Determining Consensus Numbers

Eric Ruppert
Department of Computer Science
University of Toronto
Toronto, Ontario, Canada, M5S 3G4
`ruppert@cs.utoronto.ca`

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 [15] 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 practical 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.

1 Introduction

A shared memory distributed system consists of a number of asynchronous processes that communicate with one another by performing operations on shared data objects. The power of a system to solve problems depends on the types of shared data objects that are used. 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 [9] showed that the consensus problem, in which each process begins with an input and all non-faulty 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 [9, 12]. 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-consensus can be solved using objects whose types are in $\mathcal{S} \cup \{\text{register}\}$, or ∞ if no such n exists.

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 practical distributed systems are studied: read-modify-write (RMW) objects [15] 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 a RMW object.

A *readable* object O is modelled by an I/O automaton whose state set is the Cartesian product $Q = \bigtimes_{k\in\Gamma} Q_k$, for some index set Γ which need not be finite. For each $k \in \Gamma$, processes may execute the operation $read(O,k)$, 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. It is assumed that each operation can update only a finite number of components of the state. 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.

A RMW object type allows various kinds of *RMW* operations to be performed on it. Let V be the state set of the RMW object. For any function $f : V \to V$, the *RMW* operation defined by f reads the current value, x, of the RMW variable, updates the value to $f(x)$, and returns x. A RMW object type is defined by the set F of functions mapping V to itself that may be applied by the *RMW* operations.

To appear in the 16th ACM Symposium on Principles of Distributed Computing, August 1997, Santa Barbara, California.

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 [9] 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 (but not sufficient) 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. Jayanti and Toueg [14] showed that this question is undecidable for arbitrary object types if they are allowed to have infinite state sets. It will be shown here that a 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, called *n-universality* conditions, which are decidable for types that have finite specifications.

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. Recently, Herlihy and Rajsbaum [10] 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 n-universality conditions are sufficient for the solvability of n-process consensus will also provide upper bounds on the resources required to solve the consensus problem. If a RMW or readable object type T can be used together with registers to solve n-consensus, 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 a RMW object type, each process takes $O(n)$ steps in this protocol. It also follows from the construction of the protocol that n-consensus can be solved using $2(n-1)$ registers and $n-1$ objects, $X_2, \ldots, X_n$, where X_i's type satisfies the conditions for i-universality.

The n-universality conditions are also used to obtain a clear proof that the consensus hierarchy is robust [12] for RMW and readable objects. This means that if n-consensus can be solved using RMW and readable objects of types $T_1, \ldots, T_r$ and registers, then n-consensus can also be solved using objects of type T_i and registers only, for some i. Two teams of researchers, Borowsky, Gafni and Afek [5] and Peterson, Bazzi and Neiger [19] claimed that the consensus hierarchy is robust for all deterministic objects. Full versions of these papers are not yet available. The robustness of the hierarchy means that the consensus number of a set of RMW and readable types can be determined by finding the consensus number of each type separately.

The universality 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.

A more detailed version of this paper is also available [21].

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 [17]). 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 behaviour 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* [11]. This paper deals only with deterministic, linearizable objects. It is assumed that the designer of a protocol may choose the initial states of the shared objects. For simplicity, it is assumed that all objects are *oblivious*: The state transition and response for any operation do not depend on the process that invokes the operation and every process can invoke every operation.

Processes are completely asynchronous and the wait-free model [9] of fault tolerance is used. This means that the algorithms executed by each process must work correctly even if all other processes experience stopping failures or if their speeds vary arbitrarily.

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 non-faulty 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 n-universality conditions are 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. A configuration of a consensus protocol is *x-valent* if all 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 Read-Modify-Write Objects

Consider a distributed system that uses RMW objects and registers for communication between processes. Let V be the set of possible values for the shared RMW variables. Let F be a set of functions mapping V to itself. This section describes the power of a distributed system with $n \geq 2$ processes, $P_1, \ldots, P_n$, to solve consensus, where any process may apply the operation $RMW(X, f)$ to any shared RMW object X for any function $f \in F$. The sets V and F may have arbitrary cardinality; they need not be finite.

Let v_0 be a value in V. Partition the set of processes $\{P_1, \ldots, P_n\}$ into two non-empty 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 set of functions F is called *n-universal* 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 operation $RMW(X, f_i)$ 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. This can be formalized as follows.

