represents proposing the value i. So for all $\ell \geq 1$ and $m > \ell$, we define the (m, ℓ)-BD object as the tuple $(OP, RES, Q, s_0, \delta)$ such that:

- $OP = \mathbb{Z}$.
- $RES = \mathbb{Z}$.
- The set of states Q consists of:
  - Every pair (S_{OP}, M_{RES}), where S_{OP} is a set of integers and M_{RES} is a multiset of integers such that $|M_{RES}| \leq m$. If $|M_{RES}| = m$, the state is called *full*.
  - The upset state $\perp$.
- $s_0 = (\varnothing, \varnothing)$.
- The state transition relation δ contains an element (s, op, s', res) if and only if one of the following two conditions holds:
  - s and s' are states (S_{OP}, M_{RES}) and (S'_{OP}, M'_{RES}) such that all of the following hold:
    - s is not full.
    - $S'_{OP} = S_{OP} \cup \{op\}$.
    - $M'_{RES} = M_{RES} + \{res\}$.
    - **Validity:** Each element in M'_{RES} is in S'_{OP}.
    - **ℓ-BD:** There exists a value v such that at most ℓ elements in M'_{RES} are not v.
  - s is either a full state or the upset state $\perp$, and s' is the upset state $\perp$.

From these definitions, we can make a few observations. First, the validity property implies:

▶ **Observation 16.** *The response to the first operation performed on a BD object is the operation's own proposal value.*

Next, the upset state $\perp$ works as intended:

▶ **Observation 17.** *A BD object that is in either a full state or the upset state $\perp$ can nondeterministically respond with any value to all future operations.*

Furthermore, the validity and ℓ-BD properties hold as long as the object is not upset:

▶ **Observation 18.** *If at most m operations are applied to an (m, ℓ)-BD object $\mathcal{D}$, then each response given by $\mathcal{D}$ is a value proposed to $\mathcal{D}$, and there is a value v such that at most ℓ of the responses given by $\mathcal{D}$ are not v.*

This immediately implies:

▶ **Observation 19.** *The (m, ℓ)-BD task can be solved with a single (m, ℓ)-BD object.*

From the above observation, it is clear that Theorem 14 and Corollary 15 also apply for BD objects.

4 Unsolvability of $(m - 2\ell + 1)$-Consensus by (m, ℓ)-BD Objects

We now show that (m, ℓ)-BD objects and registers *cannot* solve the $(m - 2\ell + 1)$-consensus task. This, together with Corollary 15, allows us to determine the consensus number of every instance of the BD task and BD object.

▶ **Theorem 20.** *For all $\ell \geq 0$ and $m > 2\ell$, (m, ℓ)-BD objects and registers cannot solve the $(m - 2\ell + 1)$-consensus task.*

Proof. Consider any $\ell \geq 0$ and $m > 2\ell$. For the special case of $\ell = 0$, observe that the (m, ℓ)-BD object is equivalent to an m-consensus object. Thus the theorem immediately follows from the fact that m-consensus objects and registers cannot solve the $(m+1)$-consensus task. Hence it suffices to consider the case where $\ell \geq 1$.

For $\ell \geq 1$, we prove a stronger result, namely, that (m, ℓ)-BD objects and registers cannot solve the *binary*[5] $(m - 2\ell + 1)$-consensus task, even when the (m, ℓ)-BD objects are further strengthened by restricting their nondeterministic behavior as follows: As long as it is not upset, each (m, ℓ)-BD object outputs at most two distinct response values.

Assume, for contradiction, that there is an $\ell \geq 1$, an $m > 2\ell$ and a wait-free algorithm that solves binary consensus among $m - 2\ell + 1 \geq 2$ processes using only registers and the strengthened (m, ℓ)-BD objects. It suffices to prove the existence of an infinite execution of this algorithm where processes never decide.

The proof uses the bivalency technique introduced by Fischer *et al.* [12]. We assume the reader is familiar with bivalency proof terminology such as *configuration*, *bivalent*, and 0-*valent* [12, 16]. In the following, we omit the proofs of Claims 21 to 24 since they are standard (see [16]).

