Consensus Numbers of Multi-Objects

Eric Ruppert

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

Abstract

This paper studies the ability of shared memory distributed systems to solve the wait-free consensus problem if processes are permitted to access more than one shared data object in a single atomic action. Suppose T is any deterministic object type that can be used, with read/write registers, to solve consensus among n processes, with $n > 2$. A multi-object of type T^m consists of a collection of objects of type T, any m of which can be accessed in a single atomic action. It will be shown that a multi-object of type T^m can be used, with registers, to solve consensus among $\Omega(n\sqrt{m})$ processes. Furthermore, if the type T is equipped with operations that allow processes to read its state without altering the state, then the multi-object can be used with registers to solve consensus among $\Omega(nm)$ processes. Neither of these lower bounds can be improved.

1 Introduction

In a shared memory distributed system, processes communicate by accessing shared data objects of various types. Many different types of objects have been proposed as the basis for fault-tolerant distributed systems. The power of a distributed system to solve problems depends on the types of objects that are used. It is therefore important to determine whether the set of object types provided by a distributed system are capable of implementing the shared data structures required by an application. Herlihy [4] showed that objects that solve the n-process consensus problem, where each process begins with an input value and the processes must agree on one of the values, can be used, together with read/write registers, to obtain a wait-free implementation of any other object type for n processes. Thus, the ability of a shared object type to solve the wait-free consensus problem is an important measure of the type's power to implement other types of shared data structures and to solve problems. This leads to the idea of classifying object types according to their consensus number [4, 6]. The *consensus number* of an object type T, denoted $\text{cons}(T)$, is the maximum number of processes that can solve consensus using objects of type T and registers, or infinity if no such maximum exists. This paper studies the increase in the consensus number of an object type when processes are permitted to access several objects simultaneously.

Let T be a shared object type, and let m be a positive integer. Afek, Merritt and Taubenfeld [1] defined a *multi-object* of type T^m to be a collection of objects of type T, indexed by the natural numbers, that can be accessed m at a time. The type T is called the *base type* of the multi-object type T^m. The objects of type T that make up the multi-object are called the *base objects*. An operation on the multi-object consists of at most m pairs of the form (op, i), where op is an operation on the base type T, and i is a natural number that indexes one of the base objects. The indices used in an operation on the multi-object must be distinct from one another. The operation on the multi-object applies the operations on all of the specified base objects as a single atomic action. The operation applied to one base object cannot depend on the result of an operation on another base object that is part of the same multi-object operation.

Multi-objects often simplify the task of programming a distributed system [2]. It would therefore be useful to know whether it is possible to use software to implement multi-objects from the (usually single-access) objects that are provided as primitives of a distributed system. A systematic study of the consensus power of multi-objects is one way to approach this problem.

In this paper, a general lower bound on the consensus numbers of multi-objects is derived. It will be shown in Section 2 that $\text{cons}(T^m) = \Omega(\text{cons}(T) \cdot \sqrt{m})$ for any object type T that has $\text{cons}(T) > 2$. This general lower bound cannot be improved, since $\text{cons}(n\text{-consensus}^m) = \Theta(n\sqrt{m})$ [1]. However, a stronger bound will be derived for the class of readable objects in Section 3. A readable object [10] is one that is equipped with *read* operations that allow processes to read the object's state without altering it. For a readable type T with $\text{cons}(T) > 2$, it will be shown that $\text{cons}(T^m) = \Omega(\text{cons}(T) \cdot m)$. This lower bound cannot be improved: a readable type T_n with $\text{cons}(T_n) = n$ and $\text{cons}(T_n^m) = O(mn)$ will be described in Section 3. In Section 4, it will be shown that no general upper bound on $\text{cons}(T^m)$ can be found, even for the class of readable object types.

The lower bounds on the consensus numbers of multi-objects given in this paper apply only when the base type's consensus number is at least three. The following two examples show why there cannot be general lower bounds on the consensus numbers of multi-objects whose base types have consensus number one or two, even if only readable object types are considered.

As observed by Jayanti and Khanna [7], a multi-object whose base objects always return the same response to all operations will have consensus number one, regardless of the number of base objects that can be accessed simultaneously. This trivial base object type can be viewed as a readable type that has a single possible state and no operations other than the *read* operation.

Now consider the readable object type that has state set $\mathbb{N}$ and is equipped with a *read* operation and a *fetch&increment* operation, which increments the state and returns the old value of the state. Objects of this type are commutative: starting from any state, the state that results from applying any two operations does not depend on the order in which the two operations are performed. A multi-object built from this base type will also be commutative and therefore have consensus number two, regardless of the number of base objects that can be accessed in a single operation [1].

