SIAM J. COMPUT.
Vol. 30, No. 4, pp. 1156–1168

DETERMINING CONSENSUS NUMBERS*

ERIC RUPPERT†

Abstract. Conditions on a shared object type T are given that are both necessary and sufficient for wait-free n-process consensus to be solvable using objects of type T and registers. The conditions apply to two large classes of deterministic shared objects: read-modify-write objects [C. P. Kruskal, L. Rudolph, and M. Snir, *ACM Trans. Prog. Lang. Syst.*, 10 (1988), pp. 579–601] and readable objects, which have operations that allow processes to read the state of the object. These classes include most objects that are used as the primitives of distributed systems. When the sequential specification of T is finite, the conditions may be checked in a finite amount of time to decide the question "Is the consensus number of T at least n?" The conditions are also used to provide a clear proof of the robustness of the consensus hierarchy for read-modify-write and readable objects.

Key words. consensus, distributed computing, asynchronous, wait-free, shared memory, read-modify-write, readable, consensus hierarchy, robust

AMS subject classifications. 68Q22, 68Q10

PII. S0097539797329439

1. Introduction.

A shared-memory distributed system consists of a number of processes that communicate with one another by performing operations on shared data objects. This paper studies asynchronous systems where processes take steps at arbitrarily varying speeds. The wait-free model of fault tolerance is used: each nonfaulty process must complete its task correctly even if any number of processes experiences halting failures. The power of a system to solve problems depends on the types of shared data objects that are available. Determining whether a given collection of object types can be used to solve a given problem is therefore of great importance in the design of distributed systems. A problem may, itself, be modelled as a shared object type: processes pass their input to the object, and the object returns the desired output. Thus, a question about computability may be formulated as a question about implementations: Can one object type be implemented using a given set of primitive object types? Herlihy [10] showed that the consensus problem, in which each process begins with an input and all nonfaulty processes must agree on one of the input values, plays a central role in the study of the power of object types. He proved that objects that solve the consensus problem for n processes can be used, along with read/write registers, to obtain a wait-free implementation of any object for n or fewer processes. This leads to the idea of classifying object types according to their consensus number [10, 14]. An object type T *has consensus number* n if n-process consensus can be solved by using objects of type T and read/write registers, but $(n+1)$-process consensus cannot. If there is no such n, the consensus number of T is ∞. Similarly, the consensus number of a set $\mathcal{S}$ of types is the largest n for which n processes can solve consensus using objects whose types are in $\mathcal{S} \cup \{\text{register}\}$, or ∞ if no such n exists.

*Received by the editors November 3, 1997; accepted for publication (in revised form) April 14, 2000; published electronically September 20, 2000. A preliminary version of this paper appeared at the 16th ACM Symposium on Principles of Distributed Computing, Santa Barbara, CA, 1997, pp. 93–99. This research was supported by the Natural Sciences and Engineering Research Council of Canada.

http://www.siam.org/journals/sicomp/30-4/32943.html

†Department of Computer Science, Brown University, Providence, RI 02912 (ruppert@cs.brown.edu).

This paper addresses the problem of determining whether a given deterministic object type has consensus number n. Two important classes of objects that include most of the objects considered as primitives for distributed systems are studied: read-modify-write (RMW) objects [16] and readable objects. It is reasonable to assume that processes will be able to access the information stored in the data object in some simple way. If processes may read the information directly, without altering the object's state, then the object is called readable. If the operation that reads the information in the data object is combined with an update operation, so that the read and update occur as a single atomic action, the object may be modelled as an RMW object.

A *readable* object O is modelled by an I/O automaton whose state set is the Cartesian product $Q = \bigtimes_{k\in\Gamma} Q_k$, where Γ is an index set and Q_k is a set for each $k \in \Gamma$. Neither Γ nor Q_k must be finite. For each $k \in \Gamma$, processes may execute the operation $read_k$, which returns component k of the current state of O. In addition to the *read* operations, the object may have an arbitrary set of other (deterministic) operations defined on it. Each operation can update any set of components of the state, and the set of components that are updated by an operation may depend on the current state of the object. Any object type with a *read* operation that returns the entire state of the object is readable; in this case, $|\Gamma| = 1$. An array of m 1-bit registers is a readable object, with $\Gamma = \{1, \ldots, m\}$ and $Q_k = \{0, 1\}$ for all $k \in \Gamma$. Readable objects include arrays of registers whose elements can be read, copied, or swapped atomically. An array of unbounded registers that can be read or used for indirect addressing is another example of a readable object; in this case the state set would be $Q = \bigtimes_{k\in\mathbb{N}} \mathbb{N}$ and an indirect write to component i updates the component indexed by the number stored in component i.

An RMW object type allows various kinds of *RMW* operations to be performed on it. Let V be the state set of the RMW object. Any function $f : V \to V$ defines an *RMW* operation, denoted RMW_f, that reads the current value, x, of the RMW object, updates the value to $f(x)$, and returns x. For example, a *fetch&increment* operation applies the function $f(x) = x + 1$, and a *read* operation applies the identity function. An RMW object type is defined by the set F of functions mapping V to itself that may be applied by the *RMW* operations. This class of objects includes compare&swap, test&set, and fetch&add variables.