▶ **Claim 21.** *The algorithm has an initial bivalent configuration C_{init}.*

[5] In the binary consensus task, every proposal value is restricted to being either 0 or 1.

▶ **Claim 22.** *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 23.** *At C_{bi}, every process is about to perform an operation on the same object $\mathcal{D}$.*

▶ **Claim 24.** *$\mathcal{D}$ is not a register.*

Since the algorithm only uses registers and (m, ℓ)-BD objects,

▶ **Claim 25.** *$\mathcal{D}$ is an (m, ℓ)-BD object.*

▶ **Claim 26.** *There exists a pair of distinct processes p_0 and p_1, and a pair of (not necessarily distinct) values v_0 and v_1, such that starting from C_{bi},*

- *p_0 can take a step with response v_0, and the resulting configuration C_0 is 0-valent.*
- *p_1 can take a step with response v_1, and the resulting configuration C_1 is 1-valent.*

Proof. Follows immediately from Claim 22 since there are at least 2 processes. ◀

Given a configuration C such that $\mathcal{D}$ is not upset in C, we denote by $\mathcal{D}.M_{RES}(C)$ the multiset M_{RES} in the state of $\mathcal{D}$ in C.

▶ **Claim 27.** *$\mathcal{D}$ is not upset in C_{bi} and $|\mathcal{D}.M_{RES}(C_{bi})| < 2\ell$.*

Proof. Assume, for contradiction, that either $\mathcal{D}$ is upset in C_{bi} or $|\mathcal{D}.M_{RES}(C_{bi})| \geq 2\ell$. By Claim 23, every process was about to apply an operation on $\mathcal{D}$ at C_{bi}, and so at both C_0 and C_1, every process other than p_0 and p_1 will perform an operation on $\mathcal{D}$ as its next step. Thus consider the following configurations:

- A 0-valent configuration C_0' reached from C_0 by letting every process other than p_0 and p_1 (if any) take a step in an arbitrary order (while p_0 and p_1 take no steps).
- A 1-valent configuration C_1' reached from C_1 by letting every process other than p_0 and p_1 (if any) take a step in an arbitrary order (while p_0 and p_1 take no steps).

There are two cases:

1. $\mathcal{D}$ is upset in C_{bi}. Clearly, $\mathcal{D}$ is also upset in both C_0' and C_1'.
2. $|\mathcal{D}.M_{RES}(C_{bi})| \geq 2\ell$. Since there are $m - 2\ell + 1$ processes, by Claim 23, in both C_0' and C_1', the number of operations performed on $\mathcal{D}$ is $|\mathcal{D}.M_{RES}(C_{bi})| + (m - 2\ell + 1) - 1 \geq m$. Thus in both C_0' and C_1', $\mathcal{D}$ is in either a full state or the upset state $\perp$.

So in all cases, $\mathcal{D}$ is in either a full state or the upset state $\perp$, in both C_0' and C_1'. By Observation 17, $\mathcal{D}$ is henceforth allowed to nondeterministically return any response to all subsequent operations. Thus consider the following configurations:

- The 0-valent configuration C_0'' reached from C_0' by letting p_1 take a step with response v_1.
- The 1-valent configuration C_1'' reached from C_1' by letting p_0 take a step with response v_0.

Since p_0 received v_0 as the response from $\mathcal{D}$ in both C_0'' and C_1'', the state of p_0 is the same in C_0'' as in C_1''. Furthermore, the state of every object is the same in C_0'' as in C_1'': $\mathcal{D}$ is in the upset state $\perp$, and all other objects have not changed their state since C_{bi}. Consequently, if all processes other than p_0 crash, p_0 will not be able to distinguish between the 0-valent configuration C_0'' and the 1-valent configuration C_1'', so if p_0 runs solo starting from configurations C_0'' and C_1'', it decides the same value in both runs – a contradiction. ◀

▶ **Claim 28.** $|\mathcal{D}.M_{RES}(C_{bi})| \geq 1$.

