Linearizability Is Not Always a Safety Property

Rachid Guerraoui[1] and Eric Ruppert[2]

[1] EPFL, Switzerland
[2] York University, Canada

Abstract. We show that, in contrast to the general belief in the distributed computing community, linearizability, the celebrated consistency property, is not always a safety property. More specifically, we give an object for which it is possible to have an infinite history that is not linearizable, even though every finite prefix of the history is linearizable. The object we consider as a counterexample has infinite nondeterminism. We show, however, that if we restrict attention to objects with finite nondeterminism, we can use König's lemma to prove that linearizability is indeed a safety property. In the same vein, we show that the backward simulation technique, which is a classical technique to prove linearizability, is not sound for arbitrary types, but is sound for types with finite nondeterminism.

1 Introduction

One of the most challenging problems in concurrent and distributed systems is to build software objects that appear to processes using them as "perfect": always available and consistent. In particular, proving that implementations of such objects are correct can be very difficult.

To make the challenge more tractable, it is common to divide the difficulty into proving two properties: a *safety* and a *liveness* property [2, 3]. In short, a safety property says "bad things should never happen" whereas a liveness property says "good things should eventually happen." Traditionally, in the context of building "perfect" shared objects, safety has been associated with the concept of *linearizability* [9] while liveness has been associated with a progress guarantee such as *wait-freedom* [8].

– **Linearizability**: Despite concurrent accesses to an object, the operations issued on that object should appear as if they are executed sequentially. In other words, each operation *op* on an object X should appear to take effect at some indivisible instant between the invocation and the response of *op*. This property, also called *atomicity*, transforms the difficult problem of reasoning about a concurrent system into the simpler problem of reasoning about one where the processes access each object one after another.
– **Wait-freedom**: No process p ever prevents any other process q from making progress when q executes an operation on any shared object X. This means that, provided it remains alive and kicking, q completes its operation on X

regardless of the speed or even the failure of any other process p. Process p could be very fast and might be permanently accessing shared object X, or could have failed or been swapped out by the operating system while accessing X. None of these situations should prevent q from executing its operation.

Ensuring each of linearizability or wait-freedom alone is simple. The challenge is to ensure both. In particular, one could easily ensure linearizability using locks and mutual exclusion. But this would not guarantee wait-freedom: a process that holds a lock and fails can prevent others from progressing. One could also forget about linearizability and ensure wait-freedom by creating copies of the object that never synchronize: this would lead to different objects, one per process, defeating the sense of a shared object. So indeed the challenge is to design shared abstractions that ensure both linearizability and wait-freedom. But proving correctness can be made simpler if we could prove each property separately.

It was shown that properties of distributed systems can be divided into *safety* and *liveness* properties [2], each requiring specific kinds of proof techniques. So the general approach in proving the correctness of shared objects is that linearizability, being a safety property, requires techniques to reason about finite executions (such as backward simulation [13]), whereas wait-freedom, being a liveness property, requires another set of techniques to reason about infinite executions. The association between safety and linearizability on the one hand, and liveness and wait-freedom on the other, is considered a pillar in the theory of distributed computing.

This paper shows that, strictly speaking, this association is wrong for the most general definition of object type specifications. More specifically, we show that, in contrast to what is largely believed in the distributed computing literature, linearizability is not a safety property. This might be surprising because (a) it was often argued that linearizability is a safety property, *e.g.*, in [12], and (b) linearizability proofs have used techniques specific to safety properties, *e.g.*, backward simulation [13]. In fact, there is no real contradiction with our new result for the following reasons.

- To prove that linearizability is not a safety property, we exhibit an object, which we call the *countdown* object, and a non-linearizable history such that every finite prefix of the execution is linearizable. The object we consider has infinite nondeterminism, which might occur, for instance, in a distributed system that seeks to ensure fairness (as we discuss in Sect. 2.2). Interestingly, the execution we use in our proof is by a single process, so it demonstrates that other consistency conditions that are weaker than linearizability (such as sequential consistency) are also not safety properties for the countdown object.
- We show, however, that if we restrict attention to objects with finite nondeterminism, we can use König's lemma [10] to prove that linearizability is indeed a safety property. We thus highlight that, even if this was not

always stated in the past, claims that linearizability is a safety property, should assume finite nondeterminism.[1] Lynch's proof that linearizability is a safety property [12] applies only to the more restricted class of deterministic objects.