Previously, ad hoc arguments have been used to determine whether particular object types can be used along with registers to solve n-process consensus. An exception was Herlihy's proof [10] that 2-process consensus can be solved using registers and RMW objects, where the *RMW* operations apply functions from the set F if and only if F contains a function different from the identity. Herlihy also gave a necessary condition on the set F to describe when RMW objects can be used with registers to solve 3-process consensus. In this paper, a decision procedure is developed to determine whether given readable and RMW types with finite specifications can be used to solve n-process consensus for any n. Jayanti and Toueg [15] showed that this question is undecidable for arbitrary object types that are allowed to have infinite state sets. It will be shown here that an RMW or readable type T can be used together with registers to solve the consensus problem among n processes if and only if T satisfies certain conditions which are decidable for types that have finite specifications. An object type that satisfies the conditions is called *n-discerning*. This characterization has been useful for studying the consensus numbers of multi-objects and transactional objects, where processes can access more than one object in an atomic action [21, 22].

Some work has been done on deciding whether a given task can be solved using some particular types of shared objects. Biran, Moran, and Zaks [4] showed that one can decide whether a given task can be solved in a message-passing system if at most one process may fail. Chor and Moscovici [7] gave a decidable characterization of tasks that can be solved by randomized algorithms that use registers only. Gafni and Koutsoupias [9] showed that there is no algorithm which determines whether a given task for three processes can be solved by read/write registers in a wait-free manner. Herlihy and Rajsbaum [11] gave decidability results for the question of whether a given task can be solved using various types of objects (registers, consensus objects, and set consensus objects) in the presence of t process failures.

The proof that the property of being n-discerning is sufficient for the solvability of n-process consensus will also provide upper bounds on the resources required to solve the consensus problem. If an RMW or readable object type T can be used together with registers to solve consensus among n processes, then there is a consensus protocol that uses at most $n-1$ objects of type T and $2(n-1)$ registers. Furthermore, if T is an RMW object type, each process takes $O(n)$ steps in this protocol. It also follows from the construction of the protocol that n-process consensus can be solved using $2(n-1)$ registers and $n-1$ objects, $X_2, \ldots, X_n$, where X_i's type is i-discerning, for each i.

The characterization of n-discerning types will be used in section 5 to obtain a clear proof that the consensus hierarchy is robust [14] for RMW and readable objects. This means that if n-process consensus can be solved using RMW and readable objects of types $T_1, \ldots, T_r$ and registers, then n-process consensus can also be solved using only registers and objects of type T_i for some i. This robustness property allows the consensus number of a set of RMW and readable types to be determined by finding the consensus number of each type separately. Borowsky, Gafni, and Afek [5] claimed that the consensus hierarchy is robust for all deterministic objects. A full version of their paper is not yet available.

The n-discerning conditions for readable objects will also be used to generalize a result about atomic snapshot objects. A snapshot object can be thought of as a finite array of registers with an additional *scan* operation that reads the entire array at once. It is known that the addition of the *scan* operation does not increase the power of an array of registers to solve consensus, since the snapshot object can be implemented from ordinary registers [1, 2, 3]. Here, it will be shown that the addition of an atomic *scan* operation does not increase the power of *any* readable object type to solve consensus.

2. Preliminaries. An object type is defined using a sequential specification, which describes the operations that may be performed on the object and the responses the object should return if the operations are performed sequentially. Formally, an object can be specified as an I/O automaton (see [18]). Each operation causes a state transition and returns a response. If the state transition and response are uniquely determined by the current state of the object and the operation applied, then the object is *deterministic*. When an object is accessed by more than one process concurrently, its behavior can be specified by insisting that operations appear to occur instantaneously at some time between their invocations and their responses. Such an object is called *linearizable* [12]. This paper deals only with deterministic, linearizable objects. For simplicity, it is assumed that all objects are *oblivious*: every process can invoke every operation, and, for any operation, the state transition and response do not depend on the process that invokes the operation.

It is assumed that the designer of a consensus protocol may choose the initial states of the shared objects. There is no real loss of generality in this assumption; Borowsky, Gafni, and Afek [5] showed that if consensus can be solved when the initial states are chosen by the programmer, then consensus can be solved using objects initialized to a particular state, assuming there is some sequence of operations that will move the object from the given initial state into any other state.

The systems studied here are completely asynchronous, so that algorithms must exhibit correct behavior regardless of the way in which the steps of different processes are interleaved by a scheduler. Algorithms are required to be wait-free [10]. This means that the algorithm executed by each nonfaulty process must work correctly even if other processes experience halting failures.

An implementation of an object of type T from objects $O_1, \ldots, O_m$ is a protocol that uses only the shared objects $O_1, \ldots, O_m$. This protocol consists of an algorithm *apply*(*op*) for each process that can simulate every operation *op* on an object of type T. The protocol should also specify the initial states for $O_1, \ldots, O_m$ to be used for any possible initial state of the simulated object. If each process performs the *apply* routine repeatedly, the responses returned should appear as if the operations were carried out atomically on an object of type T. All implementations are assumed to be wait-free.

The consensus number of a set of object types is defined in terms of its ability to solve the consensus problem. For the n-consensus problem, n processes each begin with a private input value, and each nonfaulty process outputs a value in a wait-free manner. The output values must all be the same, and every output must be the input value of some process. These two conditions are called *consistency* and *validity*, respectively.

The proofs that the property of being n-discerning is necessary for the solvability of n-consensus are bivalency arguments. This type of proof was introduced by Fischer, Lynch, and Paterson [8]. The following terminology will be used in the bivalency arguments. The *configuration* of a protocol at any moment in its execution consists of the state of every shared object, together with the internal state of every process. Two configurations are *indistinguishable to process* P if the state of each shared object and the internal state of P is the same in the two configurations. A configuration of a consensus protocol is x-*valent* if all nonfaulty processes decide on the value x in all executions continuing from that configuration. A configuration is called *univalent* if it is x-valent for some x, and *multivalent* otherwise. A configuration is *critical* if it is multivalent and the next step by any process will move the system into a univalent configuration. The value that would be decided if a particular process takes the next step after a critical configuration is called the *critical value* of that process.