Proof. Assume, for contradiction, that $|\mathcal{D}.M_{RES}(C_{bi})| = 0$. In other words, $\mathcal{D}$ is in the initial state in C_{bi}. By Observation 16, v_0 is the proposal value of the operation that p_0 is about to invoke on $\mathcal{D}$ at configuration C_{bi}, and similarly v_1 is the proposal value of the operation that p_1 is about to invoke on $\mathcal{D}$ at configuration C_{bi}. We claim that:

- Starting from C_0, p_1 can take a step with response v_1, and the resulting configuration C'_0 is 0-valent.
- Starting from C_1, p_0 can take a step with response v_0, and the resulting configuration C'_1 is 1-valent.

To see why, recall that $\mathcal{D}$ is a strengthened (m, ℓ)-BD object where $\ell \geq 1$. In both C'_0 and C'_1, the number of distinct response values from $\mathcal{D}$ is at most two, and the number of responses different from v_0 is at most 1. Thus the above steps required to reach C'_0 and C'_1 are possible.

Finally, recall that the state of $\mathcal{D}$ does not record information about the order of operations and responses: it records only the set of proposals and the multiset of responses so far. Consequently, the state of $\mathcal{D}$ is the same in C'_0 as in C'_1. Furthermore, since p_0 received v_0 as the response from $\mathcal{D}$ in both C'_0 and C'_1, the state of p_0 is the same in C'_0 as in C'_1. Similarly, the state of p_1 is the same in C'_0 as in C'_1. Finally, observe that all other objects and processes have not changed their state since C_{bi}. Thus their states are also the same in C'_0 as in C'_1. Therefore $C'_0 = C'_1$, contradicting the fact that C'_0 is 0-valent and C'_1 is 1-valent. ◀

By Claim 27 and Claim 28, $\mathcal{D}$ is not upset in C_{bi} and $1 \leq |\mathcal{D}.M_{RES}(C_{bi})| < 2\ell$. Let v_{mode} denote the mode of $\mathcal{D}.M_{RES}(C_{bi})$ (ties can be decided arbitrarily).

▶ **Claim 29.** *At most $\ell - 1$ of the responses in $\mathcal{D}.M_{RES}(C_{bi})$ are not v_{mode}.*

Proof. By Claim 27, $\mathcal{D}$ is not upset at C_{bi} and $\mathcal{D}.M_{RES}(C_{bi})$ contains at most $2\ell-1$ responses. Recall that $\mathcal{D}$ is a strengthened (m, ℓ)-BD object that outputs at most two distinct values as long as it is not upset. Thus $\mathcal{D}.M_{RES}(C_{bi})$ contains at most two distinct values. So the mode v_{mode} of $\mathcal{D}.M_{RES}(C_{bi})$ appears at least $\lceil |\mathcal{D}.M_{RES}(C_{bi})|/2 \rceil$ times. Thus, the number of responses in $\mathcal{D}.M_{RES}(C_{bi})$ that are *not* v_{mode} is at most $\lfloor |\mathcal{D}.M_{RES}(C_{bi})|/2 \rfloor \leq \lfloor (2\ell - 1)/2 \rfloor = \ell - 1$. ◀

▶ **Claim 30.** *There exists a pair of distinct processes p_0^* and p_1^*, and a value v_1^* (not necessarily distinct from v_{mode}), such that starting from C_{bi},*

- *p_0^* can take a step with response v_{mode}, and the resulting configuration C_0^* is univalent.*
- *p_1^* can take a step with response v_1^*, and the resulting configuration C_1^* is univalent, with different valence from C_0^*.*

Proof. Consider the processes p_0 and p_1, the values v_0 and v_1, and the configurations C_0 and C_1 stated in Claim 26. If $v_0 = v_{mode}$ or $v_1 = v_{mode}$, we are done.