In the same vein, we show that the backward simulation technique, which is sometimes used to prove linearizability, is not sound for arbitrary types (if infinite nondeterminism is permitted). It is sound, however, for finite nondeterminism.

The rest of the paper is organized as follows. We describe our system model in Sect. 2. We recall the notion of linearizability in Sect. 3. In Sect. 4 we recall the concept of safety and give our counterexample that shows linearizability is not a safety property. Then we show in Sect. 5 that, if we restrict ourselves to objects with finite nondeterminism, linearizability becomes a safety property. We consider the implications for backward simulations in Sect. 6 and conclude the paper in Sect. 7.

2 System Model

We consider a system consisting of a finite set of n *processes*, denoted $p_1, \ldots, p_n$. Processes communicate by executing operations on *shared objects*. The execution of an operation op on an object X by a process p_i is modelled by two events, the invocation event denoted $inv[X.op \text{ by } p_i]$ that occurs when p_i invokes the operation, and the response event denoted $resp[X.res \text{ by } p_i]$ that occurs when the operation terminates and returns the response res. (When there is no ambiguity, we talk about *operations* where we should be talking about *operation executions*.)

2.1 Objects

An object has a unique identity and a type. Multiple objects can be of the same type. A type is defined by a sequential specification that consists of

- the set Q of possible states for an object of that type,
- the initial state $q_0 \in Q$,
- a set OPS of operations that can be applied to the object,
- a set RES of possible responses the object can return, and
- a transition relation $\delta \subseteq Q \times OPS \times RES \times Q$.

This specification describes how the object behaves if it is accessed by one operation at a time. If $(q, op, res, q') \in \delta$, it means that a possible result of applying operation op to an object in state q is that the object moves to state q' and returns the response res to the process that invoked op.

[1] For example, an erroneous claim is made in two recent papers [1, 11] that explicitly permit nondeterministic objects and make no restriction that the nondeterminism of the objects should be finite. The latter paper states that "linearizability is a safety property, so its violation can be detected with a finite prefix of an execution history." Using the definitions given in that paper, this statement is false. However, this does not affect the correctness of that paper's main results because those results are about objects with finite nondeterminism.

2.2 Infinite Nondeterminism

- We say that an object is *deterministic* if, for all $q \in Q$ and $op \in OPS$, there is at most one pair (res, q') such that (q, op, res, q') is in the object's transition relation δ.
- An object has *finite nondeterminism* if, for all $q \in Q$ and $op \in OPS$, the set of possible outcomes $\{(res, q') : (q, op, res, q') \in \delta\}$ is finite.

Dijkstra [6] argued that infinite nondeterminism should not arise in computing systems. He showed, for example, the functionality of nondeterministically choosing an arbitrary positive integer cannot be implemented in a reasonable sequential programming language. Nevertheless, there is a significant literature on infinite nondeterminism. For example, Apt and Plotkin [4] observed that infinite nondeterminism can arise naturally in systems that guarantee fairness. Consider a system of two processes P and Q programmed as follows, using a shared boolean variable *Stop* that is initially false.

```
Process $P$:              Process $Q$:
$Stop$ := true            $x := 1$
                          do until $Stop$
                              $x := x + 1$
                          end do
                          print $x$
```

If these processes are run in a fair environment, where each process is guaranteed to be given infinitely many opportunities to take a step (but there is no bound on the relative speeds of the processes), Q will choose and print an arbitrary positive integer. Thus, at the right level of abstraction, this system implements a choice with infinite nondeterminism. In the context of shared-memory computing, objects with infinite nondeterminism have also occasionally arisen (*e.g.*, [14]).

