Brief Announcement: How Complex Can Sequential Consistency Be?

Dimitar Dimitrov
INSAIT, Sofia University "St. Kliment Ohridski", Bulgaria

Abstract

Sequential consistency (SC) is the prototypical correctness criterion for shared-memory systems. Deciding whether a finite execution history is SC is NP-complete [11]. In this brief announcement, we study the complexity of SC over infinite execution histories. It is known that SC is not among the safety properties [12], which sit at the lowest level of the Borel hierarchy. We show that, under finite arrival, SC is in fact a liveness property. We further demonstrate that no deterministic transducer can compute sequential witnesses for all SC histories, nor can any infinite-state deterministic Büchi acceptor recognize SC. Finally, we establish that, under infinite arrival, SC for memories with unbounded registers and DCAS (double compare-and-swap) is $\mathbf{\Sigma}^1_1$-complete.

2012 ACM Subject Classification Theory of computation → Distributed computing models; Theory of computation → Complexity theory and logic

Keywords and phrases Distributed systems, Consistency, Complexity

Digital Object Identifier 10.4230/LIPIcs.DISC.2026.60

Funding This research was partially funded by the Ministry of Education and Science of Bulgaria (support for INSAIT, part of the Bulgarian National Roadmap for Research Infrastructure).

AI Disclosure We used Google Gemini Pro 3.0 to search for related work and check our results. Gemini conjectured that SC is $\mathbf{\Sigma}^1_1$-complete, but it could not prove it, so we proved it ourselves.

1 Introduction

To reduce synchronization overhead, shared-memory systems typically do not guarantee that memory operations behave sequentially; for instance, a read operation need not return the value most recently written to the register. More generally, the history of memory operations in an execution is only guaranteed to satisfy some weaker correctness condition than sequentiality. The best-known such condition is sequential consistency (SC) [17, 10]:

▶ **Definition 1.** *A possibly infinite history* $H = (op_0, op_1, \dots)$ *is* sequentially consistent *if it can be permuted into a sequential history, the* sequential witness $(op_{\rho(0)}, op_{\rho(1)}, \dots)$*, in a way that preserves the relative order of operations at each process.*

Thus, under sequential consistency, operations can be committed to memory in an order different from their logical order of execution. But as long as the processes synchronize only through the shared memory, they cannot distinguish SC executions from sequential ones [9]. By relaxing some ordering guarantees, a sequentially consistent memory can achieve greater concurrency and better response times [7] than memories with stricter guarantees, such as linearizability [14] – for example, by caching data locally at each process [1].

However, the extra permissiveness of SC can materialize as additional *complexity* when it comes to protocol design. For finite-state protocols, checking that all finite executions are SC is not decidable, whereas checking linearizability is [5]. Moreover, linearizability is normally a safety property [13] (e.g., when the memory operations are deterministic), and this allows a protocol designer to reason about finite executions only. SC, on the other hand, is not a safety property [12], which necessitates reasoning about infinite behavior.

© Dimitar Dimitrov;
licensed under Creative Commons License CC-BY 4.0

This motivates studying the complexity of SC over infinite executions. In this brief announcement, we establish a few basic impossibility results for shared memories with read-write registers. The results hold regardless of whether the memory is finite or infinite. Then we turn to the question of how complex SC can be, given enough resources: memory with unbounded registers and infinitely many arriving processes [2]. We show that if the memory supports DCAS (double compare-and-swap), then SC is $\mathbf{\Sigma}^1_1$-complete.

2 Safety vs. Liveness

Safety and liveness are two essentially different types of conditions that can be imposed on system behavior [16, 4]. The distinction between the two lies in the point in time at which a (necessarily infinite) execution history H can violate a property P:

▶ **Definition 2.** *A* safety *property P has only finite-time violations: if $H \notin P$, then H has a finite prefix that extends to no history in P; a* liveness *property P has only infinite-time violations: if $H \notin P$, then every finite prefix of H extends to a history in P.*

Every property P can be represented non-uniquely as the intersection $S \cap L$ of a safety property S and a liveness property L [4]. To establish the safety part S, one uses inductive invariants [19, 16]. However, for the liveness part L, one typically also needs a well-foundedness argument [20, 18], which may involve arbitrarily large recursive ordinals [6]. This heavier machinery suggests that liveness can be far more complex than safety.