Suppose $v_0 \neq v_{mode}$ and $v_1 \neq v_{mode}$. Let C'_0 be the configuration reached when, starting from C_{bi}, p_0 takes a step with response v_{mode} (note that this step is possible, since the number of distinct response values, and the number of responses different from v_{mode}, do not increase). Similarly, let C'_1 be the configuration reached when, starting from C_{bi}, p_1 takes a step with response v_{mode}. If C'_0 and C'_1 have different valence, we are done.

Suppose C'_0 and C'_1 have the same valence. Without loss of generality, suppose they are both 0-valent. Observe that, starting from C_{bi}, the 0-valent configuration C'_0 is reached when p_0 takes a step with response v_{mode}, and the 1-valent configuration C_1 is reached when p_1 takes a step with response v_1. ◀

We claim that:

- Starting from C_0^*, p_1^* can take a step with response v_1^*, and the resulting configuration $\widehat{C}_0^*$ is univalent.
- Starting from C_1^*, p_0^* can take a step with response v_{mode}, and the resulting configuration $\widehat{C}_1^*$ is univalent, with different valence from $\widehat{C}_0^*$.

To see why, note that in both $\widehat{C}_0^*$ and $\widehat{C}_1^*$, the number of distinct response values from $\mathcal{D}$ is at most two, and from Claim 29, the number of responses different from v_{mode} is at most $(\ell - 1) + 1 = \ell$. Thus the above steps required to reach $\widehat{C}_0^*$ and $\widehat{C}_1^*$ are possible.

Recall that the state of $\mathcal{D}$ does not record information about the order of operations and responses: it records only the set of proposals and the multiset of responses so far. Consequently, the state of $\mathcal{D}$ is the same in $\widehat{C}_0^*$ as in $\widehat{C}_1^*$. Furthermore, since p_0^* received v_{mode} as the response from $\mathcal{D}$ in both $\widehat{C}_0^*$ and $\widehat{C}_1^*$, the state of p_0^* is the same in $\widehat{C}_0^*$ as in $\widehat{C}_1^*$. Similarly, the state of p_1^* is the same in $\widehat{C}_0^*$ as in $\widehat{C}_1^*$. Finally, observe that all other objects and processes have not changed their state since C_{bi}. Thus their states are also the same in $\widehat{C}_0^*$ as in $\widehat{C}_1^*$. Therefore $\widehat{C}_0^* = \widehat{C}_1^*$, contradicting the fact that $\widehat{C}_0^*$ and $\widehat{C}_1^*$ have opposite valence. ◀

We can now determine the consensus number for every BD task and object as follows:

▶ **Theorem 31.** *For all $\ell \geq 0$ and $m > 0$, both the (m, ℓ)-BD task and the (m, ℓ)-BD object have consensus number $\max(m - 2\ell, 1)$.*

Proof. Consider any $\ell \geq 0$ and $m > 0$. There are two cases: either $m > 2\ell$, or $0 \leq m \leq 2\ell$.

First, consider the case where $m > 2\ell$. Since BD objects are at least as strong as their corresponding BD tasks (Observation 19), by Corollary 15 both the (m, ℓ)-BD task and the (m, ℓ)-BD object can, together with registers, solve the $(m - 2\ell)$-consensus task. Similarly, by Theorem 20 both the (m, ℓ)-BD task and the (m, ℓ)-BD object, together with registers, cannot solve the $(m - 2\ell + 1)$-consensus task. Thus both the (m, ℓ)-BD task and the (m, ℓ)-BD object have consensus number $m - 2\ell = \max(m - 2\ell, 1)$.

Now consider the case where $m \leq 2\ell$. From the above case, we have that both the $(2\ell + 1, \ell)$-BD task and the $(2\ell + 1, \ell)$-BD object have consensus number 1. For $m \leq 2\ell$, it is clear that the $(2\ell+1, \ell)$-BD object is at least as strong as the (m, ℓ)-BD object, and the $(2\ell+1, \ell)$-BD task is at least as difficult to solve as the (m, ℓ)-BD task. Thus both the (m, ℓ)-BD task and the (m, ℓ)-BD object also have consensus number $1 = \max(m - 2\ell, 1)$. ◀

5 An Unusual Property of Bounded Disagreement

We now show that every level $n \geq 2$ of Herlihy's consensus hierarchy contains a BD object that cannot be implemented using n-consensus and registers.