2.3 Histories

A (finite or infinite) sequence of invocation and response events is called a *history* and this is how we model an execution. We assume that processes are *sequential*: a process executes (at most) one operation at a time. Of course, the fact that processes are (individually) sequential does not preclude different processes from concurrently invoking operations on the same shared object.

The total order relation on the set of events induced by H is denoted $<_H$. A history abstracts the real-time order in which the events occur. We assume that simultaneous (invocation or response) events do not affect one another, so that we can arbitrarily order simultaneous events.

A *local* history of p_i, denoted $H|p_i$, is a projection of H on process p_i: the subsequence H consisting of the events generated by p_i. Two histories H and H' are said to be *equivalent* if they have the same local histories, *i.e.*, for each process p_i, $H|p_i = H'|p_i$.

As we are interested only in histories generated by sequential processes, we focus on histories H such that, for each process p_i, $H|p_i$ is *well-formed*: it starts with an invocation, followed by a response (the matching response associated with the same object), followed by another invocation, and so on.

An operation is said to be *complete* in a history if the history includes both the events corresponding to the operation's invocation and its response. Otherwise, we say that the operation is *pending.* A history is *complete* if it has no pending operations and *incomplete* otherwise.

A history H induces an irreflexive partial order on its operations as follows. Let op and op' be two operations. Informally, operation op precedes operation op', if op terminates before op' starts. More precisely:

$$\big(op \rightarrow_H op'\big) \overset{\text{def}}{=} \big(resp[op] <_H inv[op']\big).$$

Two operations op and op' are said to *overlap* (we also say are *concurrent*) in a history H if neither $op \rightarrow_H op'$ nor $op' \rightarrow_H op$.

2.4 Sequential Histories

A history is *sequential* if its first event is an invocation, and then (1) each invocation event, except possibly the last, is immediately followed by the matching response event, and (2) each response event, except possibly the last, is immediately followed by an invocation event. A complete sequential history always ends with a response event. A history that is not sequential is said to be *concurrent.* Given that a sequential history S has no overlapping operations, the associated partial order $\rightarrow_S$ defined on its operations is actually a total order.

Let $S|X$ (S at X) denote the subsequence of history S made up of all the events involving object X. We say that a sequential history S is *legal* if, for each object X, the sequence $X.op_1, X.res_1, X.op_2, X.res_2, \ldots$ satisfies the sequential specification $(Q, q_0, OPS, RES, \delta)$ of X in the following sense: there exists $q_1, q_2, \ldots$ in Q such that $(q_{i-1}, op_i, res_i, q_i) \in \delta$ for all i.

3 Linearizability

Linearizability [9] basically requires that each operation on an object appears to execute at some indivisible point in time, also called the operation's *linearization point*, between the invocation and response of the operation. Linearizability provides the illusion that the operations issued by the processes on the shared objects are executed one after another.

We first define linearizability for complete histories H, *i.e.*, histories without pending operations, and then extend the definition to incomplete histories. A complete history H is *linearizable* if there is a "witness" history S such that:

1. H and S are equivalent,
2. S is sequential and legal, and
3. $\rightarrow_H \subseteq \rightarrow_S$.

This means that for a history H to be linearizable, there must exist a permutation S of H, which satisfies the following requirements. First, S has to be indistinguishable from H to any process. Second, S has to be sequential (interleaving the process histories at the granularity of complete operations) and legal (respecting the sequential specification of each object). Notice that, as S is sequential, $\rightarrow_S$ is a total order. Finally, S must also respect the real-time occurrence order of the operations as defined by $\rightarrow_H$. Such a sequential history S is called a *linearization* of H.

The definition of linearizability is extended to incomplete histories as follows. An incomplete history H is linearizable if H can be *completed*, *i.e.*, modified in such a way that every invocation of a pending operation is either removed or completed with a response event, so that the resulting (complete) history H' is linearizable. Intuitively, H' is obtained by adding response events to certain pending operations of H, as if these operations have indeed been completed, but also by removing invocation events from some of the pending operations of H. We require however that all complete operations of H are preserved in H'.