The Model

A shared object can be modelled as an I/O automaton (see [8]). Each operation causes the object to change its state and return a response to the process that invoked the operation. Objects studied in this paper are *deterministic*: the operation and current state of the object uniquely determine the state transition and response. Objects are also *linearizable* [5], so that each operation appears to occur instantaneously at some time between its invocation and its response. The *wait-free* model of fault-tolerance is used in this paper. Thus, non-faulty processes solving a problem must correctly complete their execution even when any number of processes experience stopping failures or when the speeds of the processes vary arbitrarily. An *implementation* of an object of type T from objects $O_1, \ldots, O_N$ consists of an algorithm *apply(op)* for each process that can simulate, using the objects $O_1, \ldots, O_N$, any operation *op* defined for the type T. The implementation should also specify the initial states for $O_1, \ldots, O_N$ to be used for any initial state of the simulated object.

History

Herlihy [4] showed that an array of registers, where processes are allowed to read individual registers or write to m registers in a single atomic action, has consensus number $2m-2$, for $m>1$. Merritt and Taubenfeld [9] showed that n-process consensus can be solved using the multi-object type register^m in the presence of t process failures if and only if $t \leq \max(2m-3, 0)$. Afek, Merritt and Taubenfeld [1] studied the consensus numbers of multi-objects that use, as their base objects, queues, consensus objects and swap objects. (A swap object is equipped with a single operation that writes a new value into the state of the object and returns the old value of the state). They showed that $\text{cons}(queue^2) = \infty$, $\text{cons}(n\text{-consensus}^m) = \Theta(n\sqrt{m})$ and $\text{cons}(\text{swap}^m) = \Theta(\sqrt{m})$. They also observed that any multi-object built from a commutative base type has consensus number at most two. (An object type is *commutative* if, for any pair of operations and any possible state of the object, the state that results from applying the two operations does not depend on the order in which they are applied.) Jayanti and Khanna [7] recently defined a different kind of multi-object, denoted T^*, where *any* finite number of base objects of type T can be accessed in a single atomic action. They showed that, for any type T, $\text{cons}(T^*) \in \{1, 2, \infty\}$. Their proof used the result that $\text{cons}(3\text{-consensus}^m) = \Omega(\sqrt{m})$ [1] to show that $\text{cons}(T^m) = \Omega(\sqrt{m})$ for any base type T with consensus number greater than two. It follows that $\text{cons}(T^*) = \infty$ whenever $\text{cons}(T) > 2$, since $\text{cons}(T^*) = \lim_{m\to\infty} \text{cons}(T^m)$.

2 A General Lower Bound

The following three lemmata will be used in the proof of Theorem 4, below.

Afek, Merritt and Taubenfeld [1] introduced the notion of direct implementations as a tool for studying consensus numbers of multi-objects. Type T_1 *directly implements* type T_2 if an implementation of T_2 can be constructed from objects of type T_1 and registers so that each simulated operation of T_2 can be linearized at the moment the simulation first accesses an object of type T_1. The following lemma follows easily from the definition of direct implementations.

Lemma 1 [1] *If type T_1 directly implements type T_2, then T_1^m directly implements T_2^m for any positive integer m.*

The proof of the next lemma uses a bivalency argument, the proof technique originated by Fischer, Lynch and Paterson [3]. Some standard terminology used in bivalency arguments will be reviewed. A *configuration* of a protocol consists of the state of all shared objects, together with the internal states of all processes. A configuration of a consensus protocol is called *univalent* if all possible executions continuing from that configuration eventually lead to the same output value; otherwise, the configuration is called *multivalent*. A *critical configuration* of a consensus protocol is a multivalent configuration where the next step of any process will move the system into a univalent configuration. The *critical value* of a process P is the output value chosen by all processes if P takes the next step after the critical configuration.

The team(n_1, n_2) problem is defined to be a restricted version of the general consensus problem among n_1+n_2 processes where the processes are divided into two non-empty teams of sizes n_1 and n_2 and all processes on a team receive the same input value. This task can be solved by the following object. The operations permitted on a team(n_1, n_2) *object* are of the form *propose*(i, value). All *propose* operations performed on an object must have distinct arguments i in the range $1 \leq i \leq n_1+n_2$. Furthermore, all *propose* operations with $i \leq n_1$ must have the same value, and all *propose* operations with $i > n_1$ must have the same value. If these conditions are met, the object returns, as the result of each operation, the first value proposed to it.

