2-28-1997

On the Power of Multi-Objects

Prasad Jayanti
Dartmouth College

Sanjay Khanna
Dartmouth College

On the Power of Multi-Objects*

Prasad Jayanti
Sanjay Khanna

Technical Report PCS-TR97-311

3/97

*Work supported by NSF grant CCR-9410421
and Dartmouth College Startup Grant

On the Power of Multi-Objects*

Prasad Jayanti† Sanjay Khanna‡

Dartmouth College Computer Science
Technical Report Number PCS-TR97-311

February 28, 1997

Abstract

In the standard "single-object" model of shared-memory computing, it is assumed that a process accesses at most one shared object in each of its steps. A (more powerful) variant is the "multi-object" model in which each process may access multiple shared objects atomically in each of its steps. In this paper, we present results that relate the synchronization power of a type in the multi-object model to its synchronization power in the single-object model.

Although the types `fetch&add` and `swap` have the same synchronization power in the single-object model, Afek, Merrit, and Taubenfeld showed that their synchronization powers differ in the multi-object model [AMT96]. We prove that this divergence phenomenon is exhibited *only* by types at levels 1 and 2; all higher level types have the same unbounded synchronization power in the multi-object model.

This paper also identifies *all* possible relationships between a type's synchronization power in the single-object model and its synchronization power in the multi-object model.

1 Introduction

A shared-memory system consists of asynchronous processes and typed shared objects. An execution of such a system is an interleaving of the steps of individual processes. In the commonly studied model, it is assumed that a process accesses at most one shared object in each of its steps. We call this the *single-object model*. A variant (and a more powerful) model is the *multi-object model* in which each process may access multiple shared objects *atomically* in each of its steps. Specifically, each step of a process P corresponds to the following sequence of actions, all of which occur together atomically: (i) based on its present state, P determines the number m of objects to access, the identities $O_1, \ldots, O_m$ of (distinct)

*Work supported by NSF grant CCR-9410421, and Dartmouth College Startup grant.

†6211 Sudikoff Lab for Computer Science, Dartmouth College, Hanover, NH 03755

‡6211 Sudikoff Lab for Computer Science, Dartmouth College, Hanover, NH 03755

objects to access, and the operations $oper_1, \ldots, oper_m$ to apply to these objects, (ii) for all $1 \leq i \leq m$, P applies $oper_i$ on O_i and receives O_i's response res_i, and (iii) P makes a transition to a new state, where the new state depends on the responses $res_1, \ldots, res_m$ and the previous state of P. This model was studied earlier by Herlihy [Her91] and by Merritt and Taubenfeld [MT94] in the context of shared-memories that consisted only of registers, and was recently explored further by Afek, Merritt, and Taubenfeld [AMT96]. In this paper, we present results that relate the synchronization power of a type in the multi-object model to its synchronization power in the single-object model.

Let T^m denote a shared-memory consisting of infinitely many objects of type T such that in each of its steps a process may access any of at most m objects atomically. Let T^* denote a shared-memory consisting of infinitely many objects of type T such that in each of its steps a process may access any finite number of objects atomically. Since consensus is universal [Her91], the extent to which consensus is implementable in a shared-memory is a reasonable measure of the synchronization power of that shared-memory. Accordingly, as in [AMT96], we define $Con(T^m)$ as the maximum number of processes for which a consensus object can be implemented in shared-memory T^m; if there is no such maximum, $Con(T^m) = \infty$. $Con(T^*)$ is similarly defined. Notice that $Con(T^1)$, which we will simply write as $Con(T)$, denotes the synchronization power of T in the single-object model.

Afek, Merrit, and Taubenfeld observed the following "divergence phenomenon" as we shift from the single-object model to the multi-object model [AMT96]. Although the types `fetch&add` and `swap` have the same synchronization power in the single-object model ($Con($`fetch&add`$) = Con($`swap`$) = 2$ [Her91]), their synchronization powers differ in the multi-object model: $Con($`fetch&add`$^*)$ is still 2 while $Con($`swap`$^*)$ is ∞. Thus, the multi-object model enhances the power of `swap`, but not of `fetch&add`, despite the fact that the two types have the same power in the single-object model. The same divergence phenomenon also occurs for certain types at level 1.[1] Specifically, consider the type `trivial` which supports a single operation that always returns the same response. Clearly $Con($`trivial`$) = 1$. It is well-known that $Con($`register`$)$ is also 1 [CIL94, DDS87, LA87, Her91]. Yet, $Con($`trivial`$^*) = 1$ and $Con($`register`$^*) = \infty$ [Her91].