When proving that an algorithm implements a linearizable object, we need to prove that all histories generated by the algorithm are linearizable. A history H may allow for several different linearizations.

4 Linearizability Is Not a Safety Property

4.1 Safety

Intuitively, safety properties ensure that nothing "bad" ever happens. More specifically, a *safety property* is a set of histories that is non-empty, prefix-closed and limit-closed. Thus, a set P of histories is a safety property if it satisfies the following three conditions.

- P is *non-empty*: $P \neq \{\}$.
- P is *prefix-closed*: if $H \in P$, then for every prefix H' of H, $H' \in P$.
- P is *limit-closed*: for every infinite sequence $H_0, H_1, \ldots$ of histories, where each H_i is a prefix of H_{i+1} and each $H_i \in P$, the limit history $H = \lim_{i \to \infty} H_i$ is in P.

To ensure that a safety property P holds for a given implementation, it is thus enough to show that every *finite* history of the implementation is in P; an execution is in P if and only if each of its *finite* prefixes is in P. Indeed, every infinite history of an implementation is the limit of some sequence of ever-extending finite histories and thus should also be in P.

4.2 Counterexample

Theorem 1. *Linearizability is not a safety property*

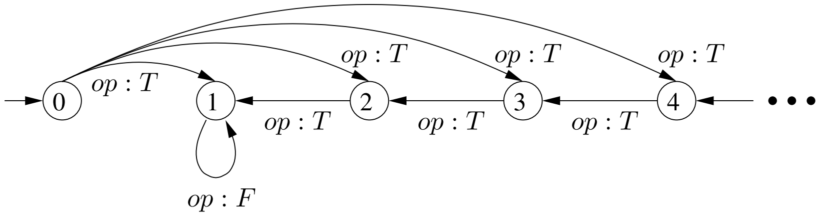

Fig. 1. The countdown object.

Proof. We define a type of object called a *countdown object*, which provides a single operation *op* that outputs T or F. The first invocation of *op* nondeterministically picks a positive integer k. The object returns T for the first k invocations of *op*. After that, it returns F for all remaining invocations of *op*. Formally, this type has the following sequential specification, which is illustrated in Fig. 1.

$$\begin{aligned} Q &= \mathbb{N} \\ q_0 &= 0 \\ OPS &= \{op\} \\ RES &= \{T, F\} \\ \delta &= \{(0, op, T, k) : k \geq 1\} \cup \{(1, op, F, 1)\} \cup \{(k, op, T, k-1) : k \geq 2\} \end{aligned}$$

Consider the following infinite sequential history H that uses a single countdown object X.

$$\begin{gathered} inv[X.op \text{ by } p], \\ resp[X.T \text{ by } p], \\ inv[X.op \text{ by } p], \\ resp[X.T \text{ by } p], \\ inv[X.op \text{ by } p], \\ resp[X.T \text{ by } p], \\ \vdots \end{gathered}$$

We first show that this history is not legal (and hence not linearizable). If we try to assign any positive integer state k to the object X after the first operation has been performed, then the states of the object after the next $k-1$ operations must be $k-1, k-2, k-3, \ldots, 1$. Thus, the $(k+1)$th invocation of *op* in the execution would have to return F. Since there is no way to assign states to the object consistent with all responses in H, we conclude that H is not legal.

Now consider any finite prefix H' of H. We show that H' is legal (and hence linearizable). Let k be the number of complete operations in H'. We can assign the sequence of states $k, k-1, \ldots, 2, 1$ to X. Note that $(0, op, T, k)$ and

$(i, op, T, i-1)$ (for $2 \leq i \leq k$) are transitions of a countdown type, so this sequence of states satisfies the definition of legality for H'.