Lemma 2 *Let $n>1$. Suppose objects of type T can be used, with registers, to solve consensus among n processes. Then, objects of type T can directly implement a team(n_1, n_2) object for some positive integers n_1 and n_2 with $n_1+n_2=n$.*

Proof: Consider a consensus protocol Π for n processes that uses objects of type T and registers. A standard bivalency argument (see [4]) can be used to show that there is some critical configuration of the protocol. Furthermore, after the critical configuration, the next step of each process is applied to a single object X of type T. Let a be the critical value of some process. Let n_1 be the number of processes that have a as their critical values and let $n_2 = n - n_1$.

Divide the processes of protocol Π into two teams. The n_1 processes with critical value a form one team and the remaining n_2 processes form the other team. The direct implementation of the team(n_1, n_2) object will use one register for each team, in addition to the objects used in the protocol Π, which are initialized to the states they have in the critical configuration of Π. When a processor P simulates a *propose*(i, val) operation on the team(n_1, n_2) object, it first writes its input value into the register of the team containing the ith process of Π. It then simulates the ith process of protocol Π, starting from the critical configuration. If the simulated process decides a when it completes protocol Π, P returns the value in the first team's register. Otherwise, P returns the value in the second team's register. This ensures that every simulated operation on the team(n_1, n_2) object returns the value proposed by the first process to access X, which is the first object of type T accessed by each process. Hence, this is a direct implementation of the team(n_1, n_2) object. □

The converse of Lemma 2 is a special case of the following lemma, whose proof makes use of a tournament algorithm.

Lemma 3 [10] *Suppose objects of type T and registers can be used to solve the team(n_1,n_2) problem for some positive integers n_1 and n_2. Then objects of type T can be used, with registers, to solve consensus among $n_1 + n_2$ processes.*

The lower bound may now be proved.

Theorem 4 *For any type T type with consensus number $n > 2$, $cons(T^m) = \Omega(n\sqrt{m})$.*

Proof: By Lemma 2, T can directly implement a team(n_1, n_2) object for some positive integers n_1 and n_2 that sum to n. Without loss of generality, assume that $n_1 \geq n_2$. By Lemma 1, T^m can directly implement a multi-object of type team$(n_1, n_2)^m$. Thus, the consensus number of T^m is at least as large as the consensus number of team$(n_1, n_2)^m$.

Let $g = \lfloor\sqrt{m}\rfloor$ and $N = g \cdot \lfloor n_1/2\rfloor + n_2$. A multi-object of type team$(n_1, n_2)^m$, together with registers, will be used to solve the team$(g \cdot \lfloor n_1/2\rfloor, n_2)$ problem. Both teams will be non-empty, since $n_1 \geq \lceil n/2\rceil \geq \lceil 3/2\rceil$ and $n_2 \geq 1$. It will then follow from Lemma 3 that the type team$(n_1, n_2)^m$ (and therefore T^m) has consensus number at least $N = \Omega(n\sqrt{m})$. The algorithm will be similar to the one used by Afek, Merritt and Taubenfeld [1] to prove that cons(n-consensus^m) $= \Theta(n\sqrt{m})$.

The algorithm will use a multi-object of type team$(n_1, n_2)^m$ with g^2 base objects, arranged into a two-dimensional $g \times g$ array. Since $g^2 \leq m$, processes may access any number of the base objects in the array in a single atomic action.

Divide the N processes into two teams, A and B, containing $g \cdot \lfloor n_1/2\rfloor$ and n_2 processes, respectively. Divide the processes of team A into g groups, $A_1, \ldots, A_g$, each of size $\lfloor n_1/2\rfloor$. Suppose the inputs of processes on teams A and B are a and b, respectively. The algorithm will ensure that every process decides on the value of the process that first accesses the multi-object. Throughout the algorithm, each process will only propose its team's input value to base objects in the array. It will be shown below that the algorithm ensures that processes access each base object correctly: the *propose* operations performed by processes on team A use distinct numbers between 1 and n_1 as their first arguments, and *propose* operations performed by processes on team B use distinct numbers greater than n_1 as their first arguments.

The array of base objects is shown in Figure 1 for the case where $m = 16$. Each of the sixteen blocks represents a team(n_1, n_2) object, which is directly implemented from an object of type T.

