

On the Nonexistence of Resilient Consensus Protocols

Gadi Taubenfeld*

Computer Science Department, Yale University, New Haven, CT 06520

Key words: resilient consensus protocols, asynchronous message passing systems, fault tolerance, crash failures, impossibility proof.

1 Introduction

In [3], Fischer, Lynch and Paterson proved that in asynchronous message passing systems there cannot exist a consensus protocol that tolerates even a single undetectable crash failure. Their proof of this fundamental result relies on operational details such as sending and receiving messages, etc.

Chandy and Misra [1] have taken an axiomatic non-operational approach to the consensus problem. Their idea is to define asynchronous systems and consensus protocols by a set of axioms, making no mention of operational details. The result in [1] is weaker than that in [3], since it is not assumed in [1] that all messages sent are eventually delivered.

In this note we use the Chandy and Misra approach to prove a result that is similar to the one in [3]. We do so without the need of introducing any operational notions. We believe that our proof is simpler than that in [3]. Most of the ideas that appear in [3] and [1] are used here, as well as new key ideas. The result here is slightly more general than that in [3], since we use a weaker resiliency requirement and assume that any enabled event eventually is either applied or becomes not enabled, in a model where processes may be nondeterministic.

The rest of the paper is organized as follows. In Section 2, we present three axioms capturing the nature of asynchronous message passing systems. In Section 3, we present five axioms defining resilient consensus protocols. In Section 4, we prove that there does not exist an asynchronous resilient consensus protocol by showing that the set of eight axioms is inconsistent.

2 Asynchronous Message Passing Systems

A *protocol* is a nonempty (possibly infinite) set C of runs and a finite set P of processes. A *run* is a finite sequence of events of the form e_p where $p \in P$. For an event e_p , we say that e_p *involves* process p . An event is *in* a run iff it is one of the events in the sequence that comprise the run. No operational meaning is assigned to runs, processes or events.

*Supported in part by the Hebrew Technical Institute scholarship and by NSF grant CCR-8405478.

In the rest of this paper p, p' denote processes and x, y, z denote runs. Also $\langle x; y \rangle$ is the sequence obtained by concatenating x and y . The notation $y \geq x$ means that y is an extension of x . For $y \geq x$, $(y - x)$ denotes the suffix of y obtained by removing x from y , and $|(y - x)|$ denotes the length of $(y - x)$. For any x and p , let x_p be the subsequence of x containing all events in x that involve p . Run y includes x iff x_p is a prefix of y_p for all $p \in P$. Runs x and y are *equivalent w.r.t. p* , denoted by $x[p]y$, iff $x_p = y_p$. For a given protocol, we define $Ext(x, p)$ to be the set of all extensions of x by events involving process p only. More formally, $Ext(x, p) = \{y \mid y \geq x \text{ and } x[p']y \text{ for every } p' \neq p\}$. Finally, an event e_p is *enabled* at run x of a given protocol iff $\langle x; e_p \rangle$ is also a run of that protocol.

Next, asynchronous message passing systems are characterized by three properties that any protocol operating in such an environment should satisfy.

Definition: *An asynchronous protocol is a protocol whose runs satisfy the properties,*

P1: *Every prefix of a run is a run;*

P2: *Let $\langle x; e_p \rangle$ and y be runs. If y includes x and $x[p]y$ then $\langle y; e_p \rangle$ is a run;*

P3: *Only finitely many events are enabled at a run.*

By P1 the empty sequence, denoted by *null*, is a run. P1 is exactly the same as A1 in [1]. P2 means that if an event e_p can happen at some point in a run, then the same event can happen at a later point, provided that p has taken no steps between the two points. P2 is slightly different from A2 in [1], since we have to use the relation “includes” instead of “extends”. P3 means that we are assuming bounded nondeterminism. P3 corresponds to the assumption in [3] that a (deterministic) process may send an arbitrary but finite set of messages to other processes in each step. We do not need to use A3 from [1].

