An Impossibility Result on Strong Linearizability in Message-Passing Systems

DAVID YU CHENG CHAN, University of Calgary, Canada
VASSOS HADZILACOS, University of Toronto, Canada
SAM TOUEG, University of Toronto, Canada

We prove that there is no 1-resilient strongly linearizable implementation of a weak object in asynchronous message-passing systems. This object, that we call Test-or-Set (ToS), allows a single distinguished process to apply the set operation once, and a different distinguished process to apply the test operation also once. Since this weak object can be directly implemented by a single-writer single-reader (SWSR) register (and other common objects such as max-register, snapshot, counter, test-and-set, queue, stack, etc...), this result implies that there is no 1-resilient strongly linearizable implementation of a SWSR register (and of these other objects) in message-passing systems.

1 INTRODUCTION

In seminal work, Golab, Higham, and Woelfel showed that linearizability has the following limitation: a randomized algorithm that works with atomic objects against a strong adversary may lose some of its properties if we replace the atomic objects that it uses with objects that are only linearizable [3]. To address this, they proposed a stronger version of linearizability, called strong linearizability: Intuitively, while in linearizability the order of all operations can be determined "off-line" given the entire execution, in strong linearizability the order of all operations has to be fixed irrevocably "on-line" without knowing the rest of the execution. Golab et al. proved that strongly linearizable (implementations of) objects are "as good" as atomic objects for randomized algorithms against a strong adversary: they can replace atomic objects while preserving the algorithm’s correctness properties.

There are many cases, however, where strong linearizability is impossible to achieve. In particular, for shared-memory systems, Helmi et al. proved that a large class of so-called non-trivial objects including multi-writer registers, max-registers, snapshots, and counters, do not have strongly linearizable non-blocking implementations from single-writer multi-reader (SWMR) registers [4].

In this paper, we consider asynchronous message-passing systems, and we show that in such systems there is no 1-resilient strongly linearizable implementation of a weak object that we call Test-or-Set (ToS): with ToS, one distinguished process can apply the set operation (only once), and another distinguished process can apply the test operation (also only once): the test operation returns 1 if set has previously been applied, and it returns 0 otherwise.

Since a single-writer single-reader (SWSR) register directly implements a ToS object, the above result immediately implies that there is no 1-resilient strongly linearizable implementation of a SWSR register in message-passing systems. This result strengthens a recent result by Attiya, Constantin, and Welch which shows that Multi-Writer registers do not have strongly linearizable implementations in message-passing systems [1], and it also answers in the affirmative an open question asked in that paper. Attiya et al. also prove that max-registers, snapshot, and counters objects do not have strongly linearizable implementations in message-passing systems. Since max-registers, snapshot, and counters also directly implement a ToS object, this is also implied by our result.

2 MODEL SKETCH

We consider a standard asynchronous message-passing distributed system where processes communicate via messages and may fail by crashing. The proof of our result is based on a bivalency argument, and we assume that the reader
is familiar with the model and terminology introduced by Fischer, Lynch, and Paterson in [2] to prove their famous impossibility result. Recall that in this model, processes take steps, and in each step a process does the following: it attempts to receive a message \( m \) previously sent to it (\( m = \bot \) if does not receive any message), it changes state according to the message received, and it sends a message to some other process. A step taken by a process \( p \) in which it receives message \( m \) is denoted \( e = (p, m) \) (we also call this an event).

### 2.1 Test-or-Set

We define a simple object called Test-or-Set (ToS) as follows. The state of a ToS object is a single bit, initially 0. A single distinguished process is allowed to apply a single operation test that returns the value of \( b \); it can apply this operation only once. A different distinguished process is allowed to apply a single operation set that sets \( b \) to 1 (and returns done); it can apply this operation only once. The sequential specification of the ToS object is the obvious one: the test operation returns 1 if a set operation has previously been applied, and 0 if not.

### 2.2 Strongly Linearizable Implementations of ToS

In an object implementation, each operation spans an interval that starts with an invocation and terminates with a response. For any two operations \( o \) and \( o' \), \( o \) precedes \( o' \) if the response of \( o \) occurs before the invocation of \( o' \), and \( o \) is concurrent with \( o' \) if neither precedes the other.

Roughly speaking, an object implementation is linearizable [5] if operations behave as if they occur in a sequential order (called "linearization order") that is consistent with the order in which operations actually occur: if an operation \( o \) precedes an operation \( o' \), then \( o \) is before \( o' \) in the linearization order (the precise definition is given in [5]).

Let \( \mathcal{H} \) be the set of histories of a ToS implementation (note that this set is prefix-closed). An operation \( o \) is complete in a history \( H \in \mathcal{H} \) if \( H \) contains both the invocation and response of \( o \), otherwise \( o \) is pending. A completion of a history \( H \) is a history \( H' \) obtained from \( H \) by removing a subset of the pending operations and completing the remaining ones with responses.

