

On the Definition of Sequential Consistency

Ali Sezgin^{*} , Ganesh Gopalakrishnan

Abstract

The definition of sequential consistency is compared with an intuitive notion of correctness. That the definition is not strong enough is illustrated through a hypothetical memory system which is clearly incorrect, yet sequentially consistent. It is claimed that the reason for this is the absence of a relation between what actually happens (temporal order) and what seems to happen (logical order). A stronger definition for sequential consistency is proposed.

Key words:

Concurrency, Shared memory, Sequential consistency, Formalization

PACS:

1 Introduction

The behavior of a shared memory system is essentially a relation between a program and the set of executions it might generate. These relations are known as shared memory models. The larger the number of possible executions per program a shared memory model allows, the more optimizations in its implementations the shared memory model is likely to permit. As such, a

^{*} Corresponding author

Email address: asezgin@atilim.edu.tr (Ali Sezgin).

formal definition of a shared memory model is essential as a means to remove any ambiguity from its description.

Many shared memory models have been proposed in the past. One of the first to be formulated, and to have enjoyed much interest from programmers, designers and researchers in formal verification, is sequential consistency [1]. Sequential consistency requires that the memory behaves as if it responds to a single logical processor which issues a single logical instruction stream. This instruction stream is an interleaving of the instruction streams of each processor that also preserves the uniprocessor (sequential) program order. What happens in real time and what seems to have happened logically are usually represented as two strings over responses (completed instructions). The former is the *temporal execution* (order); the latter is *logical explanation* (order).

Despite the apparent consensus on the semantics of the informal definition of [1], we believe that the notion of correctness, the main motivation for the introduction of sequential consistency, is not adequately captured by its definition. We argue that the complete detachment between a temporal execution and its logical explanation poses more problems than previously anticipated.

2 The Framework

A shared memory system is parameterized over the set of processors (or users) the system interacts with, the set of addresses the system possesses, and the set of data values each address can contain; the sets P , A and D , respectively.

A shared memory system might receive an *instruction* from a processor, might perform internal operations or might generate a *response* for a processor. For

simplicity, we will assume that there are two types of instructions (responses): *read* instructions (responses) and *write* instructions (responses). A read instruction issued by processor p to address a is represented by $?a_p$. A write instruction issued by processor p to address a of data value d is represented by $!a_p^d$. A read response is represented by $a_p^d?$ where p and a are as above, and d is the data value returned for the read. A write response is represented as $a_p^d!$ where p , a and d are as above.

3 The Deceiver

Let us define the following machine, called the *deceiver*. It has four main parts: the *synchronizer*, the *filter*, the *decoupled unit*, and the *serial unit*. The synchronizer checks the first $|P| \cdot |A|$ instructions. If this many instructions are issued and there is exactly one write instruction for each address by each processor, the synchronizer gets into *villain* mode. Otherwise, it goes into *angelic* mode. The filter selects which response is to be output once the synchronizer sets a mode of operation. If it is in villain mode, then the responses generated by the decoupled unit are output. Otherwise, the responses generated by the serial unit are output. It also has a storage of size $2|P| \cdot |A|$ to buffer the responses generated by the decoupled and the serial units until the mode of operation is selected by the synchronizer. The decoupled unit consists of $|P|$ copies of a storage unit of $|A|$ addresses with each address capable of holding any data value from D . Each storage unit is uniquely identified with an element from P . When a read instruction $?a_p$ is input by the system, the contents of address a in the storage p is looked up and the response $a_p^d?$ is generated where d is the value returned by the storage unit. When a write

instruction $!a_p^d$ is input by the system, the contents of the address a in storage p is changed to hold the new value d and the response $a_p^d!$ is generated. Finally, the serial unit consists of a single storage of size $|A|$, where again each address can retain any value from D . This storage unit behaves like the ones in the decoupled unit with the exception that the processor index is ignored; all read and write operations are performed on this single storage unit.