Lemma 1: *Let x, y and z be runs. If y includes x , $x[p]y$ and $z \in Ext(x, p)$ then $w = \langle y; (z - x) \rangle$ is a run.*

Proof: The proof is by induction on the length of $(z - x)$. When $|(z - x)| = 0$, w is a run since $w = y$. We assume that w is a run when $|(z - x)| = k$, and prove that w is also a run when $|(z - x)| = k + 1$. Let $|(z - x)| = k + 1$. Then, for some sequence z' and event e_p , $z = \langle z'; e_p \rangle$. By P1, z' is a run. Since $|(z' - x)| = k$, by the induction hypothesis $w' = \langle y; (z' - x) \rangle$ is a run. Since y includes x , $x[p]y$ and $z' \in Ext(x, p)$ it follows that w' includes z' and $z'[p]w'$. Thus, by P2 $w = \langle w'; e_p \rangle$ is a run. \square

In order to define consensus protocols formally, we need the concept of a fair sequence. A *fair sequence* is a sequence of events, where (1) each finite prefix is a run (of the protocol), and (2) if the sequence is finite then no event is enabled at the sequence, otherwise (when the sequence is infinite) each event that is enabled at all but finitely many prefixes appears infinitely often in the sequence.

In a fair sequence every enabled event eventually is either applied or becomes not enabled. A fair sequence may be infinite; in such a case it is not a run. Unlike in [1], a fair sequence may be finite. We notice that according to the model considered in [3], once a message is sent, the event of receiving that message is enabled until the message is received. Hence, by interpreting some of the events in our model as sending and receiving events, a fair sequence captures the intuition of an execution where all processes get a chance to proceed and all messages sent are eventually delivered.

The proof of the impossibility result in Section 4 still works, even if we interpret some of the events as *broadcast* events in which a process sends, in one atomic step, messages to all other processes. Also, the result holds, with no need to change the proof, if we add the assumption that a message is received only if it was sent previously, or that messages sent from one process to another are received in the order they are sent.

3 Resilient Consensus Protocol

In this section we define the notion of a resilient consensus protocol. The definition is adopted, with minor changes, from [1].

Informally, a (binary) *consensus protocol* is a protocol in which each process has a local write-once output register. For every fair execution (i.e., where all processes can proceed and all messages sent are eventually delivered) there exists a finite prefix in which all the processes choose one out of two possible values, called *white* and *black*. That is, each process writes a *decision value* into its local output register. The decision values written by all processes are the same. The trivial solutions, in which only one of the two values is always chosen, is ruled out by the requirement that processes do not choose the same value in all runs. A consensus protocol is resilient if in spite of a crash failure of any single process the remaining processes can still choose some value. A crash failure of a process means that the process is not involved in any subsequent event.

In order to give a formal definition of a resilient consensus protocol we assume that each run is colored by one of the three colors: white, black and gray. Intuitively, a run is white (black) if in any extension of it, processes may choose only white (black), and it is gray if both white and black are still possible decision values.

Definition: *A resilient consensus protocol is a protocol whose runs satisfy the properties,*

C1: *There is a white run and a black run;*

C2: *Every gray run has an extension that is white and an extension that is black;*

C3: *If y includes x and x is white (black) then y is white (black);*

C4: *Every fair sequence has a finite prefix that is not gray;*

Resiliency: *For every x and p there exists $y \geq x$ such that $x[p]y$ and y is not gray.*

The first four properties define the notion of a consensus protocol, while the fifth is the resiliency property. It follows from C1 and C3 that the *null* run is gray.¹

4 The Impossibility Result

In this section we prove that there does not exist an asynchronous resilient consensus protocol. First, we introduce the function $val : Runs \times Processes \rightarrow \{-1, 0, 1, 2\}$. The val function captures what some process can do by itself at a given run.

¹It is possible to interpret the first event of each process (in a given run) as reading some input value and get a model similar to that in [3]. Using the axiomatic framework we get “for free” that the null computation is gray, and do not have to do a separate proof, as in [3], to obtain an initial bivalent configuration.

$$val(x, p) = \begin{cases} -1 & \text{if } Ext(x, p) \text{ includes a white run and does not include a black run} \\ 0 & \text{if } Ext(x, p) \text{ includes only gray runs} \\ 1 & \text{if } Ext(x, p) \text{ includes a black run and does not include a white run} \\ 2 & \text{if } Ext(x, p) \text{ includes both a white run and a black run} \end{cases}$$

Intuitively, when $val(x, p) = -1$ process p at run x can make each process choose white but it cannot make any process choose black. When $val(x, p) = 1$ process p at run x can make each process choose black but it cannot make any process choose white. The fact that $val(x, p) = 2$ means that at run x process p can make each process choose white as well as black. Finally, $val(x, p) = 0$ means that at run x process p cannot make any process choose a value. All five lemmas given below refer to some fixed asynchronous consensus protocol. That is, the Resiliency property is used only in the proof of the theorem.