The main result of this paper is that the divergence phenomenon described above is exhibited *only* by types at levels 1 and 2. Specifically, we prove that if $Con(T) \geq 3$, then $Con(T^*) = \infty$. In other words, the synchronization power of *all* types at levels 3 or higher is enhanced to the fullest degree by the multi-object model. Thus, it is not a coincidence that the types which appeared above in the examples of the divergence phenomenon — `fetch&add`, `swap`, `trivial`, and `register` — are at levels 1 or 2.

We also present the following results for types at levels 1 and 2. If $Con(T) = 1$, we show $Con(T^*) \in \{1, 2, \infty\}$. Further, we show that there are types in all of these three categories. If $Con(T) = 2$, we show $Con(T^*) \in \{2, \infty\}$. Further, there are types in both these categories, as was demonstrated in [AMT96] with `fetch&add` and `swap`.

Figure 1 summarizes all possible ways in which $Con(T)$ and $Con(T^*)$ are related. There is an "X" in the table element at row labeled i and column labeled j if and only if there

[1] We refer to a type T as being at level k if $Con(T) = k$.

$Con(T)$	$Con(T^*)$		
	1	2	∞
1	X `trivial`	X `blind-increment`	X `register` [Her91]
2		X `fetch&add` [AMT96]	X `swap` [AMT96]
≥ 3			X

Figure 1: All possible ways in which $Con(T)$ and $Con(T^*)$ are related

is a type T such that $Con(T) = i$ and $Con(T^*) = j$. The table also includes example types for the different possible relationships. As the table indicates, this paper presents a complete picture of how the synchronization power of a type is affected by a shift from the single-object model to the multi-object model.

2 Preliminaries

The concepts in this section are not new and our treatment is therefore informal.

2.1 Definitions of n-`consensus` and n-`IDconsensus`

An object of type n-`consensus` can be accessed by at most n processes. Each process may invoke *propose 0* or *propose 1*. The sequential specification is as follows: all operations return the value first proposed.

An object of type n-`IDconsensus` can be accessed by at most n processes. Let $P_0, P_1, \ldots, P_{n-1}$ be the names of these processes. Process P_i may only invoke *propose* i. The sequential specification is as follows: all operations return the value first proposed.

Using a single n-IDconsensus object, $P_0, P_1, \ldots, P_{n-1}$ can determine a winner among them as follows: each P_i proposes i to the object; if the object's response is j, P_i regards P_j as the winner.

As in the above, we write the type names in the typewriter font. Thus, "`register`" denotes a type and "register" (in non-typewriter font) denotes an object.

2.2 Direct implementation

Let X and Y be types. Informally, X^m *implements* Y^n if there is a wait-free simulation of shared-memory Y^n using shared-memory X^m. (Recall that X^m denotes a shared-memory consisting of infinitely many objects of type X such that in each of its steps a process may access any of at most m objects atomically.) Each operation on the (implemented) shared-

memory Y^n is simulated by executing (possibly many) operations on the shared-memory X^m.[2]

Afek, Merritt, and Taubenfeld introduced the notion of "direct implementation" [AMT96]. X^m *directly implements* Y^n if there is an implementation of Y^n from X^m such that the linearization of every operation *op* on the shared-memory Y^n can always be placed at the first access to X^m during the simulation of *op* [AMT96].

We write $X^m \rightarrow Y^n$ to denote that X^m implements Y^n and $X^m \xrightarrow{di} Y^n$ to denote that X^m directly implements Y^n.

The transitivity of $\xrightarrow{di}$ follows easily from definitions and is therefore stated below without proof.

Proposition 2.1 *The relation* $\xrightarrow{di}$ *is transitive:* $X^m \xrightarrow{di} Y^n$ *and* $Y^n \xrightarrow{di} Z^p$ *implies* $X^m \xrightarrow{di} Z^p$.

2.3 Previous results

Here we state results from [AMT96] that will be used in this paper.