With this in mind, SC itself has sometimes been referred to as a safety property [1]. And while we believe that this is justifiable in spirit, technically

▶ **Proposition 3** (Glusman and Katz [12])**.** *SC is not a safety property.*

Proof. Consider the following history with two processes p and q, and one binary register x:

$$p{:}\mathsf{W}(x;1) \quad q{:}\mathsf{R}(x;0) \quad q{:}\mathsf{R}(x;0) \quad q{:}\mathsf{R}(x;0) \quad q{:}\mathsf{R}(x;0) \quad q{:}\mathsf{R}(x;0) \quad \dots \tag{1}$$

Process q repeatedly reads the initial value 0 and never sees process p writing 1. The history is not SC because there is nowhere to insert the write and obtain a sequential history. But if SC were a safety property, the history would have to be SC, since every finite prefix can be extended to an infinite SC history (e.g., by appending infinitely many $p{:}\mathsf{W}(x;1)$). ◀

If SC is not a safety property, then what is its liveness part? Surprisingly,

▶ **Proposition 4.** *Under finite arrival [2], SC is a liveness property.*[1]

Proof. To prove that SC is a liveness property, it suffices to show that any finite history

$$p_1{:}F_1(x_1;v_1) \quad p_2{:}F_2(x_2;v_2) \quad \dots \quad p_n{:}F_n(x_n;v_n) \tag{2}$$

can be extended to an infinite SC history. Choose a process $q \notin \{p_1, \dots, p_n\}$ that has not arrived yet. Let $r_1 < \dots < r_m$ be the positions of all read operations in the finite history. For every such read $p_{r_i}{:}\mathsf{R}(x_{r_i};v_{r_i})$, we append a corresponding write by the new process; after all of these writes, we append infinitely many copies of an arbitrary write $q{:}\mathsf{W}(x;v)$:

$$\dots \quad q{:}\mathsf{W}(x_{r_1};v_{r_1}) \quad \dots \quad q{:}\mathsf{W}(x_{r_m};v_{r_m}) \quad q{:}\mathsf{W}(x;v) \quad q{:}\mathsf{W}(x;v) \quad q{:}\mathsf{W}(x;v) \quad \dots. \tag{3}$$

Evidently, this history is SC because we can obtain a sequential witness simply by moving each write $q{:}\mathsf{W}(x_{r_i};v_{r_i})$ immediately before the corresponding read $p_{r_i}{:}\mathsf{R}(x_{r_i};v_{r_i})$. ◀

[1] A reviewer kindly noted that the proof adapts to objects whose states are all reachable from one another.

Therefore, an SC memory can exhibit histories in which SC is violated at arbitrarily long finite prefixes. In fact, SC can be violated at all finite prefixes of a single history.

▶ **Example 5.** We can demonstrate this with two processes p and q, and a binary register x. We let p read from x the values $1, 0, 1, 0, \ldots$, and we let q write the same sequence. Then, we interleave these operations, making sure that p is always at least one read ahead of q:

$$p{:}\mathsf{R}(x;1) \quad p{:}\mathsf{R}(x;0) \quad q{:}\mathsf{W}(x;1) \quad p{:}\mathsf{R}(x;1) \quad q{:}\mathsf{W}(x;0) \quad p{:}\mathsf{R}(x;0) \quad q{:}\mathsf{W}(x;1) \quad \ldots \tag{4}$$

Every prefix contains a read not matched by a write, yet in the limit every read is matched by a write; therefore, no prefix is SC, but the whole history is. In fact, the delay between a read and its matching future write can be made arbitrarily long.

