Computer Science Department Technical Report
University of California
Los Angeles, CA 90024-1596

THE IMPLICATION OF THE BOROWSKY-GAFNI SIMULATION ON THE SET-CONSENSUS HIERARCHY

E. Borowsky
E. Gafni

July 1993
CSD-930021

The Implication of the Borowsky-Gafni Simulation on the Set-Consensus Hierarchy *

(Extended Abstract)

Elizabeth Borowsky (borowsky@cs.ucla.edu)
Eli Gafni (eli@cs.ucla.edu)

Computer Science Department
University of California, Los Angeles
Los Angeles, CA 90024
U.S.A.

Abstract

We elaborate on applications of the *non-blocking-busy-wait* simulation method, introduced in [6], to extend the linear consensus hierarchy of Herlihy [11] into a partial order encompassing set consensus objects. In particular, we show (n, k)-set consensus can not be implemented using (m, j)-set consensus objects if $n/k > m/j$. We explore cases cases where this hierarchy is strict.

1 Introduction

The problem of distributed consensus is fundamental to many applications of distributed systems. In consensus, processors communicate with each other to agree on a common value. The k-set consensus (agreement) problem introduced by Chaudhuri [7] is a generalization of the standard problem of consensus. In k-set consensus, processors with private initial values each decide on some processor initial value such that the set of decision values is size at most k. When n processors are to execute k-set consensus, we refer to the problem as (n, k)-set consensus. The standard consensus problem is the special case of set consensus with $k = 1$. Throughout this paper we abbreviate $(n, 1)$-set consensus as n-consensus.

In one of the most celebrated results in distributed computing, Fisher, Lynch and Patterson (FLP) [9] established that asynchronous consensus tolerating a single undetected fail-stop processor is impossible. When a processor fail-stops, from that point on the processor takes no steps. In an asynchronous system, a fail-stopped processor is

*Work supported by NSF Presidential Young Investigator Award under grant DCR84-51396 .

indistinguishable from a slow processor. Upon introducing (n, k)-set consensus, Chaudhuri [7] conjectured that (n, k)-set consensus is impossible if k processors may fail-stop undetected. She proved the problem is solvable if at most $k-1$ processors may fail-stop. Recently her conjecture was proven correct [6, 12, 15].

While investigating the problem of consensus, Herlihy [11] showed the n-consensus task is universal in the sense that if n processors can achieve wait-free n-consensus then they can wait-free implement any task. He then proposed to define a hierarchical relationship between wait-free objects. He defines object A to be above B in the hierarchy if A can wait-free implement B, but not vice-versa. He established such hierarchical relationships between n-consensus objects for even n. Jayanti and Toueg [13] extend this relationship to all n. Here we elaborate on what we have written rather tersely in [6], which extends Herlihy's hierarchy into a partial order encompassing set-consensus objects. We prove a wait-free (n, k)-set consensus object can not be implemented using a wait-free (m, j)-set consensus object if $n/k > m/j$. This extension is a easy outcome of the non-blocking busy-wait simulation technique introduced in [6] and yields as a special case an alternate proof of the Jayanti and Toueg result.

The paper is organized as follows. In Section 2 we define our model. In Section 3 we present the non-blocking busy-wait agreement protocol. Then, in Section 4 we use it in the simulation, proving the extended hierarchical relationship. We end with concluding remarks.

2 Model

Our model consists of n processors communicating asynchronously through shared memory. W.l.o.g. (e.g. [14]) the shared memory is constructed from single-writer multi-reader atomic registers. The processors communicate to execute a task, after which each processor decides on an output value.

Formally, a *task* is a point to set mapping of a set of processors with given input to possible combinations of outputs. A processor output is a *decision*, and an implementation of a task is a *decision protocol.* A decision protocol is said to be *wait-free* if, in every execution, each processor is guaranteed to reach a decision after taking a finite number of steps. Such a protocol is a *wait-free implementation* of the corresponding task. An *object* is a black-box implementation of a task. All objects and protocols in this paper are wait-free.