**Definition 1.** A function \( f \) is a linearization function for \( \mathcal{H} \) (with respect to the type ToS) if it maps each history \( H \in \mathcal{H} \) to a sequential history \( f(H) \) such that:

1. \( f(H) \) has exactly the same operations as some completion \( H' \) of \( H \).
2. If operation \( o \) precedes \( o' \) in \( H' \), then \( o \) occurs before \( o' \) in \( f(H) \).
3. For any test operation \( t \) in \( f(H) \), if no set operation occurs before \( t \) in \( f(H) \), then \( t \) reads 0; otherwise, \( t \) reads 1.

**Definition 2.** [3] A function \( f \) is a strong linearization function for \( \mathcal{H} \) if:

(L) \( f \) is a linearization function for \( \mathcal{H} \), and

(P) for any histories \( G, H \in \mathcal{H} \), if \( G \) is a prefix of \( H \), then \( f(G) \) is a prefix of \( f(H) \).

**Definition 3.** An algorithm \( A \) that implements a ToS object is strongly linearizable if there is a strong linearization function (with respect to the type ToS) for the set of histories \( \mathcal{H} \) of \( A \).

### 3 IMPOSSIBILITY RESULT

**Theorem 4.** For all \( n \geq 2 \), there is no 1-resilient strong-linearizable implementation of the Test-or-Set object from a message-passing system of \( n \) processes.
An Impossibility Result on Strong Linearizability in Message-Passing Systems

**Proof.** Suppose, for contradiction, that there is a 1-resilient strong linearizable implementation of the ToS object from a message-passing system of \( n \) processes.

We say a configuration \( C \) is \( v \)-valent for \( v \in \{0, 1\} \) if there is no history applicable to \( C \) such that in \( H(C) \), the \textsc{test} operation returns \( 1 - v \).

We assume the processes may take steps outside the intervals of their own operations, and that the adversary determines when the operations are invoked.

Note that the ToS object only allows one process to invoke a \textsc{test} operation and only allows one process to invoke a \textsc{set} operation. The other \( n - 2 \) processes may take steps to help complete these two operations, but do not invoke any operations themselves.

**Claim 4.1.** The initial configuration \( C_{\text{init}} \) (before any operation is invoked) is bivalent.

**Proof.** If the \textsc{test} operation is invoked and completed before the \textsc{set} operation is invoked, the \textsc{test} operation returns 0. If the \textsc{set} operation is invoked and completed before the \textsc{test} operation is invoked, the \textsc{test} operation returns 1 if it completes. Thus the initial configuration is bivalent. \( \square \)

**Claim 4.2.** Let \( H \) be any history applicable to \( C_{\text{init}} \) such that an operation has completed in \( H(C_{\text{init}}) \). Then \( H(C_{\text{init}}) \) is univalent.

**Proof.** Suppose, for contradiction, that there is a history \( H \) applicable to \( C_{\text{init}} \) such that an operation completes in \( H(C_{\text{init}}) \), yet \( H(C_{\text{init}}) \) is bivalent. Since \( H(C_{\text{init}}) \) is bivalent, the \textsc{test} operation has not yet completed in \( H(C_{\text{init}}) \). So the \textsc{set} operation completes in \( H(C_{\text{init}}) \).

Since \( H(C_{\text{init}}) \) is bivalent, for each \( v \in \{0, 1\} \) there is a history \( H_v \) applicable to \( H(C_{\text{init}}) \) such that in \( HH_v(C_{\text{init}}) \), the \textsc{test} operation returns \( v \). For each \( v \in \{0, 1\} \), let \( H'_v = HH_v \).

For any strong linearization function \( f \):

- In \( f(H'_v) \), the \textsc{test} operation occurs before the \textsc{set} operation because otherwise the \textsc{test} operation could not return 0.
- In \( f(H'_v) \), the \textsc{test} operation occurs after the \textsc{set} operation because otherwise the \textsc{test} operation could not return 1.
- \( f(H) \) includes the \textsc{set} operation since the \textsc{set} operation completes in \( H(C_{\text{init}}) \).

Thus \( f(H) \) cannot be a prefix of both \( f(H'_0) \) and \( f(H'_1) \). So the implementation cannot be strongly linearizable. \( \square \)

**Claim 4.3.** Let \( C \) be any bivalent configuration reachable from \( C_{\text{init}} \), \( e = (p, m) \) be any event that is applicable to \( C \), \( C_{-e} \) be the set of configurations reachable from \( C \) without applying \( e \), and \( D = e(C_{-e}) = \{ e(E) | E \in C_{-e} \text{ and } e \text{ is applicable to } E \} \). Then \( D \) contains a bivalent configuration.

**Proof.** Note that since \( e \) is applicable to \( C \), \( e \) is applicable to every configuration in \( C_{-e} \) because messages may be delayed arbitrarily.