Obviously, to realize such histories in full generality, a shared-memory protocol would need to be prescient, which is impossible. That is why Bingham et al. [8] proposed a stronger condition that does not permit "prophetic reads". Nonetheless, it is an interesting question whether prophetic reads can arise in a broader context, for example, when a program is designed to exhibit only SC behaviors under a weak memory such as causal memory [3].

3 Determinism Is Not Enough

Even if no memory protocol can exhibit all SC behaviors, perhaps a sequential witness can nonetheless be computed deterministically for any SC history. However,

▶ **Proposition 6.** *No deterministic transducer can find a sequential witness for all of SC, even if it has an infinite working tape, operates under no time constraints on when to produce output, and is required to produce correct output only when the input is SC.*

Proof. The key fact about transducers is that the output tape is append-only, implying that any prefix of the output depends only on a finite prefix of the input. In other words, the transducer implements a function that *monotonically* maps input prefixes to output prefixes. We shall prove that no such function can find a sequential witness for all of SC.

Suppose we are given such a monotone function f from input prefixes to output prefixes. Consider the following history, which involves a three-valued register x and three processes, where p_1 writes 1 once, p_2 writes 2 once, and q reads 2 infinitely many times:

$$p_1{:}\mathsf{W}(x;1) \quad p_2{:}\mathsf{W}(x;2) \quad q{:}\mathsf{R}(x;2) \quad q{:}\mathsf{R}(x;2) \quad \ldots \tag{5}$$

This history is sequential, and any sequential witness must order $p_1{:}\mathsf{W}(x;1)$ before $p_2{:}\mathsf{W}(x;2)$. Therefore, after f reads a sufficiently long input prefix P that contains both writes, f must output the writes in this order. Now let us feed f the history

$$\boxed{P} \quad q{:}\mathsf{R}(x;1) \quad q{:}\mathsf{R}(x;1) \quad q{:}\mathsf{R}(x;1) \quad q{:}\mathsf{R}(x;1) \quad \ldots \tag{6}$$

This history is SC but not sequential: $p_2{:}\mathsf{W}(x;2)$ must appear before $p_1{:}\mathsf{W}(x;1)$ in the witness. However, by monotonicity, f has already committed to the opposite ordering. ◀

Finding sequential witnesses for all of SC has proven too much to ask, but perhaps recognizing SC deterministically is possible. Let us consider recognition by ω-Turing acceptors. These are similar to Turing's original machines, but they operate on an infinite input tape, with the input head moving only in one direction. The key point is how such a machine decides to accept or reject its infinite input. We consider machines with Büchi acceptance: the input is accepted if and only if the machine reads the entire input and at the same time visits a designated set of internal states infinitely often.

▶ **Proposition 7.** *No deterministic ω-Turing acceptor with Büchi acceptance recognizes SC.*

Proof. Let M be such an acceptor and suppose that it accepts all SC histories. We shall construct a non-SC history H that M must also accept. We again need only two processes p and q, and a binary register x. Process p repeats $W = p{:}\mathsf{W}(x;1)$, while q repeats $R = q{:}\mathsf{R}(x;0)$. We construct H as the limit of a sequence $H_0 \prec H_1 \prec H_2 \prec \ldots$ of finite prefixes.

For this, we shall use the following construction. If we are given an infinite history G accepted by M and a finite prefix $P \prec G$, then after reading P, the machine M must eventually visit some internal state in the designated set, after which it must eventually read the next input symbol. Let $E(P,G)$ be the prefix read by then (note that $P \prec E(P,G)$).