3. Solving consensus with RMW objects. Consider an RMW type T with state set V whose *RMW* operations can apply functions from the set F. This section describes the power of a distributed system with $n \geq 2$ processes, $P_1, \ldots, P_n$, to solve consensus using objects of type T and registers. The sets V and F need not be finite.

Let v_0 be a value in V. Partition the set of processes $\{P_1, \ldots, P_n\}$ into two nonempty teams, A and B. Associate a function $f_i \in F$ with each process P_i. The functions f_i need not be distinct. The type T is called n-*discerning* if there exist choices for $v_0, A, B, f_1, \ldots, f_n$ so that, in any schedule in which each process P_i applies the single operation RMW_{f_i} to an object X which initially has value v_0, every process can determine (from the value returned by its operation) whether a process from team A or a process from team B was the first process to take a step in the schedule. Such a type T is called n-discerning since processes can easily use an object of type T to

discern the difference between schedules that start with a process on team A from those that begin with a process on team B. This description is formalized in the following definition. The notation $f \circ g$ is used to denote functional composition: $(f \circ g)(x) = f(g(x))$.

DEFINITION 1. *Let $n \geq 2$. The RMW type defined by the state set V and the set F of functions is n-discerning if there exist*

- *$v_0 \in V$,*
- *a partition of the set of processes $\{P_1, \ldots, P_n\}$ into two nonempty teams A and B, and*
- *a function $f_i \in F$ for each process P_i*

such that

I. *for all $j \in \{1, \ldots, n\}, V_{A,j} \cap V_{B,j} = \emptyset$,*

II. *for all $P_j \in B, v_0 \notin V_{A,j}$, and*

III. *for all $P_j \in A, v_0 \notin V_{B,j}$, where*

$V_{A,j} = \{(f_{i_\alpha} \circ \cdots \circ f_{i_1})(v_0) \mid P_{i_1} \in A,\ \alpha \geq 1$, and $i_1, \ldots, i_\alpha$ are distinct process indices, not including $j\}$, and $V_{B,j} = \{(f_{i_\alpha} \circ \cdots \circ f_{i_1})(v_0) \mid P_{i_1} \in B,\ \alpha \geq 1$, and $i_1, \ldots, i_\alpha$ are distinct process indices, not including $j\}$.

Suppose each process performs its assigned operation to the object X, which is initialized with the value v_0. Consider a process P_j on team A. The set $V_{A,j}$ contains the values that P_j can see when it accesses object X if some other process on team A performs the first step. (The set $V_{A,j}$ will be empty if P_j is the only process on team A. It may be the case that v_0 is in $V_{A,j}$.) The set $V_{B,j}$ is the set of responses that P_j can receive if a process from team B takes the first step. Thus, condition I ensures that P_j can distinguish, using the response it receives, executions starting with another process on team A from those in which a process on team B takes the first step. Condition III ensures that P_j can distinguish executions in which P_j itself takes the first step from those executions in which a process from team B takes the first step. (Similarly, if the process P_j is on team B, conditions I and II guarantee that it can tell which team took the first step.)

It will be shown in Theorems 3 and 9 that RMW objects of type T can be used with registers to solve n-consensus if and only if T is n-discerning.

LEMMA 2. *If S_0 is a critical configuration of an n-process consensus protocol where the next step of every process accesses the same RMW object X of type T, then T is n-discerning.*

Proof. Suppose each process P_i applies the operation RMW_{f_i} to X during its first step after S_0. Let v_0 be the value of X in the configuration S_0. Let a be the critical value of one of the processes. Let A be the set of processes with critical value a, and let B contain the rest of the processes. Team A is nonempty by construction. Since S_0 is multivalent, team B must also be nonempty.

Condition I of Definition 1 must hold for these values of $v_0, A, B, f_1, \ldots, f_n$. Otherwise, let j be a process index such that $V_{A,j} \cap V_{B,j}$ is nonempty. Then, X has the same value in two configurations that can be reached from S_0 by sequences of steps in which each process takes at most one step, process P_j takes no steps, and the first steps in the two sequences are taken by processes on opposite teams. These two configurations are indistinguishable to P_j. A solo execution by P_j from either of these configurations would therefore lead to the same decision value, contradicting the definitions of teams A and B.

Condition II must also hold. Otherwise, let j be the index of a process on team B such that $v_0 \in V_{A,j}$. Then, some sequence of processes, not including process $P_j \in$

B and starting with a process on team A, could each take a step to arrive at a configuration that P_j cannot distinguish from S_0. A solo execution of P_j from these two configurations would then lead to the same decision, contradicting the fact that $P_j \in B$.

The argument that condition III holds is symmetric. □

This lemma can be combined with a bivalency argument to prove that the conditions for being n-discerning are necessary for solving n-process consensus using RMW objects.

THEOREM 3. *If the n-process consensus problem can be solved using registers and RMW objects of type T, then T must be n-discerning.*

Proof. Suppose there is some protocol for n-process consensus using registers and RMW objects of type T. If only one process is scheduled to take steps in an execution, the process must output its own input value. Thus, any initial configuration where processes begin with different input values is multivalent. It follows that there is a critical configuration S_0 of the protocol. Otherwise, one could produce an execution of infinite length by always scheduling a process whose next step produces a multivalent configuration. No process would ever output a decision in this execution, since any configuration in which some process has produced an output is univalent. This is impossible in a wait-free protocol.

