Resource Bounds and Combinations of Consensus Objects

Jon Kleinberg Sendhil Mullainathan

Abstract

The shared-memory model of computation typically provides processes with an arbitrary number of copies of the available object types; yet a simple argument shows that any consensus protocol can only make use of some finite subset of these. Thus we believe it is useful to consider the problem of consensus from the point of view of resource bounds, determining whether consensus can still be solved when the number of copies of the system's shared objects is limited. This approach leads to a general technique which we call the *combination protocol*, in which the number of processes that can achieve consensus with a given object increases as more copies of it are made available. Such a phenomenon brings up questions about the robustness of Herlihy's *consensus hierarchy*, in that objects are being combined to solve n-process consensus, even though no single copy can do so individually. We show how the ideas in the combination protocol appear even in situations where objects are not explicitly being combined with one another; we also consider the general question of resource bounds in several known consensus protocols. We analyze two such protocols that use seemingly similar primitives, achieving a substantial improvement in one case and showing a tight lower bound in the other.

1 Introduction

Any pessimistic model of distributed computing must take into account the perils of asynchrony. The traditional approaches to process coordination — semaphores and critical sections [CHP] — are based on the notion of a shared-memory system in which processes explicitly take turns modifying a globally accessible data structure (object). But processes on such a system are running asynchronously, so that the fast processes must wait for the slower ones to complete their operations. And if a process should crash while accessing a shared object, that object becomes unusable by the rest of the system.

A relatively recent development in this area has been the notion of "wait-free synchronization." One constructs objects in which each invocation is guaranteed to return in a finite number of steps of the invoking process, regardless of the actions (or failures) of the other processes; such an object is said to have a wait-free implementation. The natural question in this setting is the following: one has a system with a number of primitives that behave atomically (such as atomic registers, test-and-set registers, and the like) and wishes to know whether a given object type has a wait-free implementation in this system.

In [H1], Herlihy proposed an approach to this problem in which one classifies shared objects according to their *consensus number*. The consensus number of an object $\mathcal{O}$, as defined in [H1], is the maximum number of processes that can achieve asynchronous consensus using atomic operations on $\mathcal{O}$ and any number of atomic registers (recall that the results of [FLP, DDS, LAA] show that even two processes cannot achieve consensus using only registers). If any number of processes can achieve consensus using $\mathcal{O}$ and atomic registers, then $\mathcal{O}$ is said to have infinite consensus number. The resulting classification of shared objects is referred to as the *consensus hierarchy*. In [H1], it is shown that if the consensus number of $\mathcal{O}$ is k, the consensus number of $\mathcal{O}'$ is k', and $k > k'$, then $\mathcal{O}'$ cannot provide a wait-free implementation of $\mathcal{O}$ in a system of k or more processes.

We are interested the structure of this hierarchy, centering on the question of whether consensus number, as defined above, is truly a robust way of classifying shared objects. Specifically, we analyze the effect of combining shared objects — is it possible to combine objects having consensus number k, and obtain a system in which $k+1$ processes can solve consensus? And when the objects we combine are all of the same type, this becomes a question of resource bounds — how many copies of an object are needed for some fixed number of processes to achieve consensus? In order to carry out this analysis, we find it useful to make the model of [H1] somewhat more flexible. First of all, we will consider the "consensus number" of arbitrary (possibly infinite) collections of objects; i.e. we consider the maximum number of processes that can achieve consensus using *precisely* objects $\mathcal{O}_1, \ldots, \mathcal{O}_n$. Thus, we do not *a priori* assume that the system contains an arbitrary number of atomic registers; in this way, we make more explicit the resource requirements of our consensus protocols. With this variation of the model comes an additional subtlety: when the system contains an underlying infinite set of registers, certain natural forms of the consensus problem are reducible to others. With limited resources, however, this is no longer necessarily the case, and we identify two forms of consensus on which to concentrate: binary consensus and leader election (defined below). The definition of consensus number carries over in the obvious way to these problems, and we refer to the corresponding quantities as *bc*-number and *le*-number. We note that leader election, using unlimited registers, coincides with Herlihy's notion of consensus (and consensus number); thus, when we speak simply of consensus, we mean just this — leader election with registers.

Our main results center around a technique that we call the *combination protocol*. In the basic combination protocol, we consider an object $\mathcal{O}$ with *bc*-number and *le*-number equal to 2, and show that a collection of k copies of $\mathcal{O}$ has *le*-number at least $k+1$ (and *bc*-number at least $\lfloor \frac{k+1}{2} \rfloor$). Thus, although the consensus numbers of an individual copy of $\mathcal{O}$ are very low, an infinite set of them can solve consensus among an arbitrary set of processes. Hence, for example, even one copy of $\mathcal{O}$ cannot be implemented by any of the 2-consensus objects considered in [H1].