Theorem 2.1 ([AMT96]) *Let* X *and* Y *be any types.* $X^p \xrightarrow{di} Y^q$ *implies* $X^{pm} \xrightarrow{di} Y^{qm}$, *for all* $m > 0$.

Theorem 2.2 ([AMT96]) *Let* X *and* Y *be any types.* $X^p \xrightarrow{di} Y$ *implies* $Con(X^{pq}) \geq Con(Y^q)$, *for all* $p, q > 0$.

The following is a special case of a more general theorem from [AMT96].

Theorem 2.3 ([AMT96]) $\text{Cons}(3\text{-consensus}^m) \geq \sqrt{2m}$.

3 Multi-object theorem for types at level 3 or higher

In this section we prove that if $Con(T) \geq 3$, then $Con(T^*) = \infty$. This result follows from two intermediate results derived in Sections 3.2 and 3.3 and the results of Afek, Merritt, and Taubenfeld stated above. We conclude the result in Section 3.4 and, in Section 3.5, sketch an alternative proof for the same result. We begin with our notation for describing implementations of n-`consensus` and n-`IDconsensus`.

[2]Sometimes it is assumed that the implementation also has access to registers. We do not make such an assumption in this paper.

3.1 Notation for describing consensus implementations

Informally, the following two elements constitute an implementation of an n-consensus object $\mathcal{O}$, shared by processes $P_0, \ldots, P_{n-1}$, in shared memory T^k: (i) the objects $O_1, O_2, \ldots, O_m$ that $\mathcal{O}$ is implemented from, and (ii) the access procedures `Propose`$(P_i, v, \mathcal{O})$, for $i \in \{0, 1, \ldots, n-1\}$ and $v \in \{0, 1\}$. To apply a *propose* v operation on $\mathcal{O}$, P_i calls and executes the access procedure `Propose`$(P_i, v, \mathcal{O})$. The access procedure specifies how to simulate the operation on $\mathcal{O}$ by executing operations on $O_1, O_2, \ldots, O_m$, accessing at most k of these objects in any one step. The return value from the access procedure is deemed to be the response of $\mathcal{O}$ to P_i's operation. We refer to $O_1, O_2, \ldots, O_m$ as *base objects* of $\mathcal{O}$. The *space complexity* of the implementation is m, the number of base objects required in implementing $\mathcal{O}$.

Similarly, an implementation of an n-IDconsensus object $\mathcal{O}$, shared by processes $P_0, \ldots, P_{n-1}$, in shared memory T^k is constituted by: (i) the objects $O_1, O_2, \ldots, O_m$ that $\mathcal{O}$ is implemented from, and (ii) the access procedures `Propose`$(P_i, i, \mathcal{O})$, for $i \in \{0, 1, \ldots, n-1\}$ (recall that process P_i may only propose i on $\mathcal{O}$).

3.2 Directly implementing n-`consensus` from n-`IDconsensus`^{2}

In this section, we show that n-`IDconsensus`^{2} *directly* implements n-`consensus`. Let $\mathcal{O}$ denote the n-consensus object to be implemented. Let $P_0, \ldots, P_{n-1}$ denote the processes that share $\mathcal{O}$, and let v_i be the value that P_i wishes to propose to $\mathcal{O}$. For ease of exposition, we develop the implementation in steps. First we show a simple implementation of an n-consensus object from a single n-IDconsensus object and n registers. We then refine this implementation to eliminate the use of registers. The resulting implementation uses $2n+1$ n-IDconsensus objects, but is still not a direct implementation. We then describe how to make it direct.

Here is the first implementation: each P_i first writes its proposal v_i in a register R_i and then performs IDconsensus with other processes by proposing i to an n-IDconsensus object $\mathcal{W}$. If P_k is the winner of this IDconsensus, then P_i returns the value in register R_k as the response of the implemented n-consensus object $\mathcal{O}$.