A bivalency argument (see [10]) may be used to show that the next operation performed by any process when the system is in the configuration S_0 must be an operation on the same object, say X, and that X cannot be a register. It follows from Lemma 2 that T is n-discerning. □

The following lemma and Propositions 5 and 6 will be used first in the proof of Theorem 9, which provides a converse to Theorem 3, and again in section 4.

A set $\mathcal{O}$ of objects is said to be capable of solving *team-restricted n-consensus* if there is a partition of the n processes $P_1, \ldots, P_n$ into two nonempty teams A and B such that the consensus problem can be solved using only the objects in the set $\mathcal{O}$, provided all processes on the same team have the same input value.

LEMMA 4. *If a set $\mathcal{O}$ of objects can be used to solve team-restricted $(n+1)$-consensus, then $\mathcal{O}$ can be used to solve team-restricted n-consensus.*

Proof. Any consensus protocol for team-restricted $(n+1)$-consensus may be viewed as a protocol for team-restricted n-consensus by thinking of one of the processes on a team with at least two processes as having failed before executing any of its steps. □

PROPOSITION 5. *Let $n \geq 2$. Suppose O_i is a shared object that can be used along with k registers to solve team-restricted i-consensus for all i, $2 \leq i \leq n$. Then n-consensus can be solved using objects $O_2, \ldots, O_n$ and $k(n-1)$ registers.*

Proof. The proposition will be proved by induction on n. For $n = 2$, the team-restricted form of the consensus problem is identical to the general problem of consensus for two processes.

Let $n > 2$. Suppose the claim holds when the number of processes is less than n. This means that, for all $m < n$, $O_2, \ldots, O_m$ can be used, along with $k(m-1)$ registers, to solve m-consensus. Divide the n processes into two nonempty teams A and B as described in the definition of team-restricted n-consensus. The processes of team A first execute a consensus protocol to agree on one of their input values. If $|A| = 1$, no shared objects are used. Otherwise, team A can agree on an input value using $k(|A|-1)$ registers and the shared objects $O_2, \ldots, O_{|A|}$, by the induction hypothesis. Similarly, the processes of team B agree on one of their input values. If $|B| = 1$, no

shared objects are used. Otherwise, it can be done using $k(|B|-1)$ registers and the shared objects $O_{|A|+1}, \ldots, O_{n-1}$. This is possible, since $O_{i+|A|-1}$ can be used with k registers to solve team-restricted i-consensus for $2 \leq i \leq |B|$, by Lemma 4.

Next, the processes agree on which team's value should be used as the common decision value. The processes execute the team-restricted n-consensus protocol, with each process using the decision value of its team as its input. This can be done using O_n and an additional k registers for a total of $k(n-1)$ registers.

By the inductive hypothesis, the agreement within each team is wait-free. The team-restricted n-consensus protocol used to decide between the two teams' values is also wait-free. So, the entire consensus protocol is wait-free. The protocol satisfies the validity condition, since the value agreed upon by the winning team must be one of the input values of a process on that team, by the inductive hypothesis. The protocol satisfies the consistency condition, since the team-restricted n-consensus protocol must satisfy the consistency condition. □

PROPOSITION 6. *Suppose that one object of type T and k registers can be used to solve team-restricted n-consensus. Then, n-consensus can be solved using $n-1$ objects of type T and $k(n-1)$ registers.*

Proof. This follows immediately from Lemma 4 and Proposition 5. □

LEMMA 7. *An RMW object of type T and two registers can be used to solve team-restricted n-consensus if T is n-discerning.*

Proof. Divide the processes into two nonempty teams A and B, assign a function f_i to each process P_i, and choose v_0 to satisfy the conditions of Definition 1. An algorithm will be constructed for the team-restricted n-consensus problem. The algorithm uses an object X of type T that initially has state v_0 and two registers called R_A and R_B.

Each process P_j first writes its team's common input value into the register R_A, if it belongs to team A, or into the register R_B, if it belongs to team B. The process P_j then performs its assigned operation RMW_{f_j} on X and uses the result of this operation to determine whether a process from team A or from team B was the first to access X.

Without loss of generality, suppose that process P_j belongs to team A. If P_j's *RMW* operation returns the value v_0, then a process from team A was the first to access X. To see why this is true, suppose some processes, starting with a process from team B, did access X before P_j. Let $i_1, \ldots, i_\alpha$ be the indices of these processes. Then $(f_\alpha \circ \cdots \circ f_1)(v_0) = v_0$ and $P_{i_1} \in B$, violating condition III. If P_j's *RMW* operation returns a value different from v_0, the process must be able to deduce which team accessed X first by checking whether the value belongs to $V_{A,j}$ or $V_{B,j}$. These two finite sets are disjoint (by condition I), and contain all possible values of the object X that can be observed by process P_j in this protocol. Once P_j decides whether team A or team B accessed X first, it returns the value in R_A or R_B, respectively.

Each process performs only $O(1)$ steps, so wait-free termination is guaranteed. The protocol satisfies the validity condition, since the winning team's value is written into the team's register before any process from that team can access the object X. The protocol also satisfies the consistency condition: all processes agree on the winning team and return the value of that team's register (which never changes after it is first written, since all processes on the same team have the same input value). □

The following theorems follow immediately from Lemma 7 and Propositions 5 and 6.

THEOREM 8. *The RMW objects $X_2, \ldots, X_n$ can be used, with $2(n-1)$ registers,*

to solve n-consensus if the type of X_i is i-discerning for each i.