We now define the possible hierarchical relationships between objects. Given two objects, we say object A is *stronger* than object B if there is no wait-free implementation of A using any number of objects of type B and any number of read-write registers. Object A is *strictly stronger* than B if A is stronger than B and there is a wait free implementation of B using objects of A. Object A is *weaker* than object B if it is not stronger than B, and A is *strictly weaker* than B if B is strictly stronger than A. We say objects A and B are *incomparable* if A is stronger than B and vice-versa, in other words, if neither object can implement the other. If objects A and B can wait-free

implement each other, they are said to be *equivalent.*

The task used throughout this paper to show additional hierarchical relationships is the (n, k)-Set Consensus task. In (n, k)-set consensus, each of n processors with private input values must decide on the input value of some processor, such that the set of decision values is of size at most k. Notice that in a wait-free implementation of (n, k)-set consensus, if a processor comes alone and decides before any other processor wakes up, then that processor necessarily decides on its own input value.

3 Non-blocking-Busy-Wait Agreement Protocol

In this section we design a read-write agreement protocol which will be used in the simulation. The protocol may not terminate if even a single processor participating in it fails, thus it is not wait-free. However, it has the property that all waiting is done in the last step. A wait is a loop of reads that terminates when the read returned satisfies a given condition. The code of the agreement protocol consists of a wait-free section ending with a single read loop and a subsequent agreement decision. Processors can decide once all participating processors have reached the wait statement. Thus, if a processor at the wait statement can not make a decision, there must be at least one processor currently in the wait-free section of the code.

The agreement protocol we use is a one-shot mutual-exclusion algorithm. Here we state it in the generality of l-exclusion, with all processors observing when any processor decides to enter the critical section. The algorithm is presented in Figure 1, and is a simple variant of the FIFO l-exclusion algorithm proposed in [2]. All the variables are atomic single writer multi-reader (i.e. none are private). Assume every read step is an atomic snapshot of the memory. Thus, any read or write is done atomically although reading and writing many variables.

For the simulation, assume we have n codes which are to be simulated by some number of processors. Take l to be 1 in the agreement protocol. The simulating processors must come to an agreement on the value of each read step in a simulated code. Thus, in order to agree on a value, each processor first writes its proposed read value into the shared memory then enters the agreement protocol. When the agreement protocol terminates, all participating processors agree on a single participating processor. Since the winner is participating, it must have written a proposed value in the memory. All processors take this value as the agreement value for the simulated read. Each code is simulated sequentially, so no processor can simulate past a read statement before the agreement protocol for that statement terminates. However, a processor waiting on the outcome of one simulated read may join or start simulating a read in another code. In this manner, simulating processors can wait-free simulate an execution of the n processor protocol.

Protocol for Processor i:
Initially:
x_i=false, $S_i = \emptyset$, $label_i = (0, i)$.

$x_i := true$;
read
$label_i := (Max\{y|(y,j) = label_j\} + 1, i)$;
read
$S_i := \{j|x_j = true\}$;
read until there exists j such that:

$$S_j \neq \emptyset \text{ and} |\{k \in S_j | label_k \leq label_j\}| \leq l$$

(* let $\leq$ denote the lexicographic ordering on labels *)
choose: j

Figure 1: Agreement Protocol

4 The Set Consensus Hierarchy

We use the simulation to prove (n,k)-set consensus is stronger than (m,j)-set consensus for $n/k > m/j$. Since the cases where $n < k$ or $m < j$ are trivial (since each processor can simply choose its own value) we limit our discussion to the case where $n > k$ and $m > j$. We show that if (n,k)-set consensus can be implemented using (m,j)-set consensus objects, then a $(jk+1, jk)$-set consensus object can be implemented from read-write registers contradicting the results in [6, 12, 15]. Thus we assume we have n codes invoking the primitive operations of read, write and invoke((m,j)-object, value) followed by return((m,j)-object, value). For each (m,j)-set consensus object used, this pair of lines appears in at most m codes.

Specifically, assume that there is a wait-free (n,k)-set consensus protocol using (m,j)-set consensus objects. Then, by using j copies of this protocol, there is a wait-free (jn, jk)-set consensus protocol using (m,j)-set consensus objects as well. So, let $jk+1$ processors simulate the jn codes of (jn, jk)-set consensus. Before simulating the read, write, invoke, and return steps of a simulated code, the $jk+1$ simulating processors associate the code with an input from the simulating processors. To agree on the input, the simulating processors perform an agreement protocol on their own initial values. The simulating processors may then begin simulating a code by using the agreement protocol for each read and return((m,j)-object, value) statement of the code. A processor failing in the wait-free section of an input, read or return agreement can block at most that simulated code.