▶ **Theorem 32.** *For all $n \geq 2$, both the $(9n, 4n)$-BD task and the $(9n, 4n)$-BD object have consensus number n, but cannot be implemented using n-consensus objects and registers.*

Proof. Let $n \geq 2$, and consider the (m, ℓ)-BD object for $m = 9n$ and $\ell = 4n$.

1. Since $\ell \geq 0$ and $m > 0$, by Theorem 31, the $(9n, 4n)$-BD object has consensus number $\max(m - 2\ell, 1) = 9n - 2(4n) = n$.
2. Consider the (n', k)-SA task for $n' = 3n$ and $k = 2$. Note that $m = n' + \ell + \lfloor \ell/k \rfloor$ (because $9n = 3n + 4n + \lfloor 4n/2 \rfloor$), and $m \geq 2\ell + 2$ (because for $n \geq 2$, $9n \geq 2(4n) + 2$). Then, since $k \geq 1$, $n' \geq 1$, $\ell \geq 0$, and $m \geq \max(n' + \ell + \lfloor \ell/k \rfloor, 2\ell + 2)$, by Theorem 14 and the fact that the $(9n, 4n)$-BD object is at least as strong as the $(9n, 4n)$-BD task (Observation 19), $(9n, 4n)$-BD objects and registers can solve the $(3n, 2)$-SA task.

We claim that the $(9n, 4n)$-BD object cannot be implemented using n-consensus objects and registers. Suppose, for contradiction, that n-consensus objects and registers can implement the $(9n, 4n)$-BD object. Since $(9n, 4n)$-BD objects and registers can solve the $(3n, 2)$-SA task, this implies that n-consensus objects and registers can solve the $(3n, 2)$-SA task. This contradicts the following important result about set agreement: for all $n \geq 2$, n-consensus objects and registers cannot solve the $(3n, 2)$-SA task [8].

Thus, the $(9n, 4n)$-BD object has consensus number n, but cannot be implemented using n-consensus objects and registers. The proof that this also holds for the $(9n, 4n)$-BD *task* is almost identical. ◀

Prior to this paper, the only known objects with consensus number $n \geq 2$ and not implementable using n-consensus and registers were the ad hoc objects defined by Rachman [18] and Afek *et al.* [2]. In the appendix, we prove that these objects are fundamentally different from our BD objects.

6 Conclusion

In this paper, we introduced a novel and natural generalization of consensus that we call bounded disagreement, and investigated its relation to consensus and set agreement. Our results show that, despite apparent similarities, bounded disagreement is fundamentally different from these two problems:

- *BD and consensus are distinct problems.* In fact, by Theorem 32, for all $n \geq 2$, the $(9n, 4n)$-BD task [object] have consensus number n, but cannot be solved [implemented] by n-consensus and registers. Thus for all $k \leq n$, k-consensus and registers cannot solve the $(9n, 4n)$-BD task, and for all $k > n$, the $(9n, 4n)$-BD task and registers cannot solve k-consensus: so, for all $n \geq 2$, there is no k such that k-consensus is equivalent to the $(9n, 4n)$-BD task. Therefore there are infinitely many instances of the bounded disagreement task that are not equivalent to any consensus task.
- *BD and SA are distinct problems.* Except for the special case of $k = 1$ where the (n, k)-SA task is simply the n-consensus task, the (n, k)-SA task and registers cannot solve the 2-consensus task [8]. In contrast, by Theorem 31, the $(4, 1)$-BD task has consensus number 2, and in fact, for all $n \geq 2$, there are $\ell \geq 1$ and m such that the (m, ℓ)-BD task has consensus number n. So there are infinitely many instances of the bounded disagreement task that are not equivalent to any set agreement task.