We believe the protocol itself represents an interesting technique for designing wait-free algorithms. It consists essentially of an "elimination tournament," which processes enter in a staggered fashion so that by the end, many processes are able safely to access a single copy of $\mathcal{O}$ simultaneously. Moreover, the view of objects as discrete units represents a departure from a number of previous approaches, in which all the objects in the system were in a sense "wired together." Our protocol, on the other hand, is modular; the $(n+1)$-process version of the protocol is obtained simply by adding an additional copy of the object and an additional system step to the n-process version.

Jayanti has subsequently proved several strong statements about the structure of Herlihy's hierarchy, using objects that can implement our combination protocol and were amenable to a number of highly novel "impossibility" arguments [Ja]. Specifically, he considers the four consensus hierarchies one obtains by varying the following two parameters:

- Do processes have access to just one copy of the object $\mathcal{O}$, or arbitrarily many copies?
- Do processes have access to an infinite set of atomic registers, or just (the copies of) $\mathcal{O}$?

He denotes these by h_1, h_m, h_1^r, h_m^r, where r indicates that registers are available, and m indicates that multiple copies of $\mathcal{O}$ are available. A hierarchy is *robust* if no set of objects lying below a given level k can provide a wait-free implementation of an object on level k for k or more processes. In his terminology, our results in Section 3 provide the most basic of these results: h_1 is not robust. [Ja] extends this to the more natural structures h_m and h_1^r (Herlihy's original hierarchy), showing that neither is robust. The robustness of h_m^r is an open question.

The combination protocol appears to be applicable in a number of non-trivially different contexts. In Section 4, we show an application to the problem of *consecutive m-register assignment*, under the definitions and hierarchy of [H1]. This is based on an atomic primitive considered by Herlihy in [H1]: simultaneous assignment to m registers, which he showed has consensus number exactly $2m-2$. However, his protocol relied on the ability of one process to assign to registers very far apart in memory. We show that this is in fact necessary; when the registers in a simultaneous assignment must be contiguous in memory, the consensus number is exactly 3, for all $m \geq 3$. We use a version of the combination protocol to show the lower bound on consensus number — the elimination tournament in this case can be continued for two rounds, but then is halted, in effect, by a combinatorial property of the one-dimensional array of registers.

Finally, in Section 5, we analyze two other protocols presented in [H1] from the point of view of resource bounds. It is worth noting that both of the primitives involved are simply classified as having infinite consensus number; yet our results reveal a strong sense in which one is more powerful than the other. Consider a system with an infinite set of atomic registers, as well as some number of special pre-initialized registers $x_1, \ldots, x_k$. The registers x_i can be read atomically, and the value in x_i can be atomically copied into x_j. [H1] gives a protocol for n processes to solve consensus using $2n$ such registers, allowing atomic writing to each x_i as well. We give an n-process protocol which does not require atomic writing to the x_i and uses only $\lfloor 2\log n + 2 \rfloor$ such registers; it is based on the surprising fact that a set of two such registers has infinite *bc*-number.

In contrast to this exponential improvement in resource requirements is the superficially similar case of registers with atomic swap. Here, the contents of the special registers x_i can be atomically swapped or read. Without assuming the ability to write atomically to the x_i, [H1] gives an n-process consensus protocol using $n+1$ such registers; we show that under this model, the protocol is *resource-optimal* — any n-process consensus protocol in this system requires $n+1$ registers-with-swap.

In summary, it is our belief that analyzing consensus protocols from the point of view of resource requirements reveals a number of subtleties in the structure of the hierarchy of shared objects. We hope that further analysis along these lines will resolve a number of important questions about the relative power of wait-free objects, as well as the problem of whether there exists a robust hierarchy for classifying them.

2 Definitions and Initial Results

2.1 The Model

We use essentially the model of computation presented in [H1, H2]. A concurrent system consists of a set of n processes which can access a collection of shared objects. In what follows, we consider only deterministic processes and objects.

A deterministic object is a possibly infinite-state automaton; in one step, a process can issue an *input* to the object, causing it to change state and return an *output*. Thus, we will often view an object $\mathcal{O}$ as a labeled directed graph (Q, Σ), where the vertex set Q is the set of states, and each element of the edge set Σ is labeled by a pair (input, output). When the object is in a given state $q \in Q$ and receives input i, it follows the edge with input label i. Since we are dealing with deterministic objects, each input label will appear on at most one edge out of q; if it appears on no edge, the operation is not *enabled* in q. A finite-state object is one for which Q and Σ are finite sets. We assume that all objects are *oblivious*: in a given object state, each process has the same set of possible (enabled) inputs. Finally, by a schedule for the system we mean simply a finite or infinite sequence of process names; at each entry in the sequence, the named process is allowed to take one step (i.e. apply a single input to an object, receive the output, and perform some internal computation).