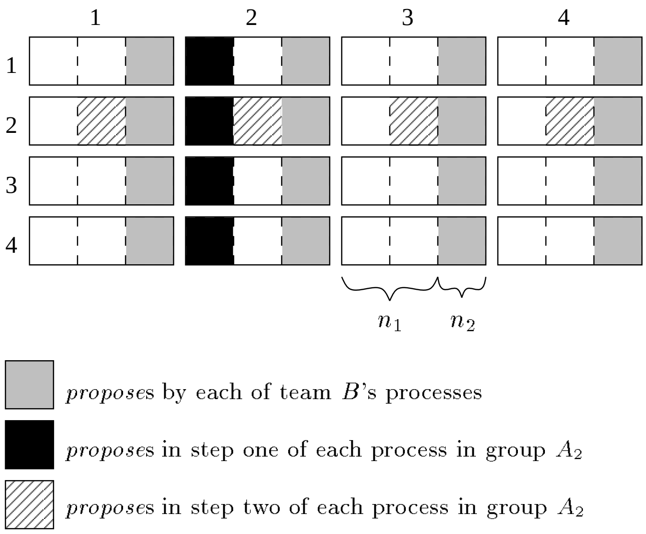

Figure 1: Accesses to the multi-object

The ith process of team B performs the operation *propose*$(n_1 + i, b)$ to all base objects in a single operation. The base objects accessed by each process on team B are indicated by light shading in Figure 1. Only the right portion of each base object is shaded lightly to indicate that the *propose* operations executed by processes on team B use first arguments in the range $n_1 + 1$ to $n_1 + n_2$. If every object returns b, this means that either $a = b$ or a process from team B accessed the multi-object first, so the process outputs b. Otherwise, some base object returns $a \neq b$. In this case, the process outputs a, since some process from team A must have accessed the multi-object first.

The ith process, P, of group A_j first performs the operation *propose*(i,a) to all objects in column j of the array in a single atomic step. If some base object returns a value b that is different from a, process P then performs the operation *propose*$(\lfloor n_1/2\rfloor + i, a)$ on all of the base objects in row j as a second atomic action. The operations in this second step will not affect the output of any of the base objects, since P performs this second step only if all base objects in the array have been previously accessed by a process on team B. The base objects accessed in the first step of each process in group A_2 of team A are indicated by dark shading in Figure 1. The left portion of each of these base objects is shaded to indicate that these *propose* operations use first arguments in the range 1 to $\lfloor n_1/2\rfloor$. The base objects accessed during the second step of each process in group A_2, if it is necessary, are indicated by stripes in Figure 1. The middle portion of each of these base objects is striped to indicate that these *propose* operations use first arguments in the range $\lfloor n_1/2\rfloor$ to n_1.

If every object accessed by the first step of P returns a, then either $b = a$ or a process from group A_j accessed the multi-object before any process from team B, so process P outputs a. Otherwise, it must be the case that a process from team B accessed the multi-object before any process in group A_j and $a \neq b$. Process P then uses the results of its second step to determine whether some process from another group of team A accessed the multi-object first. If every one of the base objects in row j returns b, then a process from team B must have accessed the multi-object first, so process P outputs b. Otherwise, the base object in some column, k, of row j returns a. This means that some process in group A_k accessed the multi-object before any process on team B, so P outputs a.

It will now be verified that this algorithm accesses each base object correctly: no two processes ever perform *propose* operations on any base object using the same first argument. Consider the base object in column k and row j. The $\lfloor n_1/2 \rfloor$ processes from group A_k access it using distinct first arguments in the range 1 to $\lfloor n_1/2 \rfloor$. The $\lfloor n_1/2 \rfloor$ processes from group A_j access it using distinct arguments in the range $\lfloor n_1/2 \rfloor + 1$ to $2 \lfloor n_1/2 \rfloor$. The n_2 processes from team B access it using distinct arguments in the range $n_1 + 1$ to $n_1 + n_2$. No other processes access the base object. Thus, the base objects must always provide the correct solution to the team(n_1, n_2) problem. Therefore, the algorithm that uses the multi-object to solve the team$(g \cdot \lfloor n_1/2 \rfloor, n_2)$ problem is correct. □

The bound given in Theorem 4 cannot be improved, since the consensus number of the n-consensus^m multi-object type is $\Theta(n\sqrt{m})$ [1]. However, in the next section, a better bound will be obtained for the class of readable object types.

It will now be shown that the power of types with finite consensus numbers increase when processes are permitted to access sixteen of the base objects in a single atomic operation.