The next implementation, eliminating the use of registers, is in Figure 2. This implementation uses $2n+1$ n-IDconsensus objects. The object named $\mathcal{W}$ serves the same purpose as before: to determine the identity of the process whose proposal is the winning proposal. The objects $O_{i,0}$ and $O_{i,1}$ help P_i communicate its proposal to other processes. Each P_i begins by proposing i to O_{i,v_i} (this corresponds to the step of writing v_i in R_i in the previous implementation). P_i then performs IDconsensus with other processes by proposing i to $\mathcal{W}$. Let *winner* be the value returned by $\mathcal{W}$. If $winner = i$, then P_i is the winner and its proposal v_i is the winning proposal, so P_i returns v_i as the response of the implemented n-consensus object $\mathcal{O}$. Otherwise, P_i must learn P_{winner}'s proposal, which is the winning proposal. For this, P_i proposes i to $O_{winner,0}$. If $O_{winner,0}$ returns *winner*, then the proposal of P_{winner} must be 0, so P_i returns 0; otherwise the proposal of P_{winner} must be 1, so P_i returns 1. The correctness of this implementation is obvious. We thus have:

$\mathcal{W}$, $\{O_{i,0}, O_{i,1} \mid 0 \leq i \leq n-1\}$: n-IDconsensus objects

Procedure Propose(P_i, v_i, $\mathcal{O}$) /* $v_i \in \{0,1\}$ */
  winner : integer local to P_i
begin
1. Propose(P_i, i, O_{i,v_i})
2. *winner* := Propose($P_i, i, \mathcal{W}$)
3. **if** *winner* $= i$ **then**
4. **return** v_i
5. **else if** Propose($P_i, i, O_{winner,0}$) returns *winner*
6. **return** 0
7. **else return** 1

end

Figure 2: Implementing n-consensus from n-IDconsensus

Lemma 3.1 n-IDconsensus $\rightarrow$ n-consensus.

The above implementation is not direct: P_i's operation on $\mathcal{O}$ is linearized at its access to $\mathcal{W}$ and *not* at its first access to a base object. We turn it into a direct implementation simply by requiring P_i to perform lines 1 and 2 in Figure 2 simultaneously, in one atomic action. This results in a direct implementation of n-consensus from n-IDconsensus^2. We thus have:

Lemma 3.2 n-IDconsensus^2 $\stackrel{di}{\rightarrow}$ n-consensus.

3.3 The main lemma

We prove that, for all T, if there is an implementation of 3-consensus from T, then there is a *direct* implementation, of twice the space complexity, of 3-IDconsensus from T^2.

Our design exploits the well-known bivalency argument due to Fischer, Lynch, and Paterson [FLP85]. Since bivalency arguments are standard, our definitions here are informal. Let $\mathcal{O}$, shared by P_0, P_1, and P_2, be a 3-consensus object implemented from objects $O_1, \ldots, O_m$ of type T. Let v_i denote P_i's proposal to $\mathcal{O}$. A *configuration* of $\mathcal{O}$ is a tuple consisting of the states of the three access procedures Propose($P_i, v_i, \mathcal{O}$) ($i \in \{0,1,2\}$) and the states of objects $O_1, \ldots, O_m$. A configuration C is *v-valent* (for $v \in \{0,1\}$) if there is no execution from C in which $\overline{v}$ is decided upon by some P_i. In other words, once in configuration C, no matter how P_0, P_1, and P_2 are scheduled, no P_i returns $\overline{v}$. A configuration is *monovalent* if it is either 0-valent or 1-valent. A configuration is *bivalent* if it is not

monovalent. If E is a finite execution of the implementation starting from configuration C, $E(C)$ denotes the configuration at the end of the execution E.

Lemma 3.3 $T \rightarrow$ 3-consensus *implies* $T^2 \stackrel{di}{\Rightarrow}$ 3-IDconsensus.

Proof Let $\mathcal{I}$ be an implementation of 3-consensus from T. Let $\mathcal{O}$, shared by P_0, P_1, and P_2, be a 3-consensus object implemented using $\mathcal{I}$ from objects $O_1, \ldots, O_m$ of type T. Pick val_0, val_1, and val_2, the proposals of P_0, P_1, and P_2, respectively, so that C_0, the initial configuration of $\mathcal{O}$, is bivalent. (For instance, $val_0 = 0$ and $val_1 = val_2 = 1$ would be adequate.)