Let H_i be the prefix of H consisting of the first i complete operations. Then, for all i, H_i is linearizable and H_i is a prefix of H_{i+1}. However, $H = \lim_{i \to \infty} H_i$ is not linearizable. Thus, the property of being linearizable is not limit-closed, and linearizability is not a safety property for this object specification. □

Remark 2. Because the execution used in the proof of Theorem 1 is a sequential execution, the argument in fact shows that even legality is not a safety property for the countdown object type, since the sequential execution is not legal but every prefix of it is. Moreover, since the execution in the proof is by a single process, it also demonstrates that other consistency conditions that are weaker than linearizability (such as sequential consistency) are also not safety properties for the countdown object.

5 When Linearizability Is a Safety Property

We now show that a slight generalization of König's (Infinity) Lemma enables us to show that linearizability, when restricted to objects with finite nondeterminism, is a safety property. König's Lemma can be formulated as follows.

Lemma 3. *(König's Lemma [10]) Let G be an infinite directed graph such that (1) each vertex of G has finite outdegree, (2) each vertex of G is reachable from some root vertex of G (a vertex with zero indegree), and (3) G has only finitely many roots. Then G has an infinite path with no repeated vertices starting from some root.*

Theorem 4. *Linearizability is a safety property for object types with finite nondeterminism.*

Proof. Consider any object type with finite nondeterminism. The set of linearizable histories is non-empty, since the empty history (consisting of 0 events) is trivially linearizable. We show that the set of linearizable histories is prefix- and limit-closed.

Consider a linearizable history H. We show that any prefix H' of H is also linearizable. Let S be any linearization of H. Let sequential history S' be the shortest prefix of S that contains all complete operations of H'.

We claim that S' is a linearization of H'. We complete H' by appending responses that are present in S' but not in H' to the end of H' and removing operations that do not appear in S'. Note that only incomplete operations are removed from H' since all complete ones appear in S'. Let $\bar{H}'$ denote the resulting complete history.

First we show that complete histories S' and $\bar{H}'$ contain the same set of operations. Any operation in $\bar{H}'$ must also be in S' (since all operations not in S' are removed when forming $\bar{H}'$). To derive a contradiction, suppose that S' contains an operation op that does not appear in $\bar{H}'$. Since only operations that do not appear in S' were removed from H' to obtain $\bar{H}'$, *op* does not appear in H' either. Since S' is the shortest prefix of S that contains all complete operations of H', the last operation op' in S' must be a complete operation in H'. Thus, $op \neq op'$. Since op' is complete in H' and *op* does not appear in H', $op' <_H op$. But $op <_S op'$, contradicting the assumption that S is a linearization of H.

Since S' is a prefix of a legal history S, it is also legal. Moreover, it also respects the real-time order in $\bar{H}'$: if $op <_{\bar{H}'} op'$, then $op <_{S'} op'$ (otherwise, S would violate the real-time order in H). Since S and $\bar{H}'$ contain the same set of operations, S' respects the real-time order of $\bar{H}'$, and local histories are well-formed, S' is equivalent to $\bar{H}'$: local histories in S' and $\bar{H}'$ are identical.

So, S' is a linearization of H' and, thus, linearizability is prefix-closed.

To prove the limit-closed property, we consider an infinite sequence of ever-extending linearizable histories $H_0, H_1, H_2, \ldots$. Our goal is to show that $H = \lim\limits_{i\to\infty} H_i$ is linearizable. We assume that H_0 is the empty history and each H_{i+1} is a one-event extension of H_i. (By prefix-closedness, each prefix of every H_i is linearizable, so there is no loss of generality in this assumption.)

Now we construct a directed graph $G = (V, E)$ as follows. Vertices of G are all tuples (H_i, S, W), where $i \in \mathbb{N}$, S is any linearization of H_i that ends with a *complete* operation present in H_i, and W is a sequence of states that witnesses the legality of S. There is a directed edge $((H_i, S, W), (H_j, S', W'))$ in G if and only if $j = i + 1$, S is a prefix of S' and W is a prefix of W'.