Definition 1 *The set F of functions is n-universal if there exist*

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

such that

I. $\forall j, V_{A,j} \cap V_{B,j} = \emptyset$,
II. $\forall P_j \in B, v_0 \notin V_{A,j}$, and
III. $\forall P_j \in A, v_0 \notin V_{B,j}$,

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 $j\}$. The set $V_{B,j}$ is defined similarly.

In this definition, the set $V_{A,j}$ contains the values that P_j can see when it accesses object X, if each process applies its assigned operation to the object X, which has initial value v_0, and the first process to access X was from the set $A \backslash \{P_j\}$. The set $V_{A,j}$ will be empty if P_j is the only process on team A. For each process index j, the conditions in this definition do not imply that $v_0 \notin V_{A,j} \cup V_{B,j}$. For example, if P_j is on team A, it may be the case that v_0 is in $V_{A,j}$.

It will be shown, in Theorems 3 and 9, that RMW objects whose operations are defined by the set F can be used with registers to solve n-consensus if and only if F is n-universal.

Lemma 2 *If S_0 is a critical configuration of an n-process consensus protocol where the next step by process P_i will be $RMW(X, f_i)$ for $1 \leq i \leq n$, then $\{f_1, \ldots, f_n\}$ is n-universal.*

Proof: 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 non-empty by construction. Since S_0 is multivalent, team B must also be non-empty.

Condition I of Definition 1 must hold for these values of $v_0, A, B, f_1, \ldots, f_n$. Otherwise, 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, some sequence of processes, not including some 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. □

Theorem 3 *If the n-consensus problem can be solved using registers and RMW objects, where the RMW operations may apply functions from the set F, then F must be n-universal.*

Proof: Suppose there is some protocol for n-process consensus using registers and RMW objects whose type is defined by the set F. 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, and this is impossible in a wait-free protocol.

A bivalency argument (see [9]) 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 the set F of functions that can be applied to the object X is n-universal. □

The following lemma and Propositions 5 and 6 will be used 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 non-empty 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. Peterson, Bazzi and Neiger [19] call objects that solve team-restricted n-consensus *splitters*.

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, for any $n \geq 2$.*

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 *Take $n \geq 2$. 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 $O_2, \ldots, O_n$, together with $k(n-1)$ 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 2-consensus.

Suppose the claim holds when the number of processes is less than n (where $n > 2$). This means that, for all $m < n$, $O_2, \ldots, O_m$ can be used together with $k(m-1)$ registers to solve m-consensus. Divide the n processes into two non-empty teams A and B as described in the definition of team-restricted n-consensus. The processes of team A will 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. $\square$

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 an array of $n-1$ objects of type T and $k(n-1)$ registers.*

Proof: This follows immediately from Lemma 4 and Proposition 5. $\square$

Lemma 7 *A RMW object and two registers can be used to solve team-restricted n-consensus if the RMW operations can apply functions from an n-universal set.*

Proof: Let F be an n-universal set. Divide the processes into two non-empty 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. Let X be a RMW object that initially has value v_0. Each process P_j first writes its team's common input value into the register regA, if it belongs to team A, or into the register regB, if it belongs to team B.

Each process P_j then performs its assigned operation $RMW(X, f_j)$ 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 regA or regB, 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 satisfies the consistency condition, since 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). $\square$

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, together with $2(n-1)$ registers, to solve n-consensus if the RMW operations on object X_i can apply functions from an i-universal set.*

Theorem 9 *An array of $n-1$ RMW objects, together with $2(n-1)$ registers, can be used to solve n-consensus if the RMW operations can apply functions from an n-universal set.*

The proof of Theorem 9 provides upper bounds on the complexity of solving the consensus problem using RMW objects and registers. Suppose T is a RMW type defined by a set F of functions that can be applied to it. If objects of type T and registers can be used to solve n-consensus at all, then F is n-universal, 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 whose operations are defined by a set F of functions can be used, with registers, to implement any type of object in a system of n processes if and only if F is n-universal.*

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 [9]. $\square$

Corollary 11 *If the set of possible values for the RMW variables is finite, the following question is decidable: "Given an integer n and a set F of functions, can n-consensus be solved using registers and RMW variables if the RMW operations can apply functions from the set F?".*