THEOREM 9. *If T is an n-discerning RMW type, then there is a protocol for n-consensus that uses $n-1$ objects of type T and $2(n-1)$ registers.*

Theorems 3 and 9 show that an RMW type can be used with registers to solve n-consensus if and only if it is n-discerning. The remainder of this section discusses some consequences of this characterization.

The constructive proof of Theorem 9 provides upper bounds on the complexity of solving the consensus problem using RMW objects and registers. If objects of an RMW type T and registers can be used to solve n-consensus at all, then T is n-discerning by Theorem 3, and the tournament-style algorithm in the proof of Proposition 5 can solve n-process consensus in $O(n)$ steps per process, using $n-1$ objects of type T and $2(n-1)$ registers.

COROLLARY 10. *RMW objects of type T can be used, with registers, to implement any type of object in a system of n processes if and only if T is n-discerning.*

Proof. This follows from Theorems 3 and 9 and the fact that n-consensus objects can be used to obtain an implementation of any shared object in a system of n processes [10]. □

COROLLARY 11. *For RMW types with finite state sets, the following question is decidable: "Given an integer n and an RMW type T, can n-consensus be solved using registers and RMW objects of type T?"*

Proof. The conditions of Definition 1 can be checked for each of the finite number of possible choices of $v_0, A, B, f_1, \ldots, f_n$ in a finite amount of time. □

For every value of n, there is an RMW object type with consensus number exactly n. This can be shown by considering an RMW object that behaves like a sticky bit [20] that gets reset after n accesses.

PROPOSITION 12. *Let $n \geq 2$. Let $V = \{\perp\} \cup (\{A, B\} \times \{1, \ldots, n-1\})$. Let T be the RMW type with state set V whose operations can apply the functions f_A and f_B, where*

$$f_{team}(x) = \begin{cases} (team, 1) & \text{if } x = \perp, \\ \perp & \text{if } x = (team', n-1), \\ (team', z+1) & \text{if } x = (team', z) \text{ and } z < n-1. \end{cases}$$

Then T is n-discerning but not $(n+1)$-discerning.

Proof. First, it is shown that T is n-discerning. Let $v_0 = \perp$, $A = \{P_1\}$, $B = \{P_2, \ldots, P_n\}$, $f_1 = f_A$, and $f_2 = \cdots = f_n = f_B$. Then, each value in $V_{\text{team},j}$ is an ordered pair whose first component is team. The conditions of Definition 1 are therefore clearly satisfied.

Next, it is shown that T is not $(n+1)$-discerning. Consider any choice of $v_0, A, B, f_1, \ldots, f_{n+1}$.

If v_0 is an ordered pair, let z be the second component of v_0. Let P_j be any process. Both $V_{A,j}$ and $V_{B,j}$ contain the element $\perp$, since any sequence of $n-z$ functions applied to v_0 will result in the value $\perp$. This violates condition I in the definition of $(n+1)$-discerning.

If $v_0 = \perp$, let P_j be a process in team B. The set $V_{A,j}$ contains v_0, since any sequence of n functions when applied to $\perp$ results in the value $\perp$. This violates condition II of the definition of $(n+1)$-discerning. □

4. Solving consensus with readable objects. Let T be a readable object type with state set $Q = \bigtimes_{k \in \Gamma} Q_k$. Consider a distributed system with $n \geq 2$ processes, called $P_1, \ldots, P_n$, which communicate via shared objects of type T and

registers. The ability of such a system to solve the consensus problem will be studied in this section.

A readable type T is defined to be *n-discerning* if the set $\{P_1, \ldots, P_n\}$ can be partitioned into two nonempty teams and a single operation can be assigned to each process so that if processes from some subset of $\{P_1, \ldots, P_n\}$ each perform their own operation on an appropriately initialized object X of type T, then each one could determine which team accessed X first, provided that it could see the final state of X. The operation assigned to each process cannot be a *read* operation: it must be possible for the operation to update the state of the readable object. This is formalized as follows.

DEFINITION 13. *The readable type T is called n-discerning if there exist*

- *a state $q_0 \in Q$,*
- *a partition of the set of processes $\{P_1, \ldots, P_n\}$ into two nonempty teams A and B, and*
- *an update operation op_i for $1 \leq i \leq n$*

such that $R_{A,j} \cap R_{B,j} = \emptyset$, for all $j \in \{1, \ldots, n\}$, where $R_{A,j}$ is the set of pairs (r, q) for which there exist distinct process indices $i_1, \ldots, i_\alpha$ including j with $P_{i_1} \in A$ such that if $P_{i_1}, \ldots, P_{i_\alpha}$ each perform their operations (in that order) on an object of type T that is initially in state q_0, P_j gets the result r, and the object ends in state q. The set $R_{B,j}$ is defined similarly as the set of pairs (r, q) for which there exist distinct process indices $i_1, \ldots, i_\alpha$ including j with $P_{i_1} \in B$ such that if $P_{i_1}, \ldots, P_{i_\alpha}$ each perform their operations (in that order) on an object of type T that is initially in state q_0, P_j gets the result r, and the object ends in state q.

It will be shown in Theorems 15 and 18 that readable objects of type T can be used, along with registers, to solve the consensus problem for n processes if and only if T is n-discerning. First, the conditions are shown to be necessary.

LEMMA 14. *If S_0 is a critical configuration of an n-process consensus protocol and the next step by every process is an operation on a readable object X of type T, then T is n-discerning.*

Proof. This proof is similar to the proof of Lemma 2. Let q_0 be the state of X in S_0, and let op_i be the operation performed by P_i in its first step after S_0. Partition the processes into two teams A and B according to their critical values.