Let E be a finite execution from C_0 such that (1) $C_{crit} = E(C_0)$ is bivalent, and (2) For all P_i, if P_i takes a step from C_{crit}, the resulting configuration is monovalent. (If such E does not exist, it is easy to see that there is an infinite execution E' in which no process decides. Thus, some process takes infinitely many steps in E' without deciding, contradicting the wait-freedom property of the implementation of $\mathcal{O}$.) Let S_v be the set of P_i whose step from C_{crit} results in a v-valent configuration. Since C_{crit} is bivalent, neither S_0 nor S_1 is empty. Furthermore, $S_0 \cap S_1 = \emptyset$. Without loss of generality, let $S_0 = \{P_0\}$ and $S_1 = \{P_1, P_2\}$. Thus, if P_0 is the first to take a step from C_{crit}, then regardless of how P_0, P_1, and P_2 are scheduled subsequently, every P_i eventually decides 0. Similarly, if either of P_1 and P_2 is the first to take a step from C_{crit}, then regardless of how P_0, P_1, and P_2 are scheduled subsequently, every P_i eventually decides 1.

In the configuration C_{crit}, let σ_0, σ_1, and σ_2 denote the states of the access procedures Propose$(P_0, val_0, \mathcal{O})$, Propose$(P_1, val_1, \mathcal{O})$, and Propose$(P_2, val_2, \mathcal{O})$, respectively. Also let $\mu_1, \ldots, \mu_m$ denote the states of $O_1, \ldots, O_m$, respectively, in C_{crit}.

Given the above context, we are ready to describe the *direct* implementation of a 3-IDconsensus object $\mathcal{A}$, shared by processes Q_0, Q_1, and Q_2, from objects $O'_1, \ldots, O'_m$, $O''_1, \ldots, O''_m$ of type T. Each Q_i may access up to two base objects atomically in a single step.

The idea is to use the given implementation $\mathcal{I}$ to build two 3-consensus objects (from the available objects $O'_1, \ldots, O'_m, O''_1, \ldots, O''_m$), initialize both of them to C_{crit}, and require Q_0, Q_1, and Q_2 to access them in such a way that, if Q_i is the first to take a step, all of Q_0, Q_1, and Q_2 eventually return i. The details are as follows.

Using implementation $\mathcal{I}$ and the objects $O'_1, \ldots, O'_m$, implement a 3-consensus object $\mathcal{O}'$ that can be shared by P'_0, P'_1, and P'_2. Similarly, using implementation $\mathcal{I}$ and the objects $O''_1, \ldots, O''_m$, implement another 3-consensus object $\mathcal{O}''$ that can be shared by P''_0, P''_1, and P''_2.

Initialize each of $\mathcal{O}'$ and $\mathcal{O}''$ to C_{crit}; more specifically,

1. Since $\mathcal{O}'$ is implemented to be shared by P'_0, P'_1, and P'_2, it supports the access procedures Propose$(P'_0, val_0, \mathcal{O}')$, Propose$(P'_1, val_1, \mathcal{O}')$, and Propose$(P'_2, val_2, \mathcal{O}')$. Initialize the states of these three access procedures to σ_0, σ_1, and σ_2, respectively.

2. Initialize the states of objects $O'_1, \ldots, O'_m$ to $\mu_1, \ldots, \mu_m$, respectively.

3. Since $\mathcal{O}''$ is implemented to be shared by P_0'', P_1'', and P_2'', it supports the access procedures $\texttt{Propose}(P_0'', val_0, \mathcal{O}'')$, $\texttt{Propose}(P_1'', val_1, \mathcal{O}'')$, and $\texttt{Propose}(P_2'', val_2, \mathcal{O}'')$. Initialize the states of these three access procedures to σ_0, σ_1, and σ_2, respectively.

4. Initialize the states of objects $O_1'', \ldots, O_m''$ to $\mu_1, \ldots, \mu_m$, respectively.

Each Q_i executes two access procedures, one of $\mathcal{O}'$ and one of $\mathcal{O}''$. The exact mapping of which two access procedures Q_i executes is as follows: Process Q_0 executes $\texttt{Propose}(P_0', val_0, \mathcal{O}')$ and $\texttt{Propose}(P_1'', val_1, \mathcal{O}'')$; Q_1 executes $\texttt{Propose}(P_1', val_1, \mathcal{O}')$ and $\texttt{Propose}(P_0'', val_0, \mathcal{O}'')$; and Q_2 executes $\texttt{Propose}(P_2', val_2, \mathcal{O}')$ and $\texttt{Propose}(P_2'', val_2, \mathcal{O}'')$. Each process executes its access procedures as follows. In its first step, each process executes the first step of *both* of its access procedures simultaneously (this is possible because in the implementation being designed a process is allowed to access up to two objects in one step). After its first step, each process executes any one of its access procedures to completion and then executes the other access procedure to completion. Once a process executes both its access procedures to completion, it knows the decision values d' and d'' returned by the 3-consensus objects $\mathcal{O}'$ and $\mathcal{O}''$, respectively.