The line invoke((m,j)-object, value) is simulated by using the agreement protocol with $l = j$. To block such a multi-valued agreement at least j simulating processors must fail. However, when such an agreement is blocked, it blocks the simulation of up to m simulated codes. Thus, if one of the $jk+1$ simulating processors remains working, there

can be at most $m(jk/j) = mk$ codes blocked. Since $n/k > m/j$ it holds that $nj > mk$. so there must be at least one unblocked simulated code for the remaining processor to execute to completion, thus achieving a jk-set consensus value. Consequently, if an (n, k)-set consensus protocol could be implemented using (m, j)-set consensus objects, then $jk + 1$ processors could reach jk-set consensus wait-free, contradicting [6, 12, 15].

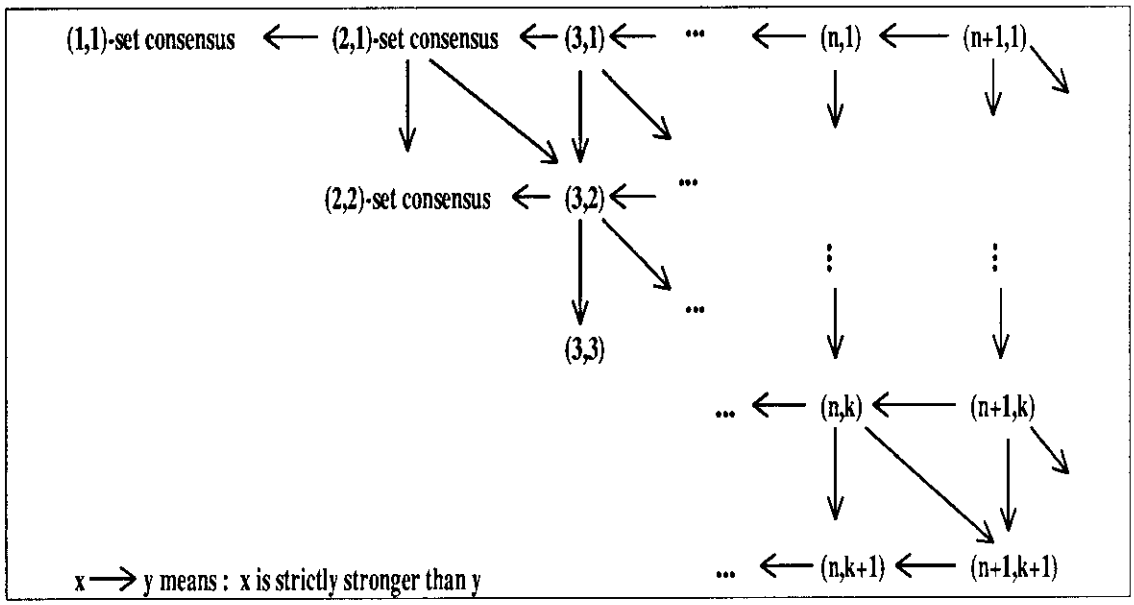

Figure 2: The Set Consensus Hierarchy.

We have shown that (n, k)-set consensus is stronger than (m, j)-set consensus when $n/k > m/j$. Now we investigate the cases when (n, k)-set consensus is strictly stronger than (m, j)-set consensus. Clearly, in the case where $n \geq m$, $k \leq j$ and $n/k > m/j$, (n, k)-set consensus can easily implement (m, j)-set consensus. Thus, in this case (n, k)-set consensus is strictly stronger than (m, j)-set consensus. This yields the implications presented in Figure 2 that (n, k)-set consensus is strictly stronger than $(n, k + 1)$-set consensus and $(n - 1, k)$-set consensus. However, in the case where $n \geq m$, $k > j$ and $n/k > m/j$, we conjecture that (n, k)-set consensus and (m, j)-set consensus are incomparable. In addition, we conjecture that when $n < m$ and $n/k > m/j$, (n, k)-set consensus can be used to implement (m, j)-set consensus. In the case where $\lceil m/n \rceil k \leq j$ this conjecture holds easily as we can partition the m processors into $\lceil m/n \rceil$ groups of size $\leq n$, each of which can access a separate (n, k)-set consensus object, and thus achieve the result. Also, if $j - \lfloor m/n \rfloor k \leq m - \lfloor m/n \rfloor n$ the conjecture holds as we can partition the m processors into $\lfloor m/n \rfloor$ groups of size n which each use an (n, k)-set consensus object, while the remaining processors each choose their own value. This last case proves the implication shown in Figure 2 that (n, k)-set consensus is strictly stronger than $(n + 1, k + 1)$-set consensus. Thus, we have shown strict hierarchical relationships between various set consensus objects, and supplied our conjectures about

the cases yet to be determined. A schematic representation of the strict hierarchy is presented in Figure 2.