To derive a contradiction, suppose these choices for $q_0, A, B, op_1, \ldots, op_n$ do not satisfy Definition 13. Then, there is a pair $(r, q) \in R_{A,j} \cap R_{B,j}$ for some j. There is some sequence $i_1, \ldots, i_\alpha$ of distinct process indices, including j, such that $P_{i_1} \in A$ and if processes $P_{i_1}, \ldots, P_{i_\alpha}$ each perform their next operation, in that order, starting from S_0, process P_j will receive the result r, and the system will end in a configuration S_A where X is in state q. There is some other sequence $k_1, \ldots, k_\beta$ of distinct process indices, including j, such that $P_{k_1} \in B$ and if processes $P_{k_1}, \ldots, P_{k_\beta}$ each perform their next operation, in that order, starting from S_0, process P_j will again receive the result r, and the system will end in a configuration S_B where X is in state q. The configurations S_A and S_B are indistinguishable to P_j, so a solo execution by P_j from either of these two configurations would lead to the same decision value. This contradicts the fact that S_A and S_B are univalent configurations that lead to different decision values.

The operation performed by each process must be an update operation; otherwise the configuration obtained from S_0 by allowing the process to perform its operation could not be distinguished from S_0 by any process on the opposite team. □

Combining this lemma with a bivalency argument yields the following theorem.

THEOREM 15. *If n-process consensus can be solved using registers and objects of a readable type T, then T is n-discerning.*

Proof. A bivalency argument, as in the proof of Theorem 3, shows that any n-consensus protocol that uses registers and objects of type T must have a critical configuration S_0, and that the next operation by each process will be applied to the same object of type T. The theorem then follows from Lemma 14. □

Theorem 15 may be used to establish an upper bound on the consensus number of any (deterministic) type T, whether it is readable or not. If, for some n, the update operations that are permitted for type T do not satisfy Definition 13, then type T cannot be used with registers to solve n-consensus. This is because the addition of a *read* operation to the specification of type T would create a readable type T' that is at least as powerful as type T but has consensus number less than n, by Theorem 15.

The team-restricted n-consensus problem will now be used to provide a converse to Theorem 13.

LEMMA 16. *Let T be an n-discerning readable object type. An object of type T and two registers can be used to solve team-restricted n-consensus.*

Proof. Choose $q_0, A, B, op_1, \ldots, op_n$ to satisfy Definition 13. A protocol will be developed for team-restricted n-consensus that uses one register for each team and one shared object X of type T, initialized to the state q_0. Each process P_j writes its team's common input value into its team's register. It then applies the operation op_j to X and attempts to read the state of X to determine which team accessed X first.

The state set of T has the form $Q = \bigtimes_{k\in\Gamma} Q_k$. Since Γ may be an infinite set, it will first be shown that process P_j can determine the winning team from the values of a finite number of the components. Let $R_{A,j}$ and $R_{B,j}$ be the disjoint sets defined in Definition 13. These sets are finite, since the number of ways to choose $\alpha, i_1, \ldots, i_\alpha$ in the definitions of $R_{A,j}$ and $R_{B,j}$ is bounded by $n \cdot n!$. For $(r, q) \in R_{A,j}$ and $(r', q') \in R_{B,j}$, let $k(q, q')$ be an element of Γ that indexes some state component where q and q' differ, if such a component exists. Let Δ_j be the set of such indices $k(q, q')$ for all possible choices of (r, q) and (r', q'). The number of such choices is finite, so Δ_j is a finite set. Let π_j be the projection function from Q to the set $\bigtimes_{k\in\Delta_j} Q_k$. This projection function discards all components of the state, except for the finite number of components indexed by the elements of Δ_j.

Suppose the sets $\{(r, \pi_j(q)) : (r, q) \in R_{A,j}\}$ and $\{(r', \pi_j(q')) : (r', q') \in R_{B,j}\}$ have an element in common. Then there are two distinct pairs $(r, q) \in R_{A,j}$ and $(r, q') \in R_{B,j}$ such that $q \neq q'$ and $\pi_j(q) = \pi_j(q')$. This is impossible, since $k(q, q') \in \Delta_j$. Thus, process P_j can discern executions in which team A performed the first update from executions in which team B performed the first update, using only the response to its own update operation and the projection $\pi_j(q)$ of the state q of X at any time after P_j's update has been performed.

After performing its update operation, the process P_j reads, one by one, the components of the state that are indexed by Δ_j. The state of X may be updated by other processes while process P_j is performing this scan of the components. Each scan produces a view of the image of the state of X under the projection π_j. Such a view is called *accurate* if it correctly reflects the state of X at some moment during the execution of the scan. If another process performs an update operation during a scan, the resulting view may not be accurate, but any scan that is not interrupted by an update will produce an accurate view.

To ensure that P_j can correctly determine which team accessed X first, the scan of the components of X is repeated $2n - 1$ times. An update of X can occur during

at most $n-1$ of these scans, so at least n of the scans will return an accurate view of the state of X. By Definition 13, P_j can correctly determine which team accessed X first using the information from any accurate scan and the result of its operation op_j. Since a majority of the scans are accurate, P_j can correctly determine which team accessed X first. Process P_j then decides on the value stored in the register belonging to the team that accessed X first.

The validity condition for the consensus problem is satisfied, since every process must agree on the team that accessed X first. The consistency condition is also satisfied, since a process from the winning team must have written its value to its team's register before accessing X. The protocol is wait-free, since each of the $2n-1$ scans reads only a finite number of components of X. □

The following theorems follow immediately from Lemma 16 and Propositions 5 and 6.