We build the prefixes recursively: $H_0 = \varepsilon$, and $H_{n+1} = E(H_n, H_n \cdot W^\omega) \cdot R$, where W^ω is an infinite padding by W operations. This is well-defined because the histories $H_n \cdot W^\omega$ are SC, and hence accepted by M. The limit $H = \lim_{n\to\infty} H_n$ is accepted, since M necessarily reads every prefix H_n, $n > 0$, and – by determinism – it visits a designated state each time. But no sequential history can contain all the Rs and Ws in H, and hence $H \notin \mathrm{SC}$. ◀

4 Infinite Resources Result in Σ_1^1-Completeness

We have seen that SC resists computation by some very powerful machines. We shall now see that if the shared memory supports unbounded registers, then SC can indeed be extraordinarily complex in the sense of descriptive set theory [15].

In descriptive set theory, complexity is measured with respect to a topology. The set of histories carries a topology in which the closed sets are the safety properties, and the open sets are their complements. These two classes of sets, denoted $\mathbf{\Pi}_1^0$ and $\mathbf{\Sigma}_1^0$, are considered the simplest, and they are given rank 1. More complex sets are defined from simpler ones by means of countable unions and intersections, where $\alpha > 1$ is a countable ordinal:

$$A \in \mathbf{\Pi}_\alpha^0 \iff A = \bigcap B_n, \text{ where each } B_n \in \mathbf{\Sigma}_{\beta_n}^0 \text{ with } \beta_n < \alpha$$
$$A \in \mathbf{\Sigma}_\alpha^0 \iff A = \bigcup B_n, \text{ where each } B_n \in \mathbf{\Pi}_{\beta_n}^0 \text{ with } \beta_n < \alpha.$$

These classes of sets form the *Borel* hierarchy, and a set is *Borel* if it appears in one of them. The complexity of a Borel set is measured by its *rank*, which is the least level α where the set appears in the hierarchy.

To reach sets that are more complex than the Borel sets, we also need a new kind of operation: projection. A set A is $\mathbf{\Sigma}_1^1$ if it can be expressed as[2]

$$A = \{y : \exists x : (x,y) \in B\} \qquad \text{where } B \text{ is Borel.} \tag{7}$$

The most complex sets in $\mathbf{\Sigma}_1^1$ are those A that are $\mathbf{\Sigma}_1^1$-complete under continuous reductions: for every $A' \in \mathbf{\Sigma}_1^1$, there is a continuous map f such that $A' = f^{-1}(A)$.

▶ **Proposition 8.** *Under infinite arrival [2], SC is $\mathbf{\Sigma}_1^1$-complete, provided that the shared memory has unbounded registers that support a DCAS operation that returns no value.*

Proof. Consider the set SCW of pairs (ρ, H), where H is a history and $\rho\colon \mathbb{N} \to \mathbb{N}$ is such that the condition in Definition 1 holds. This set is Borel under the standard assumption that the sequential histories form a safety property. Hence, $\mathrm{SC} = \{H : \exists \rho : (\rho, H) \in \mathrm{SCW}\} \in \mathbf{\Sigma}_1^1$.

[2] We tacitly assume that the Borel hierarchy has also been constructed for sets of pairs.

To show $\mathbf{\Sigma}^1_1$-completeness, we will reduce to SC the archetypal $\mathbf{\Sigma}^1_1$-complete set, namely the set IF of *ill-founded* trees, i.e., trees containing an infinite branch [15, 27.1]. A tree is a prefix-closed set $T \subseteq \mathbb{N}^{<\mathbb{N}}$ of finite sequences of natural numbers. Each $\tau \in T$ is identified with a node, and removing its last element yields the parent node. A branch is a possibly infinite sequence $(\varepsilon = \tau_0, \tau_1, \tau_2, \dots)$ of nodes such that each τ_n is the parent of τ_{n+1}.

The reduction. For a tree T, we need to produce a history H such that $H \in$ SC iff $T \in$ IF. Let $\mathsf{pid}\colon \mathbb{N}^{<\mathbb{N}} \to \mathbb{N}$ be any bijection from potential tree nodes to processes. We will construct H using two registers, x and y, and DCAS operations, such that any infinite branch $(\tau_0, \tau_1, \tau_2, \dots)$ of T corresponds to a sequential witness W where

- x is initially $\bot$, and then it successively holds $\mathsf{pid}(\tau_0), \mathsf{pid}(\tau_1), \mathsf{pid}(\tau_2), \dots$;
- y is initially $\bot$, and then, in step with x, it holds the depths $0, 1, 2, \dots$.