4 Deceiver is Sequentially Consistent!

First of all, it is easy to see that the deceiver is not a *correct* memory system. When it goes into the villain mode, it behaves as a non-coherent memory system where no processor is able to see the updates done by any other processor in the system. Each processor reads only what it itself writes and nothing else. Therefore, in villain mode there is no communication among processors through shared memory which means that parallel algorithms developed for sequentially consistent systems will fail: a mutual exclusion algorithm will possibly reach a deadlock; semaphores in general will not work. However, as we shall argue below, the deceiver is sequentially consistent since behaviorally there is nothing wrong with it!

Now, if the machine is in the angelic mode, what is observed is basically a serial machine's behavior. Each instruction is processed in the order it is input and the logical explanation (*i.e.*, how the system seems to have behaved) is the same as the temporal execution (*i.e.*, how the responses are temporally ordered).

If the machine goes into the villain mode after reading the first $|P| \cdot |A|$ in-

structions, each instruction again is processed in order. However, this time, the logical explanation is much different from the temporal execution. The logical explanation, in fact, is the juxtaposition of all the responses of each processor where no interleaving among responses of different processors is allowed. For instance, one logical explanation will first have all the responses for processor 1, then all the responses for processor 2, then for processor 3, and following this pattern up to processor $|P|$.

We will demonstrate the operation of a sample deceiver. We have $P = \{1, 2, 3\}$, $A = \{a\}$ and $D = \{0, 1, 2\}$.

Let us first consider the instruction sequence $!a_1^2 \ ?a_3 \ !a_2^1 \ ?a_1$. Then, since the synchronizer will go into angelic mode after reading the second instruction, which is a read (in other words, $|P| \cdot |A|$ writes have not been performed), the output of the deceiver will be $a_1^2! \ a_3^2? \ a_2^1! \ a_1^1?$. Hence, as can be seen, it is indeed behaving like a serial memory. It is a well-known fact that the serial memory is a sequentially consistent system; in fact, it was proposed as an operational model for sequential consistency.

Let us now consider the instruction sequence

$$!a_1^1 \ !a_3^2 \ !a_2^0 \ ?a_1 \ ?a_2 \ ?a_3 \ !a_1^2 \ !a_2^1 \ ?a_1 \ ?a_2 \ ?a_3$$

This time, the machine will not output any response until the third instruction after which the synchronizer will go into the villain mode (since $|P| \cdot |A|$ writes have been performed). So the execution will be

$$a_1^1! \ a_3^2! \ a_2^0! \ a_1^1? \ a_2^0? \ a_3^2? \ a_1^2! \ a_2^1! \ a_1^2? \ a_2^1? \ a_3^2?$$

One possible logical explanation for this execution can be obtained by rear-

ranging the responses as follows:

$$a_1^1! \ a_1^1? \ a_1^2! \ a_1^2? \ a_2^0! \ a_2^0? \ a_2^1! \ a_2^1? \ a_3^2! \ a_3^2? \ a_3^2?$$

As was mentioned above, the first four responses are all the responses for processor 1, the next four responses are all the responses for processor 2, and the remaining responses all belong to processor 3.

It is not difficult to prove the following.

Claim 1 *All the executions that the deceiver generates belong to a sequentially consistent memory system.*

Hence, deceiver is sequentially consistent due to the absence of a program whose execution violates sequential consistency.

It is also worth noting that using only 2 copies of storage units, each of size $|A|$, we could give a smaller version of the deceiver where only one processor, say processor 1, disconnects itself from the rest of the system when the first $|A|$ instructions all belong to processor 1, are all write's, each one to a different address.

5 Proposal for a New Definition

First of all, we should admit that the deceiver is not likely to be developed by any designer who wants to keep his/her job. It is a highly unreasonable machine which we just used in order to demonstrate the mathematical inaccuracy in the definition of sequential consistency. However, as systems get more and more complex where no one designer will be capable of “knowing it all”,

being aware of every minute detail about the internal operation of the system, modes of operation which behave like the decoupled unit can be possible. We cannot always depend on the sanity of the designer(s), especially when “we” represents a team of formal verification researchers. As such, the designer is always a collaborative competing agent; he/she should help in formalizing the design but his/her assertions about anything should be subject to scrutiny. In short, everything needs to be proved. If we go back to the definition of sequential consistency, we can see that what it intends to carve out as sequentially consistent and what it mathematically defines are two different sets, as our “deceiver” machine exhibits. We believe that the major reason for this inequality is the absence of a constraining relation between the temporal order of execution and its logical explanation. How the responses are generated temporally and what the memory system seems to be doing logically should not be completely arbitrary.