The basic task we consider is that of consensus. A *consensus problem* for a set $\{p_1, \ldots, p_n\}$ of processes is characterized by a collection of sets $\{U_1, \ldots, U_n\}$. In a protocol to solve this problem, each participating process p_i begins by proposing a value $v \in U_i$; the processes then try to agree on a common value using operations on the shared objects in the system. Each process decides on a value, then halts. The protocol must satisfy the following properties.

- Wait-freedom: Each process decides and halts in a finite number of steps, regardless of whether other processes have crashed.
- Agreement: Each process decides on the same value.
- Validity: This common value is the proposal of some process.

When we simply use the term "consensus," we will mean specifically *leader election*, which is the consensus problem in which each process proposes its own name ($U_i = \{p_i\}$; the decision can be viewed as agreement on a "leader"). Thus, the validity condition can be viewed as follows: the decision value must be the name of a process that has taken at least one step. This essentially is the type of consensus considered in [H1]. We will also consider the other most common variation — *binary consensus*, in which all proposals must be 0 or 1; i.e. $U_i = \{0, 1\}$ for each i. The differences between these two problems will be considered below in Section 2.2.

Let us say that a consensus problem is *finite-choice* if each set U_i is finite. Consider a protocol for this problem which uses an object $\mathcal{O}$. Moreover, assume that the state of a process beginning the protocol depends only on the value of its proposal v. Then for a fixed choice of proposals, one for each process, we can view the set of schedules as a finitely branching tree in which every path is finite. By König's Lemma, the tree itself must be finite. Since there are only $\prod_{i=1}^{n} |U_i|$ possible trees, we have the following fact; it says in effect that if $\mathcal{O}$ is infinite-state, it can be "pruned" down to a finite-state object on which the identical protocol works.

Lemma 1 (Pruning Lemma) *Let $\mathcal{P}$ be a protocol for a finite-choice consensus problem which uses object $\mathcal{O}$. Then there is a finite-state object $\mathcal{O}_n$ which is formed by taking a finite subgraph on the states of $\mathcal{O}$, on which $\mathcal{P}$ is still an n-process consensus protocol.*

2.2 Two Kinds of Consensus

In a system with infinitely many atomic registers, any consensus problem can be reduced to leader election; if n processes can solve leader election, then they can solve any consensus problem. Without registers, however, this is not necessarily the case, so it makes less sense to speak of a single *consensus number* for an object. Thus, for a collection $\mathcal{C} = \{\mathcal{O}_1, \ldots, \mathcal{O}_k\}$ of shared objects, we will define the *le*-number and *bc*-number of this collection to be the maximum number of processes that can solve leader election and binary consensus, respectively, using $\mathcal{C}$.

It is possible for object's *bc*-number to exceed its *le*-number by an arbitrary amount. Consider the *sticky bit* of [Pl]; initially it is empty, and the first 0 or 1 written to it "sticks" permanently. A single copy of this object has infinite *bc*-number, but an *le*-number of 2.

In the other direction, however, we can show the following tight bound.

Theorem 1 *If an object $\mathcal{O}$ can solve leader election among n processes, then its bc-number is at least $\lfloor n/2 \rfloor$. For each $n \geq 4$, this bound is achieved by an object $\mathcal{O}_n$ with le-number n and bc-number $\lfloor n/2 \rfloor$.*

Proof. One direction is easy. Consider an arbitrary object $\mathcal{O}$ which can solve n-process leader election, and let $k = \lfloor n/2 \rfloor$. k processes $p_1, \ldots, p_k$ can use $\mathcal{O}$ to solve binary consensus by collectively simulating the execution of a leader election protocol among the $2k$ processes $q_1^0, \ldots, q_k^0, q_1^1, \ldots, q_k^1$, as follows. In order to propose u, p_i simulates the behavior of q_i^u. When the protocol terminates with leader q_j^v, the processes decide on v. It is easily verified that this protocol satisfies the properties of binary consensus.

Now let $n \geq 4$, and consider the following object $\mathcal{O}_n$. The states of $\mathcal{O}_n$ are $\{c, q_1, \ldots, q_n\}$; its operations are $\{x_1, \ldots, x_n\}$. The operations are defined as follows:

in state c, operation x_i leads to q_i; in state q_i, operation x_i leads to c. For $j \neq i$, performing x_j in state q_i leaves the object in state q_i. Finally, $\mathcal{O}_n$ supports an atomic read operation which returns the current state. It is clear that n processes can elect a leader using $\mathcal{O}_n$. The object is initialized to state c; process p_i performs operation x_i and then reads the current state. It is also easily verified that the le-number of $\mathcal{O}_n$ is in fact exactly n.