Suppose, for contradiction, that \( D \) contains only univalent configurations. Since \( C \) is bivalent, for each \( v \in \{0, 1\} \) there is a configuration \( E_v \) reachable from \( C \) such that \( E_v \) is \( v \)-valent. For each \( v \in \{0, 1\} \), if \( E_v \) is in \( C_{-e} \), then let \( E'_v = e(E_v) \in D \); otherwise \( e \) was applied to reach \( E_v \), so let \( E'_v \) be a configuration in \( D \) from which \( E_v \) is reachable. Thus for each \( v \in \{0, 1\} \), \( E'_v \) is in \( D \) and is \( v \)-valent. So \( D \) contains both 0-valent and 1-valent configurations.
Consequently, there exists a configuration \( C_{\text{deli}} \) in \( C_{-e} \) and a step \( e' \neq e \) applicable to \( C_{\text{deli}} \) such that \( e(C_{\text{deli}}) \) and \( e'e(C_{\text{deli}}) \) are univalent configurations in \( D \) that have opposite valence. There are two cases: either \( e \) and \( e' \) are steps of different processes, or \( e \) and \( e' \) are steps of the same process \( p \).

**Case 1:** \( e \) and \( e' \) are steps of different processes.

Then \( ee'(C_{\text{deli}}) = e'e(C_{\text{deli}}) \) — contradicting the fact that \( e(C_{\text{deli}}) \) and \( e'e(C_{\text{deli}}) \) have opposite valence.

**Case 2:** \( e \) and \( e' \) are steps of the same process \( p \).

Then since \( p \) is only one process, and the implementation is 1-resilient, there is a finite \( p \)-free history \( H_{-p} \) applicable to \( C_{\text{deli}} \) such that an operation has completed in \( H_{-p}(C_{\text{deli}}) \). Since \( H_{-p} \) is \( p \)-free, \( eH_{-p}(C_{\text{deli}}) = H_{-p}e(C_{\text{deli}}) \) and \( e'eH_{-p}(C_{\text{deli}}) = H_{-p}e'e(C_{\text{deli}}) \) are univalent configurations with opposite valence. So \( H_{-p}(C_{\text{deli}}) \) is bivalent — contradicting Claim 4.2 since an operation has completed in \( H_{-p}(C_{\text{deli}}) \).

\( \square \)

We now construct an infinite history \( H_{bi} \) applicable to \( C_{\text{init}} \) such that every configuration in \( H_{bi}(C_{\text{init}}) \) is bivalent as follows:

1. Let \( C = C_{\text{init}} \) (\( C \) is bivalent by Claim 4.1) and let \( S \) be an arbitrary sequence of all \( n \) processes.
2. Let \( p \) be the first process in \( S \), \( m \) be the earliest message in the message buffer of \( p \) in \( C \) (or \( \perp \) if no such message exists), and \( e = (p, m) \).
3. By Claim 4.3, there is a bivalent configuration \( C' \) reachable from \( C \) where \( e \) has been applied.
4. Move \( p \) to the end of \( S \), and let \( C = C' \).
5. Repeat from Step 2.

By Claim 4.2, since \( H_{bi}(C_{\text{init}}) \) never reaches a univalent configuration, no operation ever completes despite every process taking infinitely many steps in \( H_{bi}(C_{\text{init}}) \) — a contradiction.

\( \square \)

4 CONCLUSION

We have proven that there is no 1-resilient strongly linearizable implementation of the ToS object in asynchronous message-passing systems. This result is significant because ToS objects are weak and are directly implementable by many objects, including SWSR registers, max-registers, queues, stacks, snapshots, counters, and test-and-set, even if accessible by only two processes. This suggests that asynchronous message-passing systems cannot implement any "non-trivial" strongly-linearizable deterministic object.

REFERENCES

[1] Attiya, H., Enea, C., Welch, J.L.: Impossibility of strongly-linearizable message-passing objects via simulation by single-writer registers. CoRR abs/2105.06614 (2021). URL https://arxiv.org/abs/2105.06614
[2] Fischer, M.J., Lynch, N.A., Paterson, M.S.: Impossibility of distributed consensus with one faulty process. J. ACM 32(2), 374–382 (1985). DOI 10.1145/3149.214121. URL http://doi.acm.org/10.1145/3149.214121
[3] Golab, W., Higham, L., Woelfel, P.: Linearizable implementations do not suffice for randomized distributed computation. In: Proceedings of the Forty-Third Annual ACM Symposium on Theory of Computing, STOC ’11, p. 373–382 (2011)
[4] Helmi, M., Higham, L., Woelfel, P.: Strongly linearizable implementations: Possibilities and impossibilities. In: Proceedings of the 31st annual ACM Symposium on Principles of distributed computing, PODC ’12, p. 385–394 (2012)
[5] Herlihy, M.P., Wing, J.M.: Linearizability: A correctness condition for concurrent objects. ACM Trans. Program. Lang. Syst. 12(3), 463–492 (1990). DOI 10.1145/78969.78972. URL http://doi.acm.org/10.1145/78969.78972