Proposition 5 *Let T be an object type with $2 < cons(T) < \infty$. Objects of type T and registers cannot be used to implement the type T^{16}.*

Proof: It was shown, in the proof of Theorem 4, that objects of type T^m, together with registers, can solve consensus among $\lfloor \sqrt{m} \rfloor \cdot \lfloor n_1/2 \rfloor + n_2$ processes, for some positive integers n_1 and n_2 with $n_1 \geq n_2$ and $n_1 + n_2 = \text{cons}(T)$. If $n_1 = 2$, then $\text{cons}(T^{16}) \geq 4 + n_2 > \text{cons}(T)$. If $n_1 > 2$, then $\text{cons}(T^{16}) \geq 4 \cdot \lfloor n_1/2 \rfloor + n_2 \geq 2(n_1 - 1) + n_2 = \text{cons}(T) + n_1 - 2 > \text{cons}(T)$. □

3 Readable Multi-Objects

In this section, a better lower bound on the consensus numbers of multi-objects is obtained for the large class of readable object types. An object is readable if processes may read the object's state without altering it. For many object types, a single read operation will return the entire state of the object. However, if the object is large, it may be more reasonable to read its state one component at a time. Formally, a *readable* object O is modelled by an I/O automaton whose state set is a Cartesian product $Q = \bigtimes_{k \in \Gamma} Q_k$, where Γ is an index set and Q_k is a set for each $k \in \Gamma$. The sets Γ and Q_k 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 without changing the state of O. In addition to the *read* operations, the object may have an arbitrary set of other (deterministic) operations defined on it. These other operations are called *update* operations. It is assumed that each update operation can change only a finite number of components of the state. However, the set of components that are changed by an operation may depend on the current state of the object. A multi-object whose base type is readable is, itself, also a readable 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 one-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 unbounded 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 readable object is called *n-universal* [10] 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. A more formal definition is given below.

Definition 6 *A readable type T is 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 follows from this definition that a readable type T is n-universal if and only if the team(n_1, n_2) problem can be solved, for some positive integers n_1 and n_2 that sum to n, by a protocol that uses a single object of type T and two registers, in which each process performs exactly one update operation on the object of type T. (See [10] for more details.)

It was shown in [10] that the property of being n-universal characterizes the readable object types that have consensus number at least n.

Theorem 7 [10] *A readable type can be used with registers to solve consensus among n processes if and only if it is n-universal.*

This characterization will now be used to prove a better lower bound on the consensus numbers of readable multi-objects. The proof is similar to the proof of Theorem 4. First, the main difference between the two proofs is described. The proof of Theorem 4 used a multi-object that consisted of a $\lfloor \sqrt{m} \rfloor \times \lfloor \sqrt{m} \rfloor$ array of base objects. All of the elements in column k of the array were used to determine whether a process from group A_k or a process from team B

accessed the object first. The $\lfloor\sqrt{m}\rfloor$ base objects in the column were required to ensure that all processes could access this information. When the objects are readable, however, it will be shown that all processes can access the information from a single base object. This allows the square array of base objects to be replaced by a one-dimensional array consisting of a single row. The width of the array can then be increased from $\lfloor\sqrt{m}\rfloor$ to m, while still allowing processes on team B to access every base object in a single operation. This leads to the following improved bound for readable multi-objects.

Theorem 8 *For any readable object type T with consensus number $n > 2$, $cons(T^m) = \Omega(nm)$.*

Proof: By Theorem 7, type T is n-universal. Choose $A, B, op_1, \ldots, op_n$ and q_0 to satisfy the definition of n-universality for T. Without loss of generality, assume that processes are labelled so that $A = \{P_1, \ldots, P_{|A|}\}$ and $B = \{P_{|A|+1}, \ldots, P_n\}$, where team A has at least as many processes as team B. Hence, $|A| \geq \lceil n/2 \rceil$.

Let $N = m(|A| - 1) + |B|$. It will be shown that T^m, which is also a readable type, is N-universal. It will then follow from Theorem 7 that $\text{cons}(T^m) \geq N = (m-1)(|A|-1) + |A| + |B| - 1 \geq (m-1)(\lceil n/2 \rceil - 1) + n - 1 = \Omega(nm)$. Primes will be used to distinguish the variables in the definition of N-universality for type T^m from the variables used in the definition of n-universality for type T.