We claim that with $k = \lfloor n/2 \rfloor$, $k+1$ processes cannot solve binary consensus using $\mathcal{O}_n$. Assume by way of contradiction that such a protocol $\mathcal{P}$ exists. We consider here the case in which $\mathcal{O}_n$ begins in state c; the case in which it begins in some q_i is analogous. If process p_i proposes v and all other processes crash, then we say p_i is executing a *solo run*; let r_i^v denote the state of $\mathcal{O}_n$ in which this solo run terminates.

Clearly, we cannot have $r_i^v = c$ for any i, v, since otherwise some $p_j \neq p_i$ would not realize when it first woke up that v had already been decided. For a similar reason (since $k+1 \geq 3$), we cannot have $r_i^0 = r_j^1$ for any i, j. Thus by the pigeonhole principle there is a state $q_m = r_i^v = r_j^v$ for some $i \neq j$.

Consider the following execution. Let p_i and p_j both propose v, and some third process p_r (since $k+1 \geq 3$) propose $\bar{v} = 1 - v$.

- Run p_i alone until it is about to leave $\mathcal{O}_n$ in state $q_m = r_i^v$. Thus $\mathcal{O}_n$ must be currently in state c.

- Now schedule p_j alone; since it first sees $\mathcal{O}_n$ in state c, it will attempt to leave the object in state q_m as well. Run p_j until it is about to execute operation x_m from state c.

- Run p_i to completion; it will execute x_m and halt, deciding v.

- Let p_j take a single step – the operation x_m – and crash it. The object has now returned to state c.

- Wake up p_r and let it run to completion, deciding $\bar{v}$.

This contradiction completes the proof. ∎

A similar result clearly holds for m-valued consensus for each $1 < m < n$. The theorem above can also be viewed as an extension of the result of [JT], that there exists an object which can solve 2-process leader election but not 2-process binary consensus.

3 The Combination Protocol

As noted in the Introduction, there are a number of possible objects one could use in implementing the combination protocol; we focus on the finite-state object type $\mathcal{S}_2$, defined below, throughout this section.

Define a *monotone bit* to be a single bit that supports atomic reading and the writing of the value 1 — the value 0 cannot be written to a monotone bit. An object of type $\mathcal{S}_2$ consists of two monotone bits (the "left" and "right" bits) with the following operations:

- *read*: atomically read the value of the two bits simultaneously
- R: write 1 to the right bit.
- L: write 1 to the left bit.
- S: atomically swap the values of the two bits.

We now outline the combination protocol to solve n-process leader election using $n-1$ copies of $\mathcal{S}_2$. Note that by Theorem 1, this means that $\lfloor n/2 \rfloor$ processes can solve binary consensus using $n-1$ copies of $\mathcal{S}_2$; i.e. there is a protocol to solve n-process binary consensus using $2n-1$ copies of $\mathcal{S}_2$. Thus, these results show that an infinite set of copies of $\mathcal{S}_2$ has infinite *le*- and *bc*-numbers.

Let $\{p_1, \ldots, p_n\}$ be the set of processes and $\{s_1, \ldots, s_{n-1}\}$ the set of copies of $\mathcal{S}_2$. Each s_i is initialized with the value 01 (0 in the left bit and 1 in the right). The protocol consists of $n-1$ numbered phases, followed by a final phase. Process p_j participates in phases $\max\{1, j-1\}$ through $n-1$ in succession, followed by the final phase; at the end of the final phase, it reaches a decision. Note that there is no waiting involved; a process begins executing its first designated phase as soon as it takes its first step.

Phase j $(1 \leq j \leq n-1)$ involves only operations on s_j; the code for phase j is as follows:

```
Phase j
 p_1,...,p_j executes L on s_j
 p_{j+1} executes S on s_j
```

The code for the final phase is as follows:

```
Final Phase
 each p_i reads s_1,...,s_{n-1} one at a time
 if all values read are 11 then
  elect p_1
 else
```

```
elect p_j, where s_{j-1} is the
      highest-indexed copy of S_2
      whose value was read as 10
```

Theorem 2 *The above protocol is a solution to n-process leader election.*

Proof. It is obvious that it is wait-free. If p_1 is elected, then L must have been executed on s_1, which only p_1 can do. If p_i is elected, then S must have been executed on s_{i-1}, which only p_i can do. Thus the elected process has taken a step.

Finally, we must verify agreement. First observe that each copy of s_i assumes at most two different values over the execution of the protocol. It is initially in state 01. If the first operation invoked on it is S, then it assumes the value 10, and all further executions of L will have no effect. If the first operation is L, then its value becomes 11, and *no* subsequent operation can have an effect.