The key observation is the following: If Q_i is the first process to take a step (among Q_0, Q_1, and Q_2), since the first step of Q_i corresponds to the first step of both of its access procedures, the decision values of both $\mathcal{O}'$ and $\mathcal{O}''$ become fixed at the end of Q_i's first step. Furthermore, given our mapping between processes and access procedures, we have the following obvious relationships: $(d', d'') = (0, 1)$ if and only if Q_0 is the first process to take a step, $(d', d'') = (1, 0)$ if and only if Q_1 is the first process to take a step, and $(d', d'') = (1, 1)$ if and only if Q_2 is the first process to take a step. (Notice that $(d', d'') = (0, 0)$ cannot occur.) Thus, from the values d' and d'', each Q_j determines the identity of the Q_i which took the very first step and returns i. This completes the proof of the lemma. □

Lemma 3.4 $T \rightarrow$ 3-consensus *implies* $T^4 \xrightarrow{di}$ 3-consensus.

Proof Suppose that $T \rightarrow$ 3-consensus. By Lemma 3.3, $T^2 \xrightarrow{di}$ 3-IDconsensus. By Theorem 2.1, $T^4 \xrightarrow{di}$ 3-IDconsensus^2. This, together with Lemma 3.2 and the transitivity of $\xrightarrow{di}$ (Proposition 2.1), gives the lemma. □

3.4 Multi-object theorem for types at level 3 or higher

The next lemma states that if type T objects are strong enough to implement 3-consensus objects in the standard single-access model, then they are good for implementing n-consensus objects (for any n) provided that processes can access sufficiently many of them ($2n^2$, to be precise) in a single step.

Lemma 3.5 $Con(T) \geq 3$ *implies* $Con(T^{2n^2}) \geq n$.

Proof $Con(T) \geq 3$

$\Rightarrow$ $T \rightarrow$ 3-consensus

$\Rightarrow$ $T^4 \xrightarrow{di}$ 3-consensus (by Lemma 3.4)

$\Rightarrow$ $\forall m > 0 : Con(T^{4m}) \geq Con(\text{3-consensus}^m)$ (by Theorem 2.2)

$\Rightarrow$ $\forall m > 0 : Con(T^{4m}) \geq \sqrt{2m}$ (by Theorem 2.3)

$\Rightarrow$ $Con(T^{2n^2}) \geq n$ (by letting $m = n^2/2$)

□

Finally, we present the multi-object theorem for types at level 3 or higher. This theorem is immediate from the above lemma.

Theorem 3.1 *$Con(T) \geq 3$ implies $Con(T^*) = \infty$.*

3.5 Sketch of an alternative proof

In this section, we sketch an alternative proof of Theorem 3.1. Afek, Merritt, and Taubenfeld introduced a consensus object that also supports a read operation [AMT96]. Specifically, an object of type (f, r)-consensus can be accessed by f "proposer" processes and r "reader" processes. A proposer may only invoke *propose 0* or *propose 1*, and a reader may only invoke *read*. The sequential specification is as follows: if the first operation is *propose* v, all operations return v; if the first operation is a *read*, operations return arbitrary responses. A result in [AMT96] states that an n-consensus object, for any n, can be implemented using (f, r)-consensus objects if sufficiently many of them can be accessed simultaneously. Specifically:

Theorem 3.2 ([AMT96]) *(f, r)-consensus$^m \rightarrow n$-consensus, where $n \geq \sqrt{mrf + f^2/4} + f/2$.*

We can define the type (f, r)-IDconsensus and obtain a result analogous to Theorem 3.2. Specifically, an object of type (f, r)-IDconsensus can be accessed by at most f proposers, $P_0, P_1, \ldots, P_{f-1}$, and r readers. Proposer P_i may only invoke *propose* i, and a reader may only invoke *read*. The sequential specification is as follows: if the first operation is *propose* i, all operations return i; if the first operation is a *read*, operations return arbitrary responses. With minor modifications, the proof of Theorem 3.2 can be adapted to obtain the following result, an analog of Theorem 3.2 for IDconsensus:[3]