5 Conclusion

Expanding on the applications of the non-blocking busy-wait simulation method, first introduced in [6], we have proved a hierarchy of wait-free set-consensus objects. We extend Herlihy's consensus hierarchy to encompass set consensus, showing (n,k)-set consensus is stronger than (m,j)-set consensus if $n/k > m/j$. In particular, we show (n,k)-set consensus is strictly stronger than both $(n-1,k)$-set consensus and $(n,k+1)$-set consensus for $n > k$, and that $(n-1,k)$-set consensus is strictly stronger than $(n,k+1)$-set consensus, thus giving the complete relationships between set consensus objects which differ by one in either parameter. Notice, that between two set consensus objects of the same n/k value, it is clear that the object with smaller n value can implement that with larger n value. That is, an (n,k)-set consensus object can implement a (cn,ck)-set consensus object. We suspect that the converse is false.

References

[1] Y. Afek, H. Attiya, D. Dolev, E. Gafni, M. Merritt and N. Shavit. "Atomic Snapshots of Shared Memory", *Proc. 9th ACM Symp. on Principles of Distributed Computing*, pages 1-14, 1990.

[2] A. Afek, D. Dolev, E. Gafni, M. Merritt and N. Shavit,"First-in-first-Enabled l-Exclusion", *Proc. 4th International Workshop On Distributed Algorithms* , Bari, Italy 1990.

[3] H. Attiya, A. Bar-Noy and D. Dolev, "Sharing Memory Robustly in Message Passing Systems", *Proc. 9th ACM Symp. on Principles of Distributed Computing*, pages 363–375, 1990.

[4] H. Attiya, A. Bar-Noy, D. Dolev, D. Koller, D. Peleg, and R. Reischuk, "Achievable Cases in an Asynchronous Environment", *Proc. 28th IEEE Symp. on Foundations of Computer Science*, pages 337–346, 1987.

[5] O. Biran, S. Moran, and S. Zaks, "A Combinatorial Characterization of the Distributed Tasks Which Are Solvable in the Presence of One Faulty Processor", *Proc. 7th ACM Symp. on Principles of Distributed Computing*, pages 263–275, 1988.

[6] E. Borowsky and E. Gafni, "Generalized FLP Impossibility Result for t-resilient Asynchronous Computations," *Proc. 25th Symp. on Theory of Computing*, 1993.

[7] S. Chaudhuri, "Agreement is Harder Than Consensus: Set Consensus Problems in Totally Asynchronous Systems", *Proc. 9th ACM Symp. on Principles of Distributed Computing*, pages 311–324, 1990.

[8] D. Dolev, C. Dwork and L. Stockmeyer, "On the Minimal Synchronization Needed for Distributed Consensus", JACM 34, January 1987.

[9] M. J. Fischer, N. A. Lynch, and M. S. Paterson, "Impossibility of Distributed Consensus with One Faulty Process", *Journal of the ACM*, Vol. 32, No. 2, pages 374–382, April 1985.

[10] G. N. Frederickson and N. A. Lynch, "Electing a Leader in a Synchronous Ring", *Journal of ACM*, Vol. 34, No. 1, pages 98–115, 1987.

[11] P. M. Herlihy, "Impossibility and Universality Results for Wait-Free Synchronization", *Proc. 7th ACM Symp. on Principles of Distributed Computing*, pages 276–290, 1988.

[12] M. Herlihy and N. Shavit," The Asynchronous Computability Theorem for t-Resilient Tasks", *Proc. 25th Symp. on Theory of Computing*, 1993.

[13] P. Jayanti and S. Toueg, "Some Results on the Impossibility, Universality and Decidability of Consensus", private communication.

[14] L. Lamport, "On Interprocess Communication, Parts I and II", *Distributed Computing*, Vol. 1, pages 77–101, 1986.

[15] M. Saks and F. Zaharoglou," Wait-Free k-set Agreement is impossible: The Topology of Public Knowledge", *Proc. 25th Symp. on Theory of Computing*, 1993.