Now let σ denote the schedule associated with a given execution of the protocol, and σ^* denote the prefix of σ up to the first point at which some process has completed all its numbered phases (i.e. is about to begin the final phase). We claim that the system state following the execution of σ^* satisfies one of the following two properties.

(i) All s_i have the value 11, $i = 1, \ldots, n-1$, or

(ii) There is some $j \geq 1$ such that s_j has the value 10, and $s_{j+1}, \ldots, s_{n-1}$ have the value 11 (or $j = n-1$).

To see this, assume that the system does not have property (i), and let j be the maximal index for which s_j does not have the value 11. We claim it has the value 10, satisfying property (ii). For if not, then it must have the value 01, in which case no process has invoked an operation on s_j. If $j = n-1$, this contradicts the definition of σ^*, since no process could have completed phase $n-1$. And if $j < n-1$, then s_{j+1} has the value 11, which can only result from the execution of L by one of $p_1, \ldots, p_{j+1}$. But each of these processes must have taken an earlier step in which it invoked an operation on s_j — again, a contradiction. Thus, s_j has the value 10, and property (ii) holds.

Hence, if property (i) holds after the execution of σ^*, then all processes that complete their final phase will decide on p_1. If property (ii) holds after σ^*, then all processes that complete their final phase will decide on p_{j+1}. In both cases, agreement is satisfied. ∎

Let us consider the power of a single copy $\mathcal{O}$ of $\mathcal{S}_2$. Using fairly straightforward protocols, two processes can solve either leader election or binary consensus using $\mathcal{O}$. (In both cases, initialize $\mathcal{O}$ to 01. For leader election, let p_1 execute L and p_2 execute S; for binary consensus, let a process proposing 0 execute S and a process proposing 1 execute L.) Indeed, we can show the following tight bounds.

Theorem 3 *A single copy of $\mathcal{S}_2$ has le-number and bc-number equal to* 2.

Proof. Given the discussion above, the result will follow if we show that 3 processes can solve neither leader election nor binary consensus using a single copy of $\mathcal{S}_2$. We present the proof for binary consensus; the proof for leader election is similar but easier, and is left to the reader.

Assume by way of contradiction that there is such a protocol $\mathcal{P}$. To avoid notational confusion, we will let 0 and 1 denote the values of the bits in the copy of $\mathcal{S}_2$, and use v and $u = \bar{v}$ to denote the outcomes of binary consensus. Let p, q, and r denote the three processes.

First consider the case in which the initial state is 01 (or analogously 10). We again make use of the notion of a process p's *solo run*, in which it first "wakes up" with the system in its initial state, and runs to completion, deciding its own value. Clearly p cannot distinguish between the scenario in which it is the only live process, and the one in which other processes have executed but left the system in its initial state.

In view of this, it must be the case that the solo run of a process proposing u leaves the object in state 10, and the solo run of a process proposing v leaves it in state 11 (or vice versa). Consider an execution in which p and q propose u, and r proposes v. We have the following two scenarios.

In Scenario $S1$, process r runs to completion, deciding v. Scenario $S2$ is as follows (see Figure 1).

- Run p and q until they are about to execute S (which they must do) for the first time.
- Run p to completion; it will decide u.
- Now run q for a single step — the execution of S — and crash it, leaving the object in state 10.
- Finally, run r to completion.

These two scenarios are indistinguishable to process r, so it will decide v in $S2$ as well — but this violates agreement.

In the other case, the initial state is 00. If two processes have opposite proposals, their solo runs clearly cannot leave the object in the same state. Let us therefore assume that there is a solo run of some process proposing u which leaves the object in state 10. Furthermore, we consider the case in which a process waking up to find the object in states 01 or 11 will decide v; all other cases are analogous.

Assume that p proposes u, and its solo run leaves the object in state 10. Schedule p alone until it is about to write to a register. It cannot execute R, since we then crash it and wake up q, which also proposes u. But q must decide v, since the object

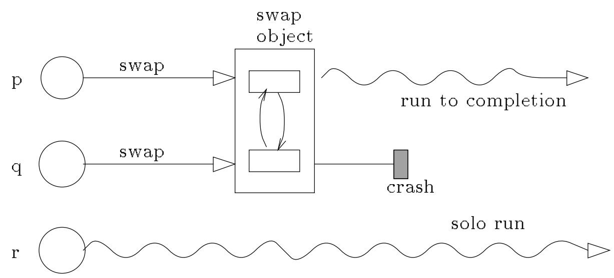

Figure 1: An indistinguishable scenario

is in state 01, violating validity. Thus, p must first perform operation L. But in this case, we have the following execution:

- Run p until just before its first write operation, which will be L.
- Let q propose v and run it until just before its first write operation — this must be R by the same reasoning.
- Run p to completion (it decides u).
- Schedule q for the single step R, and crash it.
- Finally, wake up r. Since the object is in state 11, it must decide v.