Note that for each H_i there is at least one vertex (H_i, S, W), since H_i is linearizable and if we remove all operations at the end of the linearization that are incomplete in H_i, we still have a linearization of H_i (the incomplete operations can also be removed from H_i to obtain a completion of H_i). Moreover, since S is necessarily legal, there exists a witness W for it. Thus, the graph G contains infinitely many vertices.

We use König's lemma to show that the resulting graph G contains an infinite path $(H_0, S_0, W_0), (H_1, S_1, W_1), \ldots$ and the limit $\lim\limits_{i\to\infty} S_i$ is a linearization of the infinite limit history H. The legality of $\lim\limits_{i\to\infty} S_i$ is witnessed by the infinite sequence of states $\lim\limits_{i\to\infty} W_i$.

First we observe that for each vertex (H_{i+1}, S', W') (with $i \geq 0$), there is an edge into the vertex from some vertex (H_i, S, W). There are two cases to consider.

- The last operation *op* of S' is a complete operation in H_i. In this case, S' is also a linearization of H_i. Indeed, even if the last event of H_{i+1} is the invocation of a new operation op', this operation cannot appear in S': it can only appear before *op* in S' violating the real-time order in H_{i+1}. Thus, (H_i, S', W') is a vertex in G and there is an edge from it to (H_{i+1}, S', W').
- The last operation *op* of S' is not a complete operation in H_i. But since S' ends with an operation *op* that is complete in H_{i+1} and H_{i+1} extends H_i with one event only, we conclude that the last event of H_{i+1} is the response of *op*. Thus, H_i and H_{i+1} contain the same set of operations, except that *op* is incomplete in H_i. Let S be the longest prefix of S' that ends with a complete

operation in H_i. Let W be the prefix of W' whose length corresponds is the number of operations in S'. Since W witnesses the legality of S, W' witnesses the legality of S'. Also, only incomplete operations in H_i do not appear in S. Thus, S is a linearization of H_i and (H_i, S, W) is a vertex in G and there is an edge from it to (H_{i+1}, S', W').

It follows that the graph G has only one root vertex, (H_0, S_0, W_0), where H_0, S_0 and W_0 are empty sequences, and moreover that every vertex is reachable from this root.

Now we show that the outdegree of every vertex of G is finite. There are only finitely many operations in H_{i+1} and each linearization of H_{i+1} is a permutation of these operations, so there can only be finitely many linearizations S' of H_{i+1}. Moreover for any finite-length sequential history S' there can only be finitely many witnesses to the legality of S', since the number of possible states after any finite number of operations has been performed is finite. (This is where we use the assumption that the object's specification has finite nondeterminism.) Thus, there are only finitely many vertices of the form (H_{i+1}, S', W'). Since all outgoing edges of any vertex (H_i, S, W) are directed to vertices of the form (H_{i+1}, S', W'), the outdegree of every such vertex is also finite.

By Lemma 3, G contains an infinite path starting from the root vertex: $(H_0, S_0, W_0), (H_1, S_1, W_1), \ldots$. Let $S = \lim\limits_{i\to\infty} S_i$ and $W = \lim\limits_{i\to\infty} W_i$. First, note that W witnesses the legality of S. We argue now that the S is a linearization of the infinite history H. Let H' be the completion of H obtained by removing the incomplete operations of H that are not included in S and inserting into H response events for incomplete operations of H that are included in S. (The response events should be inserted in the order they occur in S and the response to an operation *op* should be inserted after the response to any operation that appears before *op* in S.) By construction, S is equivalent to H', and S respects the real-time order of H; otherwise there would be a vertex (H_i, S_i) such that S_i is not equivalent to H_i or violates the real-time order of H_i. Thus, S is indeed a linearization of H, which concludes the proof that linearizability is a safety property. □

6 Backward Simulations

A backward simulation [13] is a technique that is sometimes used to show that an implementation of a shared object is linearizable (for example, [5, 7]). A backward simulation from a system A to a system A' (which have state sets Q and Q', respectively) is a relation $bsr \subseteq Q \times Q'$ with the following properties.