Lemma 2: *For any run x and any process p , if x is gray and $|val(x, p)| = 1$ then there exists $y \geq x$ such that $val(x, p) \times val(y, p) = -1$.*

Proof: Since x is gray, by C2, there exist a white run $y \geq x$, and a black run $y' \geq x$. By definition, $val(y, p) = -1$ and $val(y', p) = 1$. If $val(x, p) = 1$ then $val(x, p) \times val(y, p) = -1$, if $val(x, p) = -1$ then $val(x, p) \times val(y', p) = -1$. \square

Lemma 3: *For any two runs x and y , and any process p , if $val(x, p) = 2$, $x \leq y$ and $x[p]y$ then $val(y, p) = 2$.*

Proof: Since $val(x, p) = 2$, there exist two runs z and z' both belonging to $Ext(x, p)$, which are white and black respectively. By Lemma 1, $w = \langle y; (z - x) \rangle$ and $w' = \langle y; (z' - x) \rangle$ are runs belonging to $Ext(y, p)$. Clearly, w includes z and w' includes z' . By C3, w is white and w' is black, and hence $val(y, p) = 2$. \square

Lemma 4: *For any two runs $x \leq y$ and any process p , if $y \in Ext(x, p)$ or $x[p]y$ then $val(x, p) \times val(y, p) \neq -1$.*

Proof: If either $|val(x, p)| \neq 1$ or $|val(y, p)| \neq 1$ then we are done. So, assume that $|val(x, p)| = |val(y, p)| = 1$. There are two possible cases.

Case 1: $y \in Ext(x, p)$. Since $|val(y, p)| = 1$, there exists $z \in Ext(y, p)$ that is not gray. Since $y \in Ext(x, p)$, it follows that $z \in Ext(x, p)$. If $val(y, p) = 1$, then z is black, which implies that $val(x, p) \neq -1$; if, on the other hand, $val(y, p) = -1$, then z is white, which implies that $val(x, p) \neq 1$. Therefore, $val(x, p) \times val(y, p) \neq -1$.

Case 2: $x[p]y$. Since $|val(x, p)| = 1$, there exists $z \in Ext(x, p)$ which is not gray. By Lemma 1, $w = \langle y; (z - x) \rangle$ is a run. Clearly, $w \in Ext(y, p)$ and w includes z . Since w includes z and z is not gray, it follows from C3 that w is not gray and $val(z, p) = val(w, p)$. Since z is not gray, $z \in Ext(x, p)$ and $|val(x, p)| = 1$, it follows that $val(x, p) = val(z, p)$. Similarly, since w is not gray, $w \in Ext(y, p)$ and $|val(y, p)| = 1$, it follows that $val(y, p) = val(w, p)$. Thus, we get that $val(x, p) = val(y, p)$, and therefore $val(x, p) \times val(y, p) \neq -1$. \square

Lemma 5: *For any gray run x and any process p , there exists $z \geq x$ such that $|val(z, p)| \neq 1$.*

Proof: Assume to the contrary that there exists a gray run x and a process p such that for any run $z \geq x$, $|val(z, p)| = 1$. Since x is gray and $|val(x, p)| = 1$, it follows from Lemma 2 that there exists a run $w \geq x$ such that $val(x, p) \times val(w, p) = -1$. By P1, there exist runs y_0, y_1, \dots, y_m such that $y_0 = x$, $y_m = w$, and for every $0 < i \leq m$ it is the case that y_i is a one event extension of y_{i-1} . By the assumption, for every $0 \leq i \leq m$, $|val(y_i, p)| = 1$. Hence, since $val(y_0, p) \times val(y_m, p) = -1$, there must exist some $0 < j \leq m$, such that $val(y_{j-1}, p) \times val(y_j, p) = -1$. However, this contradicts Lemma 4. \square

The next lemma generalizes Theorem 1 in [1], it is similar to Lemma 22.3 in [4], and it is a special case of the Splitter Lemma in [5].