[3]The proof of Theorem 3.2 uses the following idea. To solve consensus, processes split themselves into two groups G_0 and G_1, processes in each G_i solve consensus recursively to obtain the consensus value v_i for the group, then the two groups compete with all processes in G_i proposing v_i, and finally every process adopts the value proposed by the winning group. For this idea to work in the proof of Theorem 3.3, which deals with IDconsensus instead of consensus, it should be possible for the processes in the losing group, say G_1, to determine the winner of the winning group, namely, G_0. If registers are available, processes in each group G_i can write the winner of G_i in some register $R(G_i)$ before competing with the other group $G_{\bar{i}}$. Thus, processes in G_1, the losing group in our running example, can easily determine the winner of the winning group G_0 by reading the register $R(G_0)$. Unfortunately, however, registers are not available — the only available objects are (f, r)-IDconsensus objects. We overcome this difficulty with a trick similar to the one used in Section 3.2, where we first presented a construction that uses registers and then showed how to eliminate registers.

Theorem 3.3 (f,r)-IDconsensus$^m \rightarrow n$-IDconsensus, *where* $n \geq \sqrt{mrf + f^2/4} + f/2$.

If a type implements 3-consensus, using the familiar bivalency arguments it is fairly easy to show that T directly implements $(2,1)$-IDconsensus. Thus:

Theorem 3.4 $T \rightarrow$ 3-consensus *implies* $T \stackrel{di}{\Rightarrow} (2,1)$-IDconsensus.

Now Theorem 3.1 can be proved as follows. Suppose that $T \rightarrow$ 3-consensus. By Theorem 3.4, $T \stackrel{di}{\Rightarrow} (2,1)$-IDconsensus. A result in [AMT96] states that if $X \stackrel{di}{\Rightarrow} Y$, then $X^m \stackrel{di}{\Rightarrow} Y^m$. Using this, we have $T^{n^2/2} \stackrel{di}{\Rightarrow} (2,1)$-IDconsensus$^{n^2/2}$. By Theorem 3.3, $(2,1)$-IDconsensus$^{n^2/2} \rightarrow n$-IDconsensus. Thus, we have $T^{n^2/2} \rightarrow n$-IDconsensus. Since n-IDconsensus $\rightarrow n$-consensus (by Lemma 3.1), we have $T^{n^2/2} \rightarrow n$-consensus. Therefore, $Con(T^*) = \infty$. Hence Theorem 3.1.

4 Multi-object theorems for types at levels 1 and 2

In this section, we relate $Con(T)$ and $Con(T^*)$ when $Con(T)$ is 1 or 2. Specifically, if $Con(T) = 1$, we show that $Con(T^*) \in \{1, 2, \infty\}$, and exhibit types in all three of these categories. If $Con(T) = 2$, we show that $Con(T^*) \in \{2, \infty\}$; it was shown in [AMT96] that there are types in both these categories. The following lemma is useful in establishing some of these results.

Lemma 4.1 $Con(T^*) \geq 3$ *implies* $Con(T^*) = \infty$.

Proof

$Con(T^*) \geq 3$

$\Rightarrow$ $Con(T^m) \geq 3$ for some $m > 0$

$\Rightarrow$ $Con((T^m)^*) = \infty$ (by Theorem 3.1)

$\Rightarrow$ $Con(T^*) = \infty$

□

Next we present the multi-object theorem for types at level 1.

Theorem 4.1

1. $Con(T) = 1$ *implies* $Con(T^*) \in \{1, 2, \infty\}$.
2. *There is a type* T *such that* $Con(T) = 1$ *and* $Con(T^*) = 1$.
3. *There is a type* T *such that* $Con(T) = 1$ *and* $Con(T^*) = 2$.
4. *There is a type* T *such that* $Con(T) = 1$ *and* $Con(T^*) = \infty$ *[Her91].*

Proof Part (1) follows directly from Lemma 4.1. For part (2), consider the type `trivial` which supports only a single operation that always returns the same response. Clearly, $Con(\texttt{trivial}) = 1$ and $Con(\texttt{trivial}^*) = 1$. For part (4), `register` is an example of a type T for which $Con(T) = 1$ [CIL94, DDS87, LA87, Her91] and $Con(T^*) = \infty$ [Her91]. We prove part (3) below.

