A Simple Proof of a Simple Consensus Algorithm[*]

Jayadev Misra
Department of Computer Sciences
The University of Texas at Austin
Austin, Texas 78712
(512) 471-9547
misra@cs.utexas.edu

1 Introduction

We present a proof of a simple consensus algorithm. The algorithm is known [1]; the proof style is new. It is difficult to reason about communicating processes when some of them may fail. We show that representing the algorithm at a higher level—using equations only—permits us to treat failures in a formal manner. This approach was first utilized in Chandy and Misra [1988, Chapter 18] for proving the correctness of an unauthenticated Byzantine Agreement protocol due to Dolev et al. [1982], and Srikanth and Toueg [1987], for the case when the number of reliable processes is more than twice the number of unreliable processes; this paper deals with a simpler problem—the number of reliable processes exceeds thrice the number of unreliable processes—which has a simpler algorithm.

2 The Problem

We are given s reliable processes and t unreliable processes, where $s > 3t \geq 0$. To simplify matters, assume that the total number of processes, $s+t$, is odd. Processes communicate by messages. Local to each process is a variable, which initially has a value, 0 or 1. It is required to design a protocol, to be followed by the reliable processes, such that eventually every reliable process has the same value in its local variable. Furthermore, if all reliable processes have the same initial value then their final value is the same as their initial value.

The difficulty in constructing a solution is due to the presence of unreliable processes. These processes may send arbitrary messages, and there is no way, a priori, to distinguish the reliable and the unreliable processes.

3 An Algorithm: Informal Version

Let the processes be numbered 1 through $s+t$. Processes communicate with each other in "rounds." Each round consists of two phases of message transmissions: In round k, $k > 0$, in the first phase every process sends its value to all processes (including itself); in the second phase, process k sends the majority value it received in the first phase (majority is well defined since $s + t$ is odd) to all processes. If a process receives s, or more, instances of the same value in the first phase of the round, it sets its local variable to this value; otherwise, it sets its local variable to the value received (from process k) in the second phase of this round.

[*]Partially supported by ONR Contracts N00014-86-K-0763 and N00014-87-K-0510 and by a fellowship from The John Simon Guggenheim Foundation.

It can be shown that at the end of round $(t+1)$ all reliable processes have the same value. Furthermore, if all reliable processes have the same initial value their final values are same as the (common) initial value.

4 Reasoning About Consensus Algorithms

Consider any consensus algorithm, first ignoring the faults. Any such algorithm consists of two kinds of actions: local computations at processes and communications between pairs of processes. Both kinds of actions can be represented by equations of the form

$$b = f(c)$$

where b, c are local variables that belong to the same or different processes. If b, c are variables of one process, this equation represents a local computation at the process—computing $f(c)$ and assigning it to b. If b, c are variables of different processes then the equation represents a communication—the process to which c is local sends $f(c)$ to the process to which b is local; the latter process assigns the value to b upon receiving the message. The important point to note is that any deterministic algorithm on a network of processes—and all consensus algorithms are deterministic in the absence of faults—can be represented by equations of the above form. (If a variable b is assigned k different values during a computation, we employ k different variables, b^1, b^2, …, b^k, each of which is assigned a single value; this is a standard trick for converting an imperative program to a functional one.)

We now investigate the effect of faults on such an equation. If the process to which b is local is unreliable then the equation may not be satisfied as a result of the computation—the process may fail to assign to b. Similarly if the process to which c is local is unreliable then the equation may not be satisfied, because the process may transmit an arbitrary value for c. Therefore, an equation is to be discarded if any variable in it is local to an unreliable process. Thus, for an algorithm consisting of the equations

$$b = c\ ,\ d = c$$

if the process p to which c is local is unreliable, then both equations will be discarded (it cannot then be asserted that $b = d$). This models possible malicious behavior by p; it may supply different values for c to the two processes of which b and d are local.

Designing a consensus algorithm is difficult because the algorithm designer does not know which processes are unreliable, and hence which equations will be discarded. She can only rely on the fact that there is a limited number of unreliable processes; by designing the algorithm appropriately it is possible to ensure that only a limited number of equations are discarded.

A consensus algorithm is given by (1) naming the (local) variables of each process, (2) designating the variables where the initial and the final values are stored for each process, and (3) writing a set of equations over these variables. Such an algorithm is *correct* provided that for *any* subset of s processes, the subset of equations that name variables of these processes only has a solution and the solution satisfies the requirements given in Section 2, *viz.*, all final values are equal, and if all initial values are equal then that is the final value.

We will work with equations and dispense with the notions of process, message, communication, and rounds. (These terms will be used only in giving an intuitive explanation of what each variable represents.) For the present paper, we merely construct an equational representation of the algorithm given in Section 3 and prove its correctness. In general, however, one may design an algorithm by postulating a sequence of refinement steps that culminates in a set of equations of the desired type; see Chapter 18 of Chandy and Misra [1988] for such a development.

5 Equational Representation of the Algorithm

We use x, y to stand for processes and k for the round number. Define the following variables of process x for round $k > 0$.

$d[x,k]$	$=$	value at x at the end of round k.
$r[x,y,k]$	$=$	the value received by x from process y in the first phase of round k.
$m[x,k]$	$=$	the majority value received by x in the first phase of round k.
$c[x,k]$	$=$	the number of instances of $m[x,k]$ received by x in the first phase of round k.

The initial and final values for process x are stored in $d[x,0]$ and $d[x,t+1]$, respectively. Next, we write the equations that represent the algorithm.