Values for $A', B', op'_1, \ldots, op'_N$ and q'_0 must be chosen so that each process P'_j can tell, from the result of applying op'_j, together with the state of the multi-object at any later time, which team accessed the multi-object first. Let the team A' consist of $m(|A|-1)$ processes, and B' consist of the remaining $|B|$ processes. Both teams are non-empty, since $|A'| = m(|A| - 1) \geq m(\lceil n/2 \rceil - 1) \geq m$ and $|B'| = |B| > 0$. Divide the processes on team A' into m groups, $A'_1, \ldots, A'_m$, each of size $|A| - 1$.

Only m base objects, $O_1, \ldots, O_m$, will be used. In the initial state q'_0 of the multi-object, each of the base objects is in state q_0. Each process of team B' accesses each of the m base objects in the same way that the corresponding process on team B accesses an object of type T when determining whether team A or B accessed it first. Each process in group A'_i of team A' accesses only the base object O_i. It does so in the same way that the corresponding process on team A accesses an object of type T when determining whether team A or team B accessed it first. The base object O_i is used to determine whether a process from group A'_i accessed the multi-object before any process from team B'. It will be shown below that this information can be accessed by all processes (and not only those in $A'_i \cup B$, which are actually performing updates to O_i) because only $n - 1$ processes update the object O_i. Processes can determine which team accessed the multi-object first because they can determine whether or not a process from group A'_i accessed it before a process from team B, for each i.

The assignment of operations to processes will now be specified more precisely. The jth process of group A'_i performs the operation op_{j+1} on the base object O_i. The jth process of team B' performs the operation $op_{j+|A|}$ on all m of the base objects. The possible sequences of updates that can be performed on the base object O_i are exactly those that can be performed on an object of type T when some subset of processes from $\{P_2, \ldots, P_n\}$ perform their update operations as described in the definition of n-universality.

It will now be shown that these choices of teams, initial state and operations satisfy the definition of N-universality for T^m. To derive a contradiction, suppose the definition of N-universality is not satisfied. Then, for some j, there is a pair $(r, q) \in R'_{A',j} \cap R'_{B',j}$. This implies that there are two sequences of distinct processes, one beginning with a process in some group A'_i of team A' and the other beginning with a process of team B', such that the final state of the multi-object is q if either sequence of processes perform their update operations on the multi-object. Let $op_{k_1}, \ldots, op_{k_\alpha}$ and $op_{l_1}, \ldots, op_{l_\beta}$ be the sequences of operations that are performed on O_i by these two sequences of updates to the multi-object. Both of these sequences leave object O_i in the same state. Both $k_1, \ldots, k_\alpha$ and $l_1, \ldots, l_\beta$ are sequences of distinct indices chosen from the set $\{2, \ldots, n\}$ with $P_{k_1} \in A$ and $P_{l_1} \in B$. Now consider the two sequences of operations obtained by adding a final operation, performed by P_1. The two sequences $op_{k_1}, \ldots, op_{k_\alpha}, op_1$ and $op_{l_1}, \ldots, op_{l_\beta}, op_1$, when performed on an object of type T initialized to the state q_0, will both leave the object in the same state and return the same response to the operation op_1. This implies that $R_{A,1} \cap R_{B,1}$ is non-empty, contradicting the definition of n-universality for type T. Thus, type T^m is N-universal. □

The constant in Proposition 5 can be improved for the class of readable objects, as described in the following proposition.

Proposition 9 *Let T be a readable object type with $2 < cons(T) < \infty$. Objects of type T and registers cannot be used to implement T^3.*

Proof: Let n be the consensus number of type T. It was shown in the proof of Theorem 8 that objects of type T^m, together with registers, can solve consensus among $(m-1)(\lceil n/2 \rceil - 1) + n - 1$ processes. Thus, $\text{cons}(T^3) \geq n + 2 \cdot \lceil n/2 \rceil - 3 > n$, since $\lceil n/2 \rceil > 3/2$ when $n > 2$. □

The following proposition shows that the lower bound on $\text{cons}(T^m)$ for readable types T given in Theorem 8 cannot be improved. Consider the readable object type T_n defined in [10], which behaves like a readable version of a binary consensus object that gets reset to the state $\perp$ once every $n + 1$ operations. The state set of T_n is the set $Q = \{\perp\} \cup (\{A, B\} \times \{1, \ldots, n\})$. The object has a *read* operation that returns the entire state of the object, as well as the following two propose operations.

```
proposeA(X)
   if X = ⊥ then X ← (A, 1)
   else % X is an ordered pair
      if X's second component is n then X ← ⊥
      else increment the second component of X
   return ack
end proposeA
```

```
proposeB(X)
   if X = ⊥ then X ← (B, 1)
   else % X is an ordered pair
      if X's second component is n then X ← ⊥
      else increment the second component of X
   return ack
end proposeB
```

It was shown in [10] that the consensus number of T_n is n. The following proposition demonstrates that the bound given in Theorem 8 is tight.