Consider the `blind-increment` type that supports *read* and *blindInc* operations. The *read* operation returns the value of the object without affecting it. The *blindInc* operation increments the value and returns *ack*.

Claim 4.1 $Con(\texttt{blind-increment}) = 1$.

Proof: This claim is well-known and is immediate from the following three facts:

(i) `blind-increment` has a (trivial) implementation from `atomic-snapshot`,[4]

(ii) `atomic-snapshot` has an implementation from `register` [AAD+93, And93], and

(iii) `register` cannot implement 2-`consensus` [CIL94, DDS87, LA87, Her91]. □

Claim 4.2 $Con(\texttt{blind-increment}^2) \geq 2$.

Proof: We can implement a 2-IDconsensus object, shared by processes P_0 and P_1, from two blind-increment objects O_0 and O_1, both initialized to 0, as follows. Process P_i both reads O_i and blind-increments $O_{\bar{i}}$ in the same step. If P_i reads 0, it is the winner, and so it returns i. Otherwise $P_{\bar{i}}$ is the winner, so it returns $\bar{i}$. It is easy to verify that this protocol is correct. From this and Lemma 3.1, we have the claim. □

Claim 4.3 *For all* $m > 0$, $Con(\texttt{blind-increment}^m) \leq 2$.

Proof: The operations of `blind-increment` commute. Therefore, by a result in [AMT96], $Con(\texttt{blind-increment}^m) \leq 2$. □

By the above three claims, `blind-increment` is an example of a type T for which $Con(T) = 1$ and $Con(T^*) = 2$. This completes the proof of Theorem 4.1. □

Finally, we present the multi-object theorem for types at level 2.

Theorem 4.2

1. *$Con(T) = 2$ implies $Con(T^*) \in \{2, \infty\}$.*
2. *There is a type T such that $Con(T) = 2$ and $Con(T^*) = 2$ [Her91, AMT96].*

[4]Informally, an object of type `atomic-snapshot` stores a vector of n integers, where n is the number of processes that may access the object. Any process may perform a *read* operation, which simply returns the vector. Process P_i may perform a *write* (i, v) which changes the value of the i^{th} element of the vector to v.

3. There is a type T such that $Con(T) = 2$ and $Con(T^*) = \infty$ [Her91, AMT96].

Proof Part (1) is immediate from Lemma 4.1 and the observation $Con(T^*) \geq Con(T)$. Parts (2) and (3) follow from the following known results: $Con(\texttt{fetch\&add}) = Con(\texttt{swap}) = 2$ [Her91], $Con(\texttt{fetch\&add}^*) = 2$ [AMT96] and $Con(\texttt{swap}^*) = \infty$ [AMT96]. □

References

[AAD+93] Y. Afek, H. Attiya, D. Dolev, E. Gafni, M. Merritt, and N. Shavit. Atomic snapshots of shared memory. *Journal of the ACM*, 40(4):873–890, 1993.

[AMT96] Y. Afek, M. Merritt, and G. Taubenfeld. The power of multi-objects. In *Proceedings of the 15th Annual ACM Symposium on Principles of Distributed Computing*, May 1996.

[And93] J. Anderson. Composite registers. *Distributed Computing*, 6(3):141–154, 1993.

[CIL94] B. Chor, A. Israeli, and M. Li. Wait-free consensus using asynchronous hardware. *SIAM Journal on Computing*, 23(4):701–712, August 1994.

[DDS87] D. Dolev, C. Dwork, and L. Stockmeyer. On the minimal synchronism needed for distributed consensus. *Journal of the ACM*, 34(1):77–97, January 1987.

[FLP85] M. Fischer, N. Lynch, and M. Paterson. Impossibility of distributed consensus with one faulty process. *JACM*, 32(2):374–382, 1985.

[Her91] M.P. Herlihy. Wait-free synchronization. *ACM TOPLAS*, 13(1):124–149, 1991.

[LA87] M.C. Loui and H.H. Abu-Amara. Memory requirements for agreement among unreliable asynchronous processes. *Advances in computing research*, 4:163–183, 1987.

[MT94] M. Merritt and G. Taubenfeld. Atomic m-register operations. *Distributed Computing*, 7:213–221, 1994.