Consensus is one of the fundamental problems in distributed computing, and its generalizations tend to produce deep and valuable insights about the field as a whole. The history of set agreement is one such example: since its introduction in 1990 [7], research on this problem has led to significant results, such as linking topology and distributed computing [17, 19]. It has also elucidated the relationship between consensus and other important problems in distributed computing, such as renaming and symmetry breaking [6, 13, 14]. The bounded disagreement problem may provide a similar impetus in the future.

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 Yehuda Afek, Eytan Weisberger, and Hanan Weisman. A completeness theorem for a class of synchronization objects. In *Proceedings of the Twelfth Annual ACM Symposium on Principles of Distributed Computing*, PODC'93, pages 159–170, New York, NY, USA, 1993. ACM. doi:10.1145/164051.164071.

6 Armando Castañeda, Damien Imbs, Sergio Rajsbaum, and Michel Raynal. Renaming is weaker than set agreement but for perfect renaming: A map of sub-consensus tasks. In *Proceedings of the 10th Latin American International Conference on Theoretical Informatics*, LATIN'12, pages 145–156, Berlin, Heidelberg, 2012. Springer-Verlag. doi:10.1007/978-3-642-29344-3_13.

7 Soma Chaudhuri. Agreement is harder than consensus: Set consensus problems in totally asynchronous systems. In *Proceedings of the Ninth Annual ACM Symposium on Principles of Distributed Computing*, PODC'90, pages 311–324, New York, NY, USA, 1990. ACM. doi:10.1145/93385.93431.

8 Soma Chaudhuri and Paul Reiners. Understanding the set consensus partial order using the borowsky-gafni simulation (extended abstract). In *Proceedings of the 10th International Workshop on Distributed Algorithms*, WDAG'96, pages 362–379, London, UK, UK, 1996. Springer-Verlag. URL: http://dl.acm.org/citation.cfm?id=645953.675630.

9 Matei David. A single-enqueuer wait-free queue implementation. In Rachid Guerraoui, editor, *Distributed Computing*, volume 3274 of *Lecture Notes in Computer Science*, pages 132–143. Springer Berlin Heidelberg, 2004. doi:10.1007/978-3-540-30186-8_10.

10 Matei David, Alex Brodsky, and Faith Ellen Fich. Restricted stack implementations. In *Proceedings of the 19th International Conference on Distributed Computing*, DISC'05, pages 137–151, Berlin, Heidelberg, 2005. Springer-Verlag. doi:10.1007/11561927_12.

11 David Eisenstat. Two-enqueuer queue in common2. *CoRR*, abs/0805.0444, 2008. URL: http://arxiv.org/abs/0805.0444.

12 Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382, Apr 1985. doi:10.1145/3149.214121.

13 Eli Gafni, Sergio Rajsbaum, and Maurice Herlihy. Subconsensus tasks: Renaming is weaker than set agreement. In Shlomi Dolev, editor, *Distributed Computing*, volume 4167 of *Lecture Notes in Computer Science*, pages 329–338. Springer Berlin Heidelberg, 2006. doi:10.1007/11864219_23.

14 Eli Gafni, Michel Raynal, and Corentin Travers. Test&set, adaptive renaming and set agreement: a guided visit to asynchronous computability. In *Reliable Distributed Systems, 2007. SRDS 2007. 26th IEEE International Symposium on*, pages 93–102, Oct 2007. doi:10.1109/SRDS.2007.8.

15 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.

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

17 Maurice Herlihy and Sergio Rajsbaum. Set consensus using arbitrary objects (preliminary version). In *Proceedings of the Thirteenth Annual ACM Symposium on Principles of Distributed Computing*, PODC'94, pages 324–333, New York, NY, USA, 1994. ACM. doi:10.1145/197917.198119.

18 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.

19 Michael Saks and Fotios Zaharoglou. Wait-free k-set agreement is impossible: The topology of public knowledge. *SIAM J. Comput.*, 29(5):1449–1483, Mar 2000. doi:10.1137/S0097539796307698.