This execution violates agreement, which completes the second case and the proof. ∎

As noted earlier, this means that although the consensus numbers of a single copy of $\mathcal{S}_2$ are equal to 2, an infinite set of them can solve consensus among any number of processes. Hence our belief that *consensus number*, considered by itself, does not give a complete description of the power of a shared object to support consensus protocols. In the terminology of [Ja], the following is an immediate corollary of Theorems 2 and 3 (recall that h_1 is the hierarchy of shared objects in which the consensus number of $\mathcal{O}$ is the number of processes that can solve consensus using a single copy of $\mathcal{O}$ and no registers).

Corollary 1 *The hierarchy h_1 is not robust.*

4 A Variation on the Protocol

For the remainder of this paper, we consider the combination protocol, and resource bounds in general, from the point of view of the original model of Herlihy [H1]. Thus, we will assume that our system contains an infinite set of atomic registers; again, we will use the terms *consensus* and *consensus number* to refer to the problem of leader election in such systems.

[H1] considers the primitive *simultaneous m-register assignment.* A system with this primitive consists of an infinite set of atomic registers, and supports an additional atomic operation in which a process can simultaneously assign m (potentially different) values to m distinct registers. It is shown in [H1] that a system with this primitive has consensus number $2m-2$. It seems natural to consider what the consensus number would be if we did not allow simultaneous assignment to registers that can be arbitrarily far apart in memory. Thus we examine the case in which the m registers to which a process writes must be contiguous in memory.

The technique of the combination protocol finds application here, as we use it to give a 3-process consensus protocol when $m \geq 3$. The idea is as follows. The advantage of simultaneous assignment is that by writing to sets of registers that intersect, two processes can determine which took the first step by reading the registers in the "overlap." So in the first phase, two processes can write to intersecting intervals in memory. They then move to a different part of memory, and write to intervals that intersect the opposite ends of the third process's interval. In this way, a consistent *leader* can be elected. See the proof below for the actual protocol.

Unlike the case of the basic combination protocol, one must contend here with the "geometry" of the array of registers. Specifically, a third phase in this spirit is not possible; an interval has only two endpoints. In fact, employing a combinatorial argument about the intersections of four intervals on a line, we can show that 3-process consensus is the best that can be achieved in this system.

Theorem 4 *An infinite set of registers with atomic read, write, and consecutive m-register assignment, has consensus number* 2 *if* $m=2$*, and* 3 *if* $m \geq 3$.

Proof. The case in which $m=2$ is straightforward. Thus let us assume $m \geq 3$; first we give a protocol for 3 processes to solve leader election. Let p_1, p_2, p_3 be the names of the processes, and let the registers be numbered $0, 1, \ldots$. We will say that p_i *marks* an interval $[a, b]$ by simultaneously writing its name in registers $a, a+1, \ldots, b$.

The protocol is as follows.

`Phase 1`

p_1 `marks` $[0, m-1]$

```
 p2 marks [1,m]
Phase 2
 p1 marks [m+1,2m]
 p2 marks [3m-1,4m-2]
 p3 marks [2m,3m-1]
Final phase
 Each pi reads registers 0,...,4m-2,
   following the technique of [H1]
 if p3 completed Phase 2 before p1 or p2
   then elect p3
 else if p2 completed Phase 1 before p1
   then elect p2
 else
   elect p1
```

It is straightforward to verify that this satisfies wait-freedom, validity, and agreement.

To show the impossibility of a solution to 4-process consensus, we follow the standard structure for impossibility proofs of this sort (see e.g. [FLP, DDS]). Assume that such a protocol $\mathcal{P}$ exists for 4-process consensus in this system. Let σ denote the schedule of a partial execution of $\mathcal{P}$ and $q(\sigma)$ the system state following the execution of σ. We say that $q(\sigma)$ is v-determined if in all extensions of σ, the decision v is reached. A state is *determined* if it is v-determined for some v, and *undetermined* otherwise. By the validity requirement, the starting state $q(\lambda)$ is undetermined; but a state in which all processes have decided is clearly determined. Thus, a simple and oft-repeated argument shows that there exists at least one *boundary state* $q(\sigma)$: $q(\sigma)$ is undetermined, but every extension of σ by one step results in a determined state.