1. For every state s of A, there is a state s' of A' such that $(s, s') \in bsr$.
2. If there is a transition α from state s_1 to state s_2 in A and $(s_2, s_2') \in bsr$ then there is a state s_1' and a sequence of transitions α' of A' such that

$(s_1, s'_1) \in bsr$, α' moves from state s'_1 to s'_2, and the sequence of externally observable events[2] is the same in α and α'.

3. If q_0 is a possible initial state of A and $(q_0, q'_0) \in bsr$ then q'_0 is a possible initial state of A'.

If such a backward simulation exists from an implementation automaton to an abstract automaton that specifies correct linearizable behaviour, it is easy to prove that every finite history of the implementation is also a history of the abstract automaton, and hence linearizable. Intuitively, given a history H of the implementation A, we start from the final state of that history and find a matching state of the abstract automaton A' using property 1. Then, working backwards step by step, we build a history H' of A' by finding, for each transition of H, a sequence of transitions to prepend to H' using property 2. Finally, when we reach the beginning of H we observe, using property 3, that the history we have built could take place starting from an initial state of A'. Moreover, by construction the two histories have the same sequence of externally visible events.

Thus, if we can build a backward simulation from the implementation to the abstract automaton that specifies linearizable behaviour *and* if linearizability is a safety property, it follows that the implementation is linearizable. However, if linearizability is not a safety property, then the existence of the backward simulation does *not* necessarily imply that the implementation is linearizable. In fact, we can provide an example of an incorrect implementation of the countdown object where there is a backward simulation between the implementation and the abstract automaton.

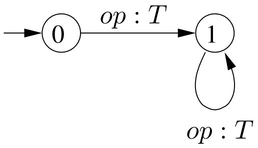

Fig. 2. An incorrect implementation of the countdown object.

Consider the following trivial (but incorrect) implementation of a countdown object: to perform an *op* on the countdown object, a process immediately returns T. One way to model this implementation is the automaton shown in Fig. 2. The reason that this implementation is incorrect is that it is possible for the implementation to return T forever in an infinite execution, something that is not permitted by the specification of the countdown object. Nevertheless, there is a backward simulation relation from the implementation to the countdown

[2] In the context of implementations of shared object of type T, observable events are just invocations and responses on the object of type T.

type. Let

$$bsr = \{(0,0)\} \cup \{(1,k) : k \geq 1\}.$$

It is easy to verify that *bsr* satisfies the three properties that define a backward simulation relation using the following correspondence between actions.

α	s'_2	α'	external actions
$0 \to 1$	k, where $k \geq 1$	$0 \to k$	$op : T$
$1 \to 1$	k, where $k \geq 1$	$k+1 \to k$	$op : T$

Thus, for objects with infinite nondeterminism, backward simulations are not necessarily a sound technique for proving linearizability (unless one can also prove that linearizability is a safety property for the object type considered). In view of Theorem 4, proving linearizability with a backward simulations *is* sound for any type with finite nondeterminism.

7 Concluding Remarks

For clarity, we have used the terms finite nondeterminism, rather than bounded nondeterminism (which is often used in the literature) because an object may have finite nondeterminism even when there is no bound B such that the number of possible responses to an operation is always bounded by B. For example, consider the *bag* object type, which stores a set of natural numbers and provides two operations: insert(k), which adds k to the set, and delete, which nondeterministically removes and returns an arbitrary element of the set. It has the following formal specification.

$$\begin{aligned}
Q &= \mathcal{P}(\mathbb{N}) \\
q_0 &= \{\} \\
OPS &= \{\text{insert}(k) : k \in \mathbb{N}\} \cup \{\text{delete}\} \\
RES &= \{ack, empty\} \cup \mathbb{N} \\
\delta &= \{(S, \text{insert}(k), ack, S \cup \{k\}) : S \subseteq \mathbb{N}, k \in \mathbb{N}\} \cup \\
&\quad\ \{(S, \text{delete}, k, S - \{k\}) : S \subseteq \mathbb{N}, k \in S\} \cup \{(\{\}, \text{delete}, empty, \{\})\}
\end{aligned}$$

Although the number of nondeterministic choices available to a delete operation depends on the current state, and there is no *a priori* bound on this number, the bag object does have finite nondeterminism, so Theorem 4 says that linearizability *is* a safety property for the bag object.