A BD Objects Versus the Object Defined by Rachman in [18]

In [18], Rachman defined the 2-*set object with parameter* m, which is a composite of an $(\infty, 2)$-SA object and an m-consensus object. Rachman proved that this object has consensus number m but cannot be implemented by m-consensus objects and registers.

▶ **Theorem 33.** *For all $m \geq 1$, the 2-set object with parameter m cannot be implemented by (n, ℓ)-BD objects and registers for any $n \geq 1$ and $\ell \geq 0$.*

Proof. For all $n \geq 1$, n-consensus objects and registers cannot solve the $(\infty, 2)$-SA task [8]. Since the n-consensus object can trivially implement the (n, ℓ)-BD object for all $\ell \geq 0$, (n, ℓ)-BD objects and registers cannot solve the $(\infty, 2)$-SA task. In contrast, by definition, the 2-set object with parameter $m \geq 1$ solves the $(\infty, 2)$-SA task [18]. ◀

B BD Objects Versus the Object Defined by Afek *et al.* in [2]

In [2], Afek *et al.* defined a family of deterministic objects $O_{m,k}$, where $m \geq 2$ and $k \geq 2$, such that: (a) $O_{m,k}$ has consensus number m, (b) $O_{m,k}$ solves the $(km + k - 1, k)$-SA task, and (c) $O_{m,k+1}$ implements $O_{m,k}$. They also proved that $O_{m,k}$ objects and registers cannot solve the $(km + m + k, k + 1)$-SA task, which is trivially solved by the $O_{m,k+1}$ object since $km + m + k = (k + 1)m + (k + 1) - 1$.

▶ **Theorem 34.** *For all $n \geq 1$, the $(9n, 4n)$-BD object is not equivalent to the $O_{m,k}$ object, for any $m \geq 2$ and $k \geq 2$.*

Proof. Consider any $n \geq 1$, $m \geq 2$, and $k \geq 2$. By Theorem 31, the $(9n, 4n)$-BD object has consensus number n. On the other hand, the $O_{m,k}$ object has consensus number m [2]. If $m \neq n$, it is clear that the two objects are not equivalent. So suppose that $n = m$.

Case 1: k is odd. Thus there exists a positive integer q such that $k + 1 = 2q$. Observe that:

- Any solution to the $(2q(m+1), 2q)$-SA task can be used to solve the $(km+m+k, k+1)$-SA task; this is because $km + m + k = (k + 1)(m + 1) - 1 = 2q(m + 1) - 1 < 2q(m + 1)$.
- Any solution to the $(2(m+1), 2)$-SA task can be used, together with registers, to solve the $(2q(m + 1), 2q)$-SA task; the algorithm to do so is obvious: divide the $2q(m + 1)$ processes into q groups of $2(m + 1)$ processes, and have each group solve the $(2(m + 1), 2)$-SA task independently [8].

- Since $m \geq 2$, $2(m+1) \leq 3m$, and so any solution to the $(3m, 2)$-SA task can be used to solve the $(2(m+1), 2)$-SA task.
- Since $m = n$, the $(3m, 2)$-SA task is the $(3n, 2)$-SA task.
- By Theorem 14 and the fact that the $(9n, 4n)$-BD object is at least as strong as the $(9n, 4n)$-BD task (Observation 19), $(9n, 4n)$-BD objects and registers can solve the $(3n, 2)$-SA task.

Thus, by transitivity, we have that $(9n, 4n)$-BD objects and registers can solve the $(km + m + k, k + 1)$-SA task. In contrast, as we mentioned above, $O_{m,k}$ objects and registers cannot solve the $(km + m + k, k + 1)$-SA task [2]. We conclude that the $(9n, 4n)$-BD object cannot be implemented using $O_{m,k}$ objects and registers.

Case 2: k is even. Then $k + 1$ is odd. From case 1, the $(9n, 4n)$-BD object cannot be implemented using $O_{m,k+1}$ objects and registers. Since the $O_{m,k+1}$ object implements the $O_{m,k}$ object [2], by transitivity, the $(9n, 4n)$-BD object cannot be implemented using $O_{m,k}$ objects and registers. ◀