Variables $m[x,k]$, $c[x,k]$ can be defined as follows: $m[x,k]$ is the majority value in $r[x,y,k]$, for all y, and $c[x,k]$ is the number of occurrences of $m[x,k]$ in $r[x,y,k]$, for all y. The other variables for $k > 0$ are related by

(1) $$d[x,k] = \begin{cases} m[x,k] & \text{if} \quad c[x,k] \geq s \\ m[k,k] & \text{if} \quad c[x,k] < s \end{cases}$$

(2) $$r[x,y,k] = d[y,k-1]$$

We treat (1) as two equations, corresponding to the two alternatives.

6 Correctness of the Algorithm

We will prove that all reliable processes have the same final value (Theorem 1) and if initial values at all reliable processes are equal then the common final value is equal to the common initial value (Theorem 2).

Designate an arbitrary subset of s processes as reliable. For the rest of this paper, u, v, w denote arbitrary reliable processes. Since u, v are reliable it follows from (2) that for $k > 0$,

(3) $$r[u,v,k] = d[v,k-1] \ .$$

Lemma 1: For all u, w, $d[u,w] = m[w,w]$.

Proof: From (1), since u, w are both reliable:

$$d[u,w] = \begin{cases} m[u,w] & \text{if } c[u,w] \geq s \\ m[w,w] & \text{if } c[u,w] < s \end{cases}$$

The lemma follows by showing that

$$c[u,w] \geq s \quad \Rightarrow \quad m[u,w] = m[w,w]$$

From the definitions of $c[u,w]$, $m[u,w]$, there are $c[u,w]$ different y's for which $r[u,y,w] = m[u,w]$. Let R be the set of reliable processes among the above y's.

$$\begin{array}{ll} & |R| \\ \geq & \quad \{\text{there are } t \text{ unreliable processes}\} \\ & c[u,w] - t \\ \geq & \quad \{\text{Assuming that } c[u,w] \geq s\} \\ & s - t \\ > & \quad \{s > 3t\} \\ & (s+t)/2 \end{array}$$

That is, R consists of a majority of processes.

From the definition of $m[w,w]$ there is a majority S of processes such that for any y in S

$$r[w,y,w] = m[w,w].$$

Since S and R both consist of a majority of processes, some process v belongs to both sets (process v is reliable since R consists of reliable processes only). Now,

$m[w, w]$
$=$ $\{v$ is in S, $r[w, y, w] = m[w, w]$ for all y in $S\}$
$r[w, v, w]$
$=$ $\{$from (3) substituting w, w for u, k, respectively$\}$
$d[v, w - 1]$
$=$ $\{$from (3) substituting w for $k\}$
$r[u, v, w]$
$=$ $\{v$ is in R, $r[u, y, w] = m[u, w]$ for all y in $R\}$
$m[u, w]$ □

Notation: We write $\langle \forall\ u\ ::\ p_u \rangle$ for the predicate that is traditionally written as $\forall\ u.p_u$.

Lemma 2: For any $k, k \geq 0$, $\langle \forall\ u\ ::\ d[u, k] = g \rangle \Rightarrow \langle \forall\ u\ ::\ d[u, k + 1] = g \rangle$.

Proof:

$\langle \forall\ u\ ::\ d[u, k] = g \rangle$
$\Rightarrow$ $\{$from (3), substituting $u, w, k + 1$ for u, v, k, respectively$\}$
$\langle \forall\ u, w\ ::\ r[u, w, k + 1] = g \rangle$
$\Rightarrow$ $\{w$ ranges over reliable processes; number of reliable processes $= s \geq$ majority of processes$\}$
$\langle \forall\ u\ ::\ m[u, k + 1] = g\ \wedge\ c[u, k + 1] \geq s \rangle$
$\Rightarrow$ $\{$from (1)$\}$
$\langle \forall\ u\ ::\ d[u, k + 1] = g \rangle$ □

Theorem 1: (All final values are equal.) There is a g such that

$$\langle \forall\ u\ ::\ d[u, t + 1] = g \rangle.$$

Proof: Since there are t unreliable processes, there is a reliable process w such that $w \leq (t + 1)$. We show that the desired g is $m[w, w]$.

$\{$From Lemma 1$\}$
$\langle \forall\ u\ ::\ d[u, w] = m[w, w] \rangle$
$\Rightarrow$ $\{$using Lemma 2, repeatedly, and that $w \leq (t + 1)\}$
$\langle \forall\ u\ ::\ d[u, t + 1] = m[w, w] \rangle$ □

Theorem 2: (If all initial values are equal, final values are equal to the common initial value.)

$$\langle \forall\ u\ ::\ d[u, 0] = g \rangle \Rightarrow \langle \forall\ u\ ::\ d[u, t + 1] = g \rangle.$$

Proof: Repeated application of Lemma 2. □

7 Conclusion

The proof as given above replaces operational reasoning—how a process behaves—by more traditional, mathematical arguments.

Acknowledgment: I am indebted to Piotr Berman for showing me this algorithm.

8 References

1. Berman, Piotr, and Juan A. Garay. Asymptotically Optimal Distributed Consensus. *Proc. ICALP 89 (16th International Colloquium on Automata, Languages and Programming)*, Stresa, Italy, July 1989, *Lecture Notes in Computer Science*, **372**,80–94, Springer-Verlag, New York.

2. Chandy, K. Mani and Jayadev Misra. *Parallel Program Design: A Foundation*. Addison-Wesley, Reading, Mass., 1988.

3. D. Dolev et al. An Efficient Byzantine Algorithm without Authentication. TR RJ 3428, IBM, March 1982.

4. Pease, M., R. Shostak, and L. Lamport. Reaching Agreement in the Presence of Faults. *J.ACM*, **27**,2 (April 1980), 228–234.

5. Srikanth, T. K., and S. Toueg, Simulating Authenticated Broadcasts to Derive Simple Fault Tolerant Algorithms. *Distributed Computing*, **2**,2 (August 1987), 80–94.