To achieve this, the processes $\mathsf{pid}(\tau)$ with $\tau \in T$ perform the following operations, where $p{:}\mathsf{DCAS}(x, y; a, b, a', b')$ sets (x, y) to (a', b') provided $(x, y) = (a, b)$, returning nothing:

- $p = \mathsf{pid}(\varepsilon)$ performs $F_p = p{:}\mathsf{DCAS}(x, y; \bot, \bot, p, 0)$;
- $p = \mathsf{pid}(\tau \cdot k)$ with parent $q = \mathsf{pid}(\tau)$ performs $F_p = p{:}\mathsf{DCAS}(x, y; q, |\tau|, p, |\tau| + 1)$.

For any $p = \mathsf{pid}(\tau)$ with $\tau \notin T$, the operation F_p is a no-op. To force the witness to describe an infinite branch, we add a fresh process $r \notin \mathbb{N}$ that successively reads $0, 1, 2, \dots$ from y: these are the operations $G_n = r{:}\mathsf{R}(y; n)$. Then we simply let $H = (F_0, G_0, F_1, G_1, F_2, G_2, \dots)$.

Correctness. Let $(\tau_0, \tau_1, \tau_2, \dots)$ be an infinite branch of T, and enumerate the remaining elements of $\mathbb{N}^{<\mathbb{N}}$ as $\sigma_0, \sigma_1, \sigma_2, \dots$. We divide these into buckets $S_n = \{\sigma_i : \max(i, |\sigma_i| + 1) = n\}$. Note that each bucket S_n is finite, so let E_n list the operations $F_{\mathsf{pid}(\sigma)}$, $\sigma \in S_n$, in an arbitrary order. We construct the sequential witness W from the following segments $W_0, W_1, W_2, \dots$:

$$\underbrace{F_{\mathsf{pid}(\tau_0)} \quad G_0 \quad E_0}_{W_0} \quad \underbrace{F_{\mathsf{pid}(\tau_1)} \quad G_1 \quad E_1}_{W_1} \quad \underbrace{F_{\mathsf{pid}(\tau_2)} \quad G_2 \quad E_2}_{W_2} \quad \dots.$$

A routine induction shows that, after each segment W_n, the state is $x = \mathsf{pid}(\tau_n)$ and $y = n$.

The other direction holds almost by construction, and so we omit its proof.

Continuity. We also need to argue that the reduction is continuous, i.e., that finite parts of H depend only on finite parts of T. Indeed, let P be a finite prefix of H of length at most $2(n+1)$ for some n. Any tree T' that agrees with T on the membership of $\mathsf{pid}^{-1}(0), \dots, \mathsf{pid}^{-1}(n)$ necessarily reduces to a history H' that also has P as a prefix. ◀

Note that the reduction is specific to DCAS. It is an interesting question whether one can construct other reductions that need weaker primitives – for example, CAS or even only read and write operations.

References

1 Yehuda Afek, Geoffrey Brown, and Michael Merritt. Lazy caching. *ACM Trans. Program. Lang. Syst.*, 15(1):182–205, January 1993. doi:10.1145/151646.151651.

2 Marcos K. Aguilera. A pleasant stroll through the land of infinitely many creatures. *ACM SIGACT News*, 35(2):36–59, June 2004. doi:10.1145/992287.992298.

3 Mustaque Ahamad, Gil Neiger, James E. Burns, Prince Kohli, and Phillip W. Hutto. Causal memory: Definitions, implementation, and programming. *Distributed Computing*, 9(1):37–49, March 1995. doi:10.1007/BF01784241.