THEOREM 17. *Let T_i be an i-discerning readable object type for $2 \leq i \leq n$. Then the n-consensus problem can be solved using one object O_i of each type T_i, together with $2(n-1)$ registers.*

THEOREM 18. *If T is an n-discerning readable object type, then there is a protocol for n-consensus that uses $n-1$ objects of type T and $2(n-1)$ registers.*

This completes the proof that a readable type T can be used with registers to solve n-consensus if and only if T is n-discerning. This characterization has the following consequences.

COROLLARY 19. *Readable objects of type T can be used, along with registers, to implement every other type of object in a system with n processes if and only if T is n-discerning.*

Proof. This follows from Theorems 15 and 18 and the fact that n-consensus objects can be used to obtain an implementation of any shared object in a system of n processes [10]. □

COROLLARY 20. *If the state set of object type T and the set of possible operations on object type T are both finite, then the following question is decidable: "Given a positive integer n and a readable type T, can n-consensus be solved using only objects of type T and registers?"*

Proof. The conditions of Definition 13 can be checked for each of the finite number of choices of $q_0, A, B, op_1, \ldots, op_n$ in a finite amount of time. □

It will now be shown that the addition of a *scan* operation, which reads the entire state atomically, to any readable type T does not increase its power to solve consensus.

COROLLARY 21. *Let T be a readable type. Let T' be a type that is the same as T, except that it allows an additional scan operation that reads the entire state of T. Then T and T' have the same consensus number.*

Proof. Let n be the consensus number of T'. Clearly, the consensus number of T is at most n. By Theorem 15, T' is an n-discerning readable type. Let $q_0, op_1, \ldots, op_n, A$ and B be chosen to satisfy Definition 13 for T'. None of the operations can be a *scan*, since *scan* operations never update the state of an object. Therefore, the choice of $q_0, op_1, \ldots, op_n, A$ and B will also satisfy Definition 13 for type T. By Theorem 18, it is possible to solve n-consensus using objects of type T and registers. □

It can be shown that, for each $n > 1$, there is a readable object type, analogous to the RMW object defined in Proposition 12, that has consensus number n. (See [23] for a detailed description of this object.)

5. Robustness for RMW and readable objects. Jayanti [14] formalized Herlihy's notion of a hierarchy [10] of shared object types and defined a number

of desirable properties for hierarchies, including robustness. A *wait-free hierarchy* classifies object types according to their power to implement one another. Formally, it is a mapping h of object types to the set of levels $\mathbb{N} \cup \{\infty\}$, where a type T is in level n only if objects of type T, together with registers, can be used to implement any other type of object in a system of n processes. If $h(T) = \infty$, then objects of type T and registers can be used to implement any other type of object in a system of n processes for all n. A wait-free hierarchy is *tight* if every object type is mapped to the highest level possible. Thus, if type T is mapped to level n of a tight wait-free hierarchy, there is some type that cannot be implemented using objects of type T and registers in a system of $n + 1$ processes. A wait-free hierarchy is *robust* if no object in any level of the hierarchy can be implemented using a finite number of types of objects from lower levels. In the consensus hierarchy, h^r_m, the level of a type T is the consensus number of T. Jayanti showed that h^r_m is the (unique) tight wait-free hierarchy [14].

Chandra et al. [6] showed that the consensus hierarchy is not robust, if nondeterministic, nonoblivious objects are allowed. Schenk [24] proved that the consensus hierarchy is not robust, even for oblivious objects, if objects with unbounded nondeterminism are allowed. Lo and Hadzilacos [17] improved this, showing that the hierarchy h^r_m restricted to oblivious objects is not robust even when nondeterminism is bounded. Moran and Rappoport [19] showed that the consensus hierarchy is not robust for deterministic nonoblivious objects using the restricted hard-wired binding model. (See Jayanti's survey [13] for a description of binding models, which restrict the ways that processes can access nonoblivious objects.)

Borowsky, Gafni, and Afek [5] claimed that the consensus hierarchy is robust for deterministic objects using a less restrictive binding model. Their paper is quite complex. Here, the characterizations of RMW and readable objects that can solve n-process consensus will be used to provide a concise proof of the robustness of the hierarchy when restricted to deterministic RMW and readable objects.

THEOREM 22. *Let T be a readable or RMW object type. Let $\mathcal{S}$ be a finite set of readable and RMW object types such that $h^r_m(T') < h^r_m(T)$ for each $T' \in \mathcal{S}$. Then an object of type T cannot be implemented using objects whose types are from the set $\mathcal{S}$.*

Proof. Let $n = \max\{h^r_m(T') \mid T' \in \mathcal{S}\} + 1$. This quantity is finite, since $h^r_m(T')$ is less than $h^r_m(T)$ and therefore finite for each $T' \in \mathcal{S}$, and $\mathcal{S}$ is a finite set.

To derive a contradiction, suppose the claim is false. Then, since $h^r_m(T) \geq n$, there is a protocol using objects whose types are from the set $\mathcal{S}$ that solves consensus among n processes. A bivalency argument [10] shows that this protocol has a critical configuration, S_0, and that the next operation taken by any process when the system is in this configuration must be an operation on the same object, X. Let T_X be the type of object X.

First, suppose that T_X is an RMW type. Then T_X is n-discerning, by Lemma 2. It follows from Theorem 9 that $h^r_m(T_X) \geq n$, contradicting the definition of n.