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. $\square$

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

Proposition 12 *Let $V = \{\perp\} \cup (\{A, B\} \times \{1, \ldots, n-1\})$, for some $n \geq 2$. The set $F = \{f_A, f_B\}$ is n-universal but not $(n+1)$-universal, where*

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

Proof: First, it is shown that the set F is n-universal. Let $v_0 = \perp$, $A = \{P_1\}$, $B = \{P_2, \ldots, P_n\}$, $f_1 = f_A$, and $f_2 = \ldots = f_n = f_B$. 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 F is not $(n+1)$-universal. 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 $(n+1)$-universality.

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 $(n+1)$-universality. □

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 $P_1, \ldots, P_n$, that 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-universal* if a set of n processes can be partitioned into two non-empty teams and a single operation can be assigned to each process so that if any group of processes each perform their own operation on an appropriately initialized object X of type T, then each process in the group 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-universal if there exist*

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

such that

$\forall j, R_{A,j} \cap R_{B,j} = \emptyset,$

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.

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

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-universal.*

Proof: This proof is similar to the proof of Lemma 2. The value q_0 is chosen to be the state of X in S_0 and op_i to be the operation performed by P_i in its first step after S_0. The processes are partitioned 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. □

Theorem 15 *Let T be a readable object type. If n-process consensus can be solved using any combination of registers and objects of type T, then T is n-universal.*

Proof: A bivalency argument (see [9]) 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 is now used to provide a converse to Theorem 13 by showing that n-consensus can be solved using n-universal readable objects and registers.

Lemma 16 *Let T be an n-universal 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. Let $\Gamma' \subseteq \Gamma$ be the set of indices of the components of the state that can be affected if each process P_j applies its operation op_j in any order to an object initially in state q_0.

This set is finite, since each operation affects at most a finite number of components of the object.

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. 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 process reads the components of the state indexed by Γ' one by one. The state of X may be updated by other processes while process P_j is performing this scan of the components. To ensure that P_j can correctly determine which team accessed X first, the scan 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 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-universal 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 *Let T be an n-universal readable object type. An array of $n-1$ objects of type T, together with $2(n-1)$ registers can be used to solve n-consensus.*

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-universal.*

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 [9]. □

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-universal 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 the definition of n-universality 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.

5 Robustness for RMW and Readable Objects

Jayanti [12] formalized Herlihy's notion of a hierarchy [9] 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, together with 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_m^r, the level of a type T is the consensus number of T. Jayanti showed that h_m^r is the (unique) tight wait-free hierarchy [12].

Chandra *et al.* [6] showed that the consensus hierarchy is not robust, if non-deterministic, non-oblivious objects are allowed. Schenk [22] proved that the consensus hierarchy is not robust, even for oblivious objects, if objects with unbounded non-determinism are allowed. Lo and Hadzilacos [16] improved this, showing that the hierarchy h_m^r restricted to oblivious objects is not robust even when non-determinism is bounded. Moran and Rappoport [18] showed that the consensus hierarchy is not robust for deterministic non-oblivious 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 non-oblivious objects.)

Borowsky, Gafni and Afek [5] and Peterson, Bazzi and Neiger [19] claimed that the consensus hierarchy is robust for deterministic objects using less restrictive binding models. These papers are 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. The approach used is similar to that of Peterson *et al.*

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 $h_m^r(T') < h_m^r(T)$ 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 $n = \max\{h_m^r(T') \mid T' \in \mathcal{S}\} + 1$. This quantity is finite, since $h_m^r(T')$ is less than $h_m^r(T)$, 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 $h_m^r(T) \geq n$, there is a protocol using objects whose types are from the set $\mathcal{S}$ that solves consensus among n processes. A bivalency argument [9] 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 a RMW type. The set of functions that can be applied to objects of type T_X is n-universal, by Lemma 2. It follows from Theorem 9 that $h_m^r(T_X) \geq n$, contradicting the definition of n.

Now suppose that T_X is a readable type. The type T_X is n-universal, by Lemma 14. By Theorem 18, $h_m^r(T_X) \geq n$, 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-universal.