Thus, we consider a boundary state $q(\sigma)$ in the execution of our supposed 4-process protocol $\mathcal{P}$. Let the "preference" v_i of p_i be the value it would decide if all other processes crashed at this point. As in [H1], observe that all four steps extending σ must be consecutive assignments (in this case, to "intervals" in memory), with the properties that the interval of each p_i must include a register that no other process writes to, and that if p_i and p_j have different preferences, their intervals must intersect in some register that no third process writes to. Assume without loss of generality that $v_1 \neq v_2$, $v_1 \neq v_3$ (i.e. the preference of p_1 differs from those of p_2 and p_3). Then the interval of p_1 intersects those of p_2 and p_3. There are two cases to consider.

1. $v_4 \neq v_1$. Then if p_4's interval intersects p_1's, one of p_2, p_3, or p_4 will have no register that it alone writes to.

2. $v_4 = v_1$. Then if p_4's interval intersects those of both p_2 and p_3, one of p_1 or p_4 will have no register that it alone writes to.

As both cases lead to contradictions, the proof is complete. ∎

5 Resource Bounds

Finally, we analyze the resource requirements of n-process consensus using two other primitives considered in [H1]. Consider a system with an infinite set of atomic registers, and also some special pre-initialized *move-registers* $x_1, \ldots, x_k$. These special registers support the atomic operations *read(x_i)*, which returns the contents of x_i, and *move(x_i, x_j)*, which atomically copies the contents of x_i into x_j. These are the only two operations allowed on the registers x_i. Alternately, we can consider a system with additional pre-initialized *swap-registers* $y_1, \ldots, y_k$; here the atomic operations are *read(y_i)* and *swap(y_i, y_j)*, which atomically exchanges the contents of y_i and y_j. In the latter case, note the difference between these swap-registers and the object $\mathcal{S}_2$ of Section 3; here, a process can swap between *any* pair of registers.

In [H1], protocols are given for n processes to solve consensus using either kind of primitive; the conclusion is that they both have infinite consensus number (actually, the protocol using move in [H1] required the ability to write directly to the move-registers; below, we give a protocol that does not need this assumption). These protocols with move and swap require $2n$ and $n+1$ special registers, respectively, suggesting that there is not much difference in the power of these two primitives.

The results of this section show a sense in which *move* is actually significantly more powerful than *swap*; they reveal an exponential gap between the number of move-registers and the number of swap-registers needed to solve n-process consensus. We give an n-process consensus protocol which requires only $\lfloor 2\log n + 2 \rfloor$ registers with move; on the other hand, we show that any n-process consensus protocol using swap-registers requires $n+1$ such registers; i.e. the protocol of [H1] is resource-optimal. We note that the protocol using *move* still requires n additional ordinary atomic registers; thus, the exponential gap does not extend to the total memory requirements of the protocols.

Theorem 5 *There is an n-process consensus protocol in a system with infinitely many atomic registers, and move-registers $x_1, \ldots, x_k$, where $k = \lfloor 2\log n + 2 \rfloor$.*

Proof. The key fact is that two move-registers are in fact sufficient for any number of processes to solve binary consensus. Let x_1 and x_2 denote the two registers; initialize x_1 with 0 and x_2 with 1. To propose 0, a process copies from x_1 to x_2; to propose 1, a process copies from x_2 to x_1. After the first step taken by any process, the values of the two registers become the same — subsequent *move* operations have no effect. Thus it is straightforward to verify that it is a solution to n-process binary consensus for any n.

Now, in [Pl] a leader election protocol using $\lfloor \log n + 1 \rfloor$ sticky bits is presented (the process is elected "one bit at a time," with n atomic registers used to ensure that validity is satisfied). It is easy to adapt this technique to our case; here each sticky bit is replaced by a pair of move-registers. Consequently, we have the stated bound. ∎

Theorem 6 *Consider a set of arbitrarily many atomic registers, and swap-registers $y_1, \ldots, y_k$. If n processes can solve consensus in this system, then $k \geq n + 1$.*

Proof. Let $\mathcal{P}$ be any protocol for n processes to solve consensus in this system, and let $q(\sigma)$ be a boundary state in the execution of this protocol (as in the proof of Theorem 4). By the usual arguments, we see that all operations scheduled out of σ must be swaps. Let the *preference* v_i of p_i be the value it would decide if all other processes crashed at this point. Also, it is easy to verify that 2 swap registers will not suffice for 2 processes; thus, in what follows, we can assume $n \geq 3$.

We define a graph $G = (V, E)$ as follows. The vertex set $V = \{s_1, \ldots, s_k\}$. There is an (undirected) edge (s_i, s_j) if and only if some process p_r is executing *swap*(s_i, s_j) from σ. Clearly two processes cannot perform the same swap operation, so this is a graph without loops or multiple edges. Hence, we label each edge with the preference v of the process performing the associated swap.