We have shown in this paper that, strictly speaking, linearizability is not a safety property if infinite nondeterminism is permitted in the definition of object types. This points out the importance of considering carefully whether theorems proved about shared-memory systems apply to arbitrary nondeterministic object type specifications, or whether one should make the (often reasonable) restriction that object types must have finite nondeterminism. In particular, this shows that if linearizability is established by proving that every finite run is linearizable, for example by using a backward simulation, then there is an additional proof obligation to show that linearizability is a safety property for the particular type, for example by showing that the type specification has finite nondeterminism and then applying Theorem 4 of this paper. One open question raised by this work would be to give a precise characterization of the object types for which linearizability *is* a safety property.

Acknowledgements. The model section and definition of linearizability are based on lecture notes written by the first author with Michel Raynal and then with Petr Kuznetsov. The proof of Theorem 2 is inspired by a proof by Petr Kuznetsov, itself inspired by a proof by Nancy Lynch [12]. We thank Franck van Breugel for helpful discussions.

References

1. Adhikari, K., Street, J., Wang, C., Liu, Y., Zhang, S.: Verifying a quantitative relaxation of linearizability via refinement. In: Proc. 20th International Symposium on Model Checking Software. LNCS, vol. 7976, pp. 24–42 (2013)
2. Alpern, B., Schneider, F.B.: Defining liveness. Inf. Process. Lett. 21(4), 181–185 (Oct 1985)
3. Alpern, B., Schneider, F.B.: Recognizing safety and liveness. Distributed Computing 2(3), 117–126 (Sep 1987)
4. Apt, K.R., Plotkin, G.D.: A cook's tour of countable nondeterminism. In: 8th Colloquium on Automata, Languages and Programming. LNCS, vol. 115, pp. 479–494 (1981)
5. Colvin, R., Groves, L., Luchangco, V., Moir, M.: Formal verification of a lazy concurrent list-based set algorithm. In: Proc. 18th International Conference on Computer Aided Verification. LNCS, vol. 4144, pp. 475–488 (2006)
6. Dijkstra, E.W.: On nondeterminacy being bounded. In: A discipline of programming, chap. 9. Prentice-Hall (1976)
7. Doherty, S., Moir, M.: Nonblocking algorithms and backward simulation. In: Proc. 23rd International Symposium on Distributed Computing. LNCS, vol. 5805, pp. 274–288 (2009)
8. Herlihy, M.P.: Wait-free synchronization. ACM Trans. Progr. Lang. Syst. 13(1), 123–149 (Jan 1991)
9. Herlihy, M.P., Wing, J.M.: Linearizability: a correctness condition for concurrent objects. ACM Trans. Progr. Lang. Syst. 12(3), 463–492 (Jul 1990)
10. König, D.: Über eine Schlussweise aus dem Endlichen ins Unendliche. Acta Litterarum ac Scientiarum Regiae Universitatis Hungaricae Francisco-Josephinae: Sectio Scientiarum Mathematicarum 3, 121–130 (1927), also in chapter VI of Dénes König, *Theory of Finite and Infinite Graphs*, Birkhäuser, Boston, 1990.
11. Liu, Y., Chen, W., Liu, Y.A., Sun, J., Zhang, S.J., Dong, J.S.: Verifying linearizability via optimized refinement checking. IEEE Transactions on Software Engineering 39(7), 1018–1039 (2013)
12. Lynch, N.: Distributed Algorithms, chap. 13. Morgan Kaufmann (1996)
13. Lynch, N.A., Vaandrager, F.W.: Forward and backward simulations. Information and Computation 121(2), 214–233 (Sep 1995)
14. Schenk, E.: The consensus hierarchy is not robust. In: Proc. 16th ACM Symposium on Principles of Distributed Computing. p. 279 (1997)