Acknowledgements

I am grateful for the guidance and support of my advisor, Faith Fich. I thank the anonymous referees who suggested many improvements to this paper, and Jim Aspnes, Maurice Herlihy, Wai-Kau Lo and Eric Schenk for helpful discussions. This research was supported by an NSERC postgraduate scholarship.

References

[1] Yehuda Afek, Hagit Attiya, Danny Dolev, Eli Gafni, Michael Merritt, and Nir Shavit. Atomic snapshots of shared memory. *Journal of the ACM*, 40(4), pages 873–890, September 1993.

[2] James H. Anderson. Composite registers. *Distributed Computing*, 6(3), pages 141–154, April 1993.

[3] James Aspnes and Maurice Herlihy. Wait-free data structures in the asynchronous PRAM model. In *Proc. 2nd ACM Symposium on Parallel Algorithms and Architectures*, pages 340–349, 1990.

[4] Ofer Biran, Shlomo Moran, and Shmuel Zaks. A combinatorial characterization of the distributed 1-solvable tasks. *Journal of Algorithms*, 11(3), pages 420–440, September 1990.

[5] Elizabeth Borowsky, Eli Gafni, and Yehuda Afek. Consensus power makes (some) sense! In *Proc. 13th ACM Symposium on Principles of Distributed Computing*, pages 363–372, 1994.

[6] Tushar Chandra, Vassos Hadzilacos, Prasad Jayanti, and Sam Toueg. Wait-freedom vs. t-resiliency and the robustness of wait-free hierarchies. In *Proc. 13th ACM Symposium on Principles of Distributed Computing*, pages 334–343, 1994.

[7] Benny Chor and L. Moscovici. Solvability in asynchronous environments. In *Proc. 30th IEEE Symposium on Foundations of Computer Science*, pages 422–433, 1989.

[8] Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2), pages 374–382, April 1985.

[9] Maurice Herlihy. Wait-free synchronization. *ACM Transactions on Programming Languages and Systems*, 11(1), pages 124–149, January 1991.

[10] Maurice Herlihy and Sergio Rajsbaum. The decidability of distributed decision tasks. In *Proc. 29th ACM Symposium on Theory of Computing*, pages 589–598, 1997.

[11] Maurice P. Herlihy and Jeannette M. Wing. Linearizability: A correctness condition for concurrent objects. *ACM Transactions on Programming Languages and Systems*, 12(3), pages 463–492, July 1990.

[12] Prasad Jayanti. On the robustness of Herlihy's hierarchy. In *Proc. 12th ACM Symposium on Principles of Distributed Computing*, pages 145–157, 1993.

[13] Prasad Jayanti. Wait-free computing. In *Distributed Algorithms, 9th International Workshop*, volume 972 of *LNCS*, 1995.

[14] Prasad Jayanti and Sam Toueg. Some results on the impossibility, universality and decidability of consensus. In *Distributed Algorithms, 6th International Workshop*, volume 647 of *LNCS*, pages 69–84, 1992.

[15] Clyde P. Kruskal, Larry Rudolph, and Marc Snir. Efficient synchronization on multiprocessors with shared memory. *ACM Transactions on Programming Languages and Systems*, 10(4), pages 579–601, October 1988.

[16] Wai-Kau Lo and Vassos Hadzilacos. All of us are smarter than any of us: wait-free hierarchies are not robust. In *Proc. 29th ACM Symposium on Theory of Computing*, pages 579–588, 1997.

[17] Nancy A. Lynch. *Distributed Algorithms*, chapter 8. Morgan Kaufmann, 1996.

[18] Shlomo Moran and Lihu Rappoport. On the robustness of h_m^r. In *Distributed Algorithms, 10th International Workshop*, volume 1151 of *LNCS*, 1996.

[19] Gary L. Peterson, Rida A. Bazzi, and Gil Neiger. A gap theorem for consensus types. In *Proc. 13th ACM Symposium on Principles of Distributed Computing*, pages 344–353, 1994.

[20] Serge Plotkin. Sticky bits and universality of consensus. In *Proc. 8th ACM Symposium on Principles of Distributed Computing*, pages 159–175, 1989.

[21] Eric Ruppert. Determining consensus numbers. Technical Report 303/96, Department of Computer Science, University of Toronto, November 1996. Available from http://www.cs.utoronto.ca/~ruppert.

[22] Eric Schenk. The consensus hierarchy is not robust. 1997. In these proceedings.