Lemma 6: *There exist a run x and a process p such that $val(x, p) = 2$.*

Proof: Assume to the contrary that for any run x and any process p , $val(x, p) \neq 2$. From this assumption and Lemma 5 it follows that for any gray run x and any process p , there exists $z \geq x$ such that $val(z, p) = 0$. Using this fact, we derive a contradiction similar to that in [3] by constructing inductively a fair sequence, called \mathcal{F} , in which every finite prefix is gray. The existence of such a fair sequence contradicts C4.

We use the the following notion. The *priority number* of e_p at z is the greatest non-negative integer k for which there exists $x \leq z$ such that $|z - x| = k - 1$, e_p does not appear in $(z - x)$, and for any run $x \leq y \leq z$ the event e_p is enabled at y . If e_p is not enabled at z then its priority number at z is 0. Intuitively, the priority number of an event is the amount of time the event has been enabled without occurring.

The construction of \mathcal{F} is done in steps. \mathcal{F}_0 is the *null* run which, as follows from C1 and C3, is gray. At each step $m \geq 1$ we extend the gray run \mathcal{F}_{m-1} constructed at step $m - 1$ to a gray run \mathcal{F}_m as follows. If no event is enabled at \mathcal{F}_{m-1} then $\mathcal{F}_m = \mathcal{F}_{m-1} = \mathcal{F}$. Otherwise, we choose arbitrarily one of the events with the greatest priority number at \mathcal{F}_{m-1} , say e_p , and as observed at the beginning of the proof, using Lemma 5, we extend \mathcal{F}_{m-1} to a run \mathcal{G}_{m-1} where $val(\mathcal{G}_{m-1}, p) = 0$. If e_p is not enabled at \mathcal{G}_{m-1} then $\mathcal{F}_m = \mathcal{G}_{m-1}$, otherwise $\mathcal{F}_m = \langle \mathcal{G}_{m-1}; e_p \rangle$. Since by P3 only finitely many events are enabled at a run, every enabled event eventually is either applied or becomes not enabled. Thus, \mathcal{F} is a fair sequence in which every prefix is gray. This contradicts property C4. \square

Theorem: *There is no asynchronous resilient consensus protocol.*

Proof: By Lemma 6, there exist a run x and a process p such that $val(x, p) = 2$. From the Resiliency property, it follows that there exists $y \geq x$ such that y is not gray and $x[p]y$. Hence, $|val(y, p)| = 1$. However, by Lemma 3, $val(y, p) = 2$, a contradiction. \square

It is possible to modify the construction of the fair run in the proof of Lemma 6, so that the impossibility result holds even if we replace P3 with the assumption that the number of enabled events at a given run is at most countable, and assume that the total number of processes is at most countable. Also, the impossibility result holds even when we redefine the notion of enabled event and say that e_p is enabled at x if $\langle y; e_p \rangle$ is a run, for some $y \in Ext(x, p)$. We can also use a weaker resiliency requirement, namely that for every x and p there exists $y \geq x$ such that $x[p]y$ and $|val(y, p)| = 1$. By giving new interpretations to white and black runs other impossibility results can be proved. Finally, it is an intriguing question whether the impossibility result holds under a stronger notion of fairness where a

sequence is fair only if every event that is enabled infinitely often appears infinitely often in the sequence.

Acknowledgements: This work has been inspired and guided by the research reported in [1, 2, 3]. I would like to thank Joe Halpern for drawing my attention to the Chandy and Misra paper and for many invaluable discussions. I am thankful to Jayadev Misra for his comments and notational ideas, and to Mike Fischer, Nissim Francez, Shmuel Katz, Shlomo, Moran and Yaron Wolfsthal for many helpful discussions. Special thanks to the editor Fred Schneider and the two referees for their very constructive comments.

References

- [1] M. Chandy and J. Misra. On the nonexistence of robust commit protocols. Manuscript, November 1985.
- [2] M. Chandy and J. Misra. How processes learn. *Journal of Distributed Computing*, 1:40–52, 1986.
- [3] M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374–382, April 1985.
- [4] N.A. Lynch and K.J. Goldman. Distributed algorithms. Technical Report MIT/LCS/RSS 5, MIT, May 1989. Lecture Notes for 6.852 - Fall 1988.
- [5] G. Taubenfeld, S. Katz, and S. Moran. Impossibility results in the presence of multiple faulty processes. *Proc. of the 9th FCT-TCS conference, Bangalore, India, 1989*. In: *LNCS 405* (eds.:C.E. Veni Madhavan), Springer Verlag 1989, pages 109-120.