4 Bowen Alpern and Fred B. Schneider. Defining liveness. *Information Processing Letters*, 21(4):181–185, October 1985. doi:10.1016/0020-0190(85)90056-0.

5 Rajeev Alur, Ken McMillan, and Doron Peled. Model-checking of correctness conditions for concurrent objects. *Information and Computation*, 160(1):167–188, July 2000. doi:10.1006/inco.1999.2847.

6 K. R. Apt and G. D. Plotkin. Countable nondeterminism and random assignment. *Journal of the ACM (JACM)*, 33(4):724–767, August 1986. doi:10.1145/6490.6494.

7 Hagit Attiya and Jennifer L. Welch. Sequential consistency versus linearizability. *ACM Trans. Comput. Syst.*, 12(2):91–122, May 1994. doi:10.1145/176575.176576.

8 Jesse D. Bingham, Anne Condon, and Alan J. Hu. Toward a decidable notion of sequential consistency. In *Proceedings of the Fifteenth Annual ACM Symposium on Parallel Algorithms and Architectures*, SPAA '03, pages 304–313, New York, NY, USA, June 2003. Association for Computing Machinery. doi:10.1145/777412.777467.

9 Ivana Filipović, Peter O'Hearn, Noam Rinetzky, and Hongseok Yang. Abstraction for concurrent objects. *Theoretical Computer Science*, 411(51):4379–4398, December 2010. doi:10.1016/j.tcs.2010.09.021.

10 Rob Gerth. Sequential consistency and the lazy caching algorithm. *Distributed computing*, 12(2):57–59, May 1999. doi:10.1007/s004460050057.

11 Phillip B. Gibbons and Ephraim Korach. Testing shared memories. *SIAM Journal on Computing*, 26(4):1208–1244, August 1997. doi:10.1137/S0097539794279614.

12 Marcelo Glusman and Shmuel Katz. Extending memory consistency of finite prefixes to infinite computations. In Kim G. Larsen and Mogens Nielsen, editors, *CONCUR 2001 — Concurrency Theory*, pages 411–425, Berlin, Heidelberg, 2001. Springer. doi:10.1007/3-540-44685-0_28.

13 Rachid Guerraoui and Eric Ruppert. Linearizability is not always a safety property. In Guevara Noubir and Michel Raynal, editors, *Networked Systems*, pages 57–69, Cham, 2014. Springer International Publishing. doi:10.1007/978-3-319-09581-3_5.

14 Maurice P. Herlihy and Jeannette M. Wing. Linearizability: A correctness condition for concurrent objects. *ACM Trans. Program. Lang. Syst.*, 12(3):463–492, July 1990. doi:10.1145/78969.78972.

15 Alexander S. Kechris. *Classical Descriptive Set Theory*, volume 156 of *Graduate Texts in Mathematics*. Springer, New York, NY, 1995. doi:10.1007/978-1-4612-4190-4.

16 L. Lamport. Proving the correctness of multiprocess programs. *IEEE Transactions on Software Engineering*, SE-3(2):125–143, March 1977. doi:10.1109/TSE.1977.229904.

17 Leslie Lamport. How to make a multiprocessor computer that correctly executes multiprocess programs. *IEEE Transactions on Computers*, C-28(9):690–691, September 1979. doi:10.1109/TC.1979.1675439.

18 Zohar Manna and Amir Pnueli. Adequate proof principles for invariance and liveness properties of concurrent programs. *Science of Computer Programming*, 4(3):257–289, December 1984. doi:10.1016/0167-6423(84)90003-0.

19 Susan Owicki and David Gries. An axiomatic proof technique for parallel programs I. *Acta Informatica*, 6(4):319–340, December 1976. doi:10.1007/BF00268134.

20 Susan Owicki and Leslie Lamport. Proving liveness properties of concurrent programs. *ACM Transactions on Programming Languages and Systems*, 4(3):455–495, July 1982. doi:10.1145/357172.357178.