A straightforward indistinguishability argument establishes the first of the following claims.

Claim 1 *If e and e' are edges with different labels in G, then they must share an endpoint.*

Claim 2 *If G contains a cycle, all edges on the cycle must have the same label.*

Proof. Let C be a cycle in G containing edges of both labels; we obtain contradictions in the following three cases.

1) C is a 3-cycle. Let e_1, e_2, e_3 be the three edges of C, and p_1, p_2, p_3 the three processes performing the swaps. We can assume without loss of generality that $v_1 \neq v_3$. Then scheduling $p_1p_2p_3$ from σ results in the same system state as scheduling $p_3p_2p_1$, yet these states are determined for different values.

2) C is a 4-cycle. Let $e_1, \ldots, e_4$ be the four edges in "clockwise" order, and $p_1, \ldots, p_4$ the associated processes. Claim 1 implies that $v_1 = v_3$ and $v_2 = v_4$; hence it must be that $v_1 \neq v_2$. But from σ, the schedules $p_1p_3p_2p_4$ and $p_2p_4p_1p_3$ leave the system in the same state.

3) C is a b-cycle, $b \geq 5$. Let e and e' be neighboring edges with different labels v, v', respectively. Let e_1 and e_1' be the other neighboring edges of e and e' respectively;

since e_1 and e_1' do not share an endpoint, Claim 1 implies that they must have the same label. But then one of the disjoint pairs $\{e_1, e'\}$, $\{e, e_1'\}$ must have different labels. ∎

Claim 3 *G is acyclic.*

Proof. Suppose G had a cycle C. By Claim 2, all edges of C must have the same label, say v. Since σ is undetermined, some edge e has label $v' \neq v$. But e cannot share a vertex with all the edges of C; this contradicts Claim 1. ∎

The theorem now follows easily. Since G is an acyclic graph with n edges, we must have $|V| = k \geq n + 1$. ∎

6 Conclusion

By viewing wait-free primitives as individual objects that can be replicated and combined, we have been able to consider consensus from the perspective of resource requirements. The *combination protocol* of Section 3 presents a general technique for designing consensus protocols within this framework.

A number of questions remain open. First of all, we are interested in other applications of the combination protocol in the design of wait-free algorithms. Our work and that of [Ja] suggests that perhaps it is most natural to define the consensus number of an object $\mathcal{O}$ as the maximum number of processes that can solve consensus using an infinite set of registers and an infinite set of copies of $\mathcal{O}$. Nevertheless, we do not know whether, even under this definition, there exist objects $\mathcal{O}_1$ and $\mathcal{O}_2$, with consensus numbers $k_1 \leq k_2$ respectively, such that more than k_2 processes can solve consensus when the two objects are used together. A solution to this problem would settle a basic question about the nature of asynchronous consensus, and would very likely reveal a number of new ideas in the analysis of consensus protocols.

Acknowledgements

The authors wish to thank Prasad Jayanti, Keith Marzullo, and Sam Toueg for helpful discussions on the topic of this paper. Thanks also to Cynthia Dwork for helping to improve the presentation.

References

[CHP] P. Courtois, F. Heymann, D. Parnas, "Concurrent Control with Readers and Writers," *Communications of the ACM*, 14(1971), pp. 667-668.

[DDS] D. Dolev, C. Dwork, L. Stockmeyer, "On the Minimal Synchronism Needed for Distributed Consensus," *Journal of the ACM*, 34(January 1987), pp. 77–99.

[FLP] M.J. Fischer, N.A. Lynch, M.S. Paterson, "Impossibility of Distributed Consensus with One Faulty Process," *Journal of the ACM* 32(April 1985), pp. 374–382.

[H2] M. Herlihy, "Impossibility Results for Asynchronous PRAM," *Proc. Third ACM Symposium on Parallel Algorithms and Architectures*, 1991.

[H1] M. Herlihy, "Wait-Free Synchronization," *ACM Transactions on Programming Languages and Systems*, Jan. 1991, pp. 124–149.

[Ja] P. Jayanti, "On the Robustness of Herlihy's Hierarchy," *Proc. Twelfth ACM Syposium on Principles of Distributed Computing*, 1993.

[JT] P. Jayanti, S. Toueg, "Some Results on the Impossibility, Universality, and Decidability of Consensus," *Proc. Sixth Workshop on Distributed Algorithms*, 1992.

[LAA] M.C. Loui, H.H. Abu-Amara, "Memory Requirements for Agreement Among Unreliable Asynchronous Processes," *Advances in Computing Research*, vol. 4, 1987, pp. 163–183.

[Pl] S. Plotkin, "Sticky Bits and Universality of Consensus," *Proc. Eighth ACM Symposium on Principles of Distributed Computing*, 1989.