Proposition 10 *Let $n > 1$. For the readable object type T_n defined above, $cons(T_n^m) = \Theta(nm)$.*

Proof: The lower bound $\text{cons}(T_n^m) = \Omega(nm)$ follows from Theorem 8. Here, it is shown that $\text{cons}(T_n^m) = O(nm)$.

Suppose the type T_n^m can be used to solve consensus among N processes. It will be shown that N is at most $2m(n+1)$. By Theorem 7, T_n^m is an N-universal readable type. Choose $A, B, op_1, \ldots, op_N$ and q_0 to satisfy the definition of N-universality for T_n^m.

Let P_i and P_j be any two processes on teams A and B, respectively. Let $\mathcal{O}_{ij}$ be the set of base objects of the multi-object that are updated by both op_i and op_j. First, it will be shown that there is some base object in $\mathcal{O}_{ij}$ that is updated by at most $n+2$ of the operations $op_1, \ldots, op_N$. To derive a contradiction, suppose each base object in $\mathcal{O}_{ij}$ is updated by at least $n+3$ of the operations. Consider the following two executions of the operations $op_1, \ldots, op_N$ on a multi-object initially in state q_0. In the first execution, op_i is performed first, op_j is performed second, and then the remaining $N-2$ operations are performed one at a time in some fixed order. In the second execution, op_j is performed first, op_i is performed second, and then the remaining $N-2$ operations are performed in the same order as in the first execution.

Let O be one of the base objects of the multi-object. Let s_1 be the state of O just before the final operation of the first execution is performed. Let s_2 be the state of O just before the final operation of the second execution is performed. If $O \notin \mathcal{O}_{ij}$ then $s_1 = s_2$: the relative order of op_i and op_j is unimportant, since at most one of them updates O. Now, consider the case where $O \in \mathcal{O}_{ij}$. Both executions perform the same number of operations on O. Therefore $s_1 = \perp$ if and only if $s_2 = \perp$. Furthermore, if s_1 and s_2 are ordered pairs, then their second components must be equal. In the latter case s_1 is also equal to s_2, since the first component of O depends only on the last n update operations performed on O, and at least n operations are performed after op_i and op_j but before the last operation in both executions.

Thus, the last process to take a step will receive the same response from the multi-object in both executions and the multi-object will be left in the same state at the end of the two executions. This contradicts the fact that $A, B, op_1, \ldots, op_N$ and q_0 were chosen to satisfy the definition of N-universality. So, it must be the case that some base object in $\mathcal{O}_{ij}$ is updated by at most $n+2$ of the operations $op_1, \ldots, op_N$.

Let P_i be any process on team A. Consider the set S of base objects that are updated by op_i and by at most $n+1$ of the other operations $op_1, \ldots, op_{i-1}, op_{i+1}, \ldots, op_N$. Since op_i updates at most m base objects, $|S| \leq m$. It was shown that $\mathcal{O}_{ij} \cap S \neq \emptyset$ for any $P_j \in B$, so every process on team B updates one of the base objects in S. Thus, $|B| \leq |S|(n+1) \leq m(n+1)$. Similarly, $|A| \leq m(n+1)$. Therefore, the total number of processes is $N = |A| + |B| \leq 2m(n+1) = O(mn)$. □

4 The Toggle Object

In this section, it is shown that a general upper bound for the consensus numbers of multi-objects cannot be found, even for readable object types. This will be done using a readable type called toggle. A toggle object has state set $\{\perp, 0, 1\}$. In addition to the *read* operation, it has two operations *toggle0* and *toggle1*, which return only the result *ack*. If the state is 0, the *toggle* operations change it to 1, and if the state is 1, then the *toggle* operations change it to 0. If the state is $\perp$, the *toggle0* operation changes it to 0 and the *toggle1* operation changes it to 1. The state transitions are shown in Figure 2.

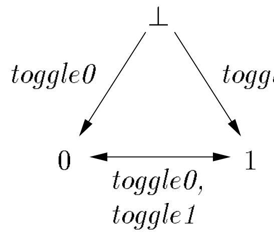

Figure 2: State transitions of a toggle object.