Based on this observation, we believe that one has to be more specific about how much the temporal order and the logical explanation can be *apart*. Let us call a memory system *logically bounded* by b_l if for any of its temporal executions, it has a logical explanation such that if a response occurs at the i^{th} position in the execution, then that response must occur in a prefix of the logical explanation of size at most $i + b_l$. For instance, for serial memory we have $b_l = 0$. This is due to the fact that the temporal order of execution and its logical explanation are the same for serial memories; that is, for the logical explanation equal to the temporal execution, the i^{th} response in the temporal order is also the i^{th} response in the logical order.

For another example, we have the following result for linearizability[2]:

Lemma 2 *For a linearizable system, $b_l = p - 1$ where p is the number of the processors.*

To further illustrate this approach, note that for the deceiver, b_l does not exist. As an example, consider the following sample execution of a system with one address and two processors.

$$a_1^1! \ a_2^2! \ a_1^1? \ a_2^2? \ a_1^1? \ a_2^2? \ \dots$$

In this execution, the deceiver is in the villain mode and hence the decoupled unit is active. The logical explanation has to order first all the responses of one of the processors. Without loss of generality, assume that it is the first processor. Then, the second response, $a_2^2!$ will not occur in the prefix of the logical order of size s_1 where s_1 is the number of responses belonging to processor 1. Since we can arbitrarily increase the number of instructions issued by processor 1, for every n , there will be an execution whose logical explanation will not have the second response in a prefix of size n .

However, neither the serial memory nor the notion of linearizability can be taken as alternative definitions for sequential consistency. Serial memory is too restrictive as it does not allow nondeterminism among different processors. Neither of the two allows a processor to have more than one pending instruction. These are properties which are almost always violated by current systems as satisfying them will be detrimental to the performance of the shared memory system. We give the following stronger definition.

Definition 3 *A shared memory system is sound sequentially consistent (ssc) if it is sequentially consistent and is logically bounded by some b_l .*

One possible point of contention when such a definition is to be used in practice

might be the determination of the bound b_l . In order to relieve the verifier of an extra assertion, one could make use of a set of machines known to be ssc. One such class has already been defined in [3].¹

Lemma 4 *For an $SC_{\mathcal{P},C}(j,k)$ machine, $b_l = k \cdot (|C| - 1)$ where $|C|$ is the cardinality of the color set, C .*

Lemma 5 *A shared memory system is ssc if its language is contained in the language of some SC machine.*

Note that, both the definition of ssc and the use of SC machines as in Lemma 5, will not leave out any sequentially consistent executions. That is, if an execution is sequentially consistent, then it is ssc (as every single execution is logically bounded) and there exists an SC machine generating it (as shown in [3]).

References

- [1] L. Lamport, How to make a multiprocessor computer that correctly executes multiprocess programs, *ACM Transactions on computer* 28 (9) (1979) 690–691.
- [2] M. P. Herlihy, J. M. Wing, Linearizability: a correctness condition for concurrent objects, *ACM Transactions on Programming Languages and Systems* 12 (3) (1990) 463–492.

¹ Unfortunately, due to space limits, we cannot summarize that work here. Briefly, SC machines are a family of shared memory systems parameterized over, besides $\mathcal{P} = \langle P, A, D \rangle$, a finite set C (the color set) and two finite constants j and k . Intuitively, j and k model the nondeterminism in instruction processing and response generation.

- [3] A. Sezgin, Formalization and verification of shared memory, Ph.D. thesis, School of Computing, University of Utah (2004).