Now suppose that T_X is a readable type. The type T_X is n-discerning, by Lemma 14. By Theorem 18, $h^r_m(T_X) \geq n$, which again contradicts the definition of n. □

This theorem allows the decidability results of Corollaries 11 and 20 to be extended to finite sets of object types. If $\mathcal{S}$ is any finite set of finitely-specified RMW and readable object types, then one can decide whether objects whose types are in $\mathcal{S} \cup \{\text{register}\}$ can be used to solve n-process consensus, by checking whether any of the types in $\mathcal{S}$ are n-discerning.

Acknowledgments. This research forms part of my Ph.D. thesis [23] which was done at the University of Toronto. I am grateful for the guidance of my adviser, Faith Fich. I thank the anonymous PODC referees for their suggestions, and Vassos Hadzilacos, Maurice Herlihy, Wai-Kau Lo, Michael Merritt, and Eric Schenk for helpful discussions.

REFERENCES

[1] Y. Afek, H. Attiya, D. Dolev, E. Gafni, M. Merritt, and N. Shavit, *Atomic snapshots of shared memory*, J. ACM, 40 (1993), pp. 873–890.
[2] J. H. Anderson, *Composite registers*, Distributed Computing, 6 (1993), pp. 141–154.
[3] J. Aspnes and M. Herlihy, *Wait-free data structures in the asynchronous PRAM model*, in Proceedings of the 2nd ACM Symposium on Parallel Algorithms and Architectures, Crete, Greece, 1990, pp. 340–349.
[4] O. Biran, S. Moran, and S. Zaks, *A combinatorial characterization of the distributed* 1*-solvable tasks*, J. Algorithms, 11 (1990), pp. 420–440.
[5] E. Borowsky, E. Gafni, and Y. Afek, *Consensus power makes (some) sense!*, in Proceedings of the 13th ACM Symposium on Principles of Distributed Computing, Los Angeles, CA, 1994, pp. 363–372.
[6] T. Chandra, V. Hadzilacos, P. Jayanti, and S. Toueg, *Wait-freedom vs. t-resiliency and the robustness of wait-free hierarchies*, in Proceedings of the 13th ACM Symposium on Principles of Distributed Computing, Los Angeles, CA, 1994, pp. 334–343.
[7] B. Chor and L. Moscovici, *Solvability in asynchronous environments*, in Proceedings of the 30th IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 1989, pp. 422–433.
[8] M. J. Fischer, N. A. Lynch, and M. S. Paterson, *Impossibility of distributed consensus with one faulty process*, J. ACM, 32 (1985), pp. 374–382.
[9] E. Gafni and E. Koutsoupias, *Three-processor tasks are undecidable*, SIAM J. Comput., 28 (1999), pp. 970–983.
[10] M. Herlihy, *Wait-free synchronization*, ACM Trans. Prog. Lang. Syst., 11 (1991), pp. 124–149.
[11] M. Herlihy and S. Rajsbaum, *The decidability of distributed decision tasks*, in Proceedings of the 29th ACM Symposium on Theory of Computing, El Paso, TX, 1997, pp. 589–598.
[12] M. P. Herlihy and J. M. Wing, *Linearizability: A correctness condition for concurrent objects*, ACM Trans. Prog. Lang. Syst., 12 (1990), pp. 463–492.
[13] P. Jayanti, *Wait-free computing*, in Distributed Algorithms, Lecture Notes in Comput. Sci. 972, Springer-Verlag, Berlin 1995, pp. 19–50.
[14] P. Jayanti, *Robust wait-free hierarchies*, J. ACM, 44 (1997), pp. 592–614.
[15] P. Jayanti and S. Toueg, *Some results on the impossibility, universality and decidability of consensus*, in Distributed Algorithms, Lecture Notes in Comput. Sci. 647, Springer-Verlag, Berlin, 1992, pp. 69–84.
[16] C. P. Kruskal, L. Rudolph, and M. Snir, *Efficient synchronization on multiprocessors with shared memory*, ACM Trans. Prog. Lang. Syst., 10 (1988), pp. 579–601.
[17] W.-K. Lo and V. Hadzilacos, *All of us are smarter than any of us: Wait-free hierarchies are not robust*, in Proceedings of the 29th ACM Symposium on Theory of Computing, El Paso, TX, 1997, pp. 579–588.
[18] N. A. Lynch, *Distributed Algorithms*, Morgan Kaufmann, San Francisco, CA, 1996, chapter 8.
[19] S. Moran and L. Rappoport, *On the robustness of* h_m^r, in Distributed Algorithms, Lecture Notes in Comput. Sci. 1151, Springer-Verlag, Berlin, 1996, pp. 344–361.
[20] S. Plotkin, *Sticky bits and universality of consensus*, in Proceedings of the 8th ACM Symposium on Principles of Distributed Computing, Edmonton, AB, Canada, 1989, pp. 159–175.
[21] E. Ruppert, *Consensus numbers of multi-objects*, in Proceedings of the 17th ACM Symposium on Principles of Distributed Computing, Puerto Vallarta, Mexico, 1998, pp. 211–217.
[22] E. Ruppert, *Consensus numbers of transactional objects*, in Distributed Computing, Lecture Notes in Comput. Sci. 1693, Springer-Verlag, Berlin, 1999, pp. 312–326.
[23] E. Ruppert, *The Consensus Power of Shared-Memory Distributed Systems*, Ph.D. thesis, Department of Computer Science, University of Toronto, Canada, 2000. Available from www.cs.yorku.ca/∼ruppert.
[24] E. Schenk, *The consensus hierarchy is not robust*, in Proceedings of the 16th ACM Symposium on Principles of Distributed Computing, Santa Barbara, CA, 1997, p. 279.