Proposition 11 *For the readable type toggle, defined above, cons(toggle) $= 1$ but cons(toggle^2) $= \infty$.*

Proof: First it is shown that toggle is not 2-universal. To derive a contradiction, suppose toggle does satisfy the definition of 2-universality. Without loss of generality, suppose $A = \{P_1\}$ and $B = \{P_2\}$. If the two processes are assigned different toggle operations, say $op_1 =$ *toggle0* and $op_2 =$ *toggle1*, then $(ack, 0) \in R_{A,1} \cap R_{B,1}$, which contradicts Definition 6. If both processes are assigned the same toggle operation, say *toggle0*, then $R_{A,1} \cap R_{B,1}$ contains $(ack, 0)$ if q_0 is 0, and contains $(ack, 1)$ if q_0 is 1 or $\perp$, and this is again a contradiction.

It will now be shown that the readable type toggle^2 is n-universal, for any n, and therefore has infinite consensus number, by Theorem 7. Only three base objects O_1, O_2 and O_3 will be used. Each of the three base objects is initialized to the state $\perp$ in q_0. The n processes are partitioned into two non-empty teams A and B arbitrarily. Processes on team A perform a *toggle0* operation on O_1 and O_2. Processes on team B perform a *toggle1* operation on O_2 and O_3. At any time after the first toggle has been performed, one can determine which team went first from the state of the multi-object as follows. If the base object O_1 is in state $\perp$, then team B went first. If the base object O_3 is in state $\perp$, then team A went first. Otherwise, the state of each of the three base objects will be in $\{0, 1\}$. The first team to take a step can be determined from the parity of these three states: team A went first if and only if zero or two of the base objects are in state 1. Thus, the definition of n-universality is satisfied. □

In the proof above, it was shown that the multi-object toggle^2 can solve consensus among any number of processes even if all base objects must be initialized to the same starting state and all of the operations performed on base objects during one multi-object operation must be the same.

It follows easily from Proposition 11 that, for any n, one cannot obtain an upper bound on $\text{cons}(T^m)$ as a function of m that applies to the class of object types T with consensus number n, even if only readable objects are considered. This can be seen by considering a readable object type, denoted toggle $\vee\, T_n$, which can behave either as a toggle object or as a an object of type T_n (defined in Section 3), depending on the initial state of the object. This type's state set will be the disjoint union of the state sets of the types toggle and T_n. Because the consensus hierarchy is robust for readable objects [10], $\text{cons}(\text{toggle} \vee T_n) = \max(\text{cons}(\text{toggle}), \text{cons}(T_n)) = n$. However, since an object of type toggle$\vee T_n$ can simulate a toggle object, $\text{cons}((\text{toggle} \vee T_n)^m) = \infty$ for all $m > 1$.

Acknowledgements

I am grateful to my advisor, Faith Fich, for all of her help during the development of this paper. I thank Yehuda Afek, Wai-Kau Lo, Michael Merritt, Steven Myers and Gadi Taubenfeld for helpful discussions and the PODC reviewers for suggesting improvements. This research was supported by a postgraduate scholarship from the Natural Sciences and Engineering Research Council of Canada.

References

[1] Yehuda Afek, Michael Merritt, and Gadi Taubenfeld. The power of multi-objects. In *Proc. 15th ACM Symposium on Principles of Distributed Computing*, pages 213–222, 1996.

[2] Yehuda Afek, Michael Merritt, Gadi Taubenfeld, and Dan Touitou. Disentangling multi-object operations. In *Proc. 16th ACM Symposium on Principles of Distributed Computing*, pages 111–120, 1997.

[3] 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.

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

[5] 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.

[6] Prasad Jayanti. Robust wait-free hierarchies. *Journal of the ACM*, 44(4), pages 592–614, July 1997.

[7] Prasad Jayanti and Sanjay Khanna. On the power of multi-objects. In *Distributed Algorithms, 11th International Workshop*, 1997.

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

[9] Michael Merritt and Gadi Taubenfeld. Atomic m-register operations. In *Distributed Algorithms, 5th International Workshop*, volume 579 of *LNCS*, pages 289–294, 1991.

[10] Eric Ruppert. Determining consensus numbers. Technical Report 303/96, Department of Computer Science, University of Toronto, 1996. Available from http://www.cs.utoronto.ca/~ruppert. A condensed version appeared in *Proc. 16th ACM Symposium on Principles of Distributed Computing*, pages 93-99, 1997.