Fast linearizability testing

Alex Horn and Daniel Kroening

University of Oxford

February 5, 2015

Abstract

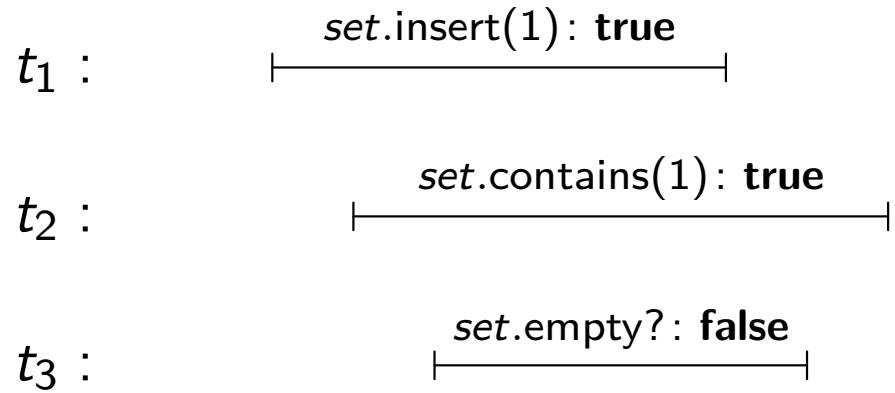
Linearizability is the de facto correctness criteria for concurrent data structures and consensus-based distributed systems. In this talk, I present some work-in-progress on a new tool for testing linearizability. We experimentally show that our tool can handle problems where existing implementations timeout or run out of memory. Our experiments include shared memory programs (Intel's TBB and Siemens' EMBB library) and a large-scale distributed system (etcd) using Aphyr's testing framework (Jepsen).

Outline

Example: history

Consider a concurrent data type for unordered sets with the usual 'insert', 'contains' and 'empty?' operations.

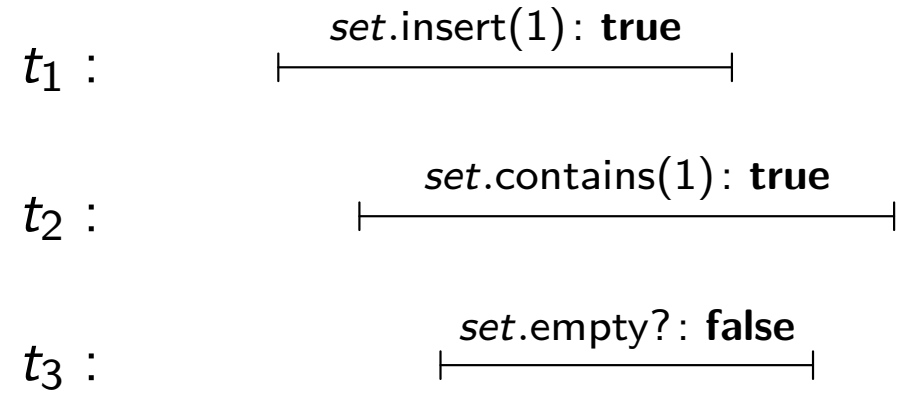
Suppose three threads execute these operations concurrently, and we record their operations in a so-called **history** (let it be H_1):



What does it mean for this execution to be correct (or incorrect)? This is answered by the concept of **linearizability** (see next slides).

Example: linearizability

Let H_1 be the history from the previous slide:



Then H_1 is linearizable because we can reorder the operations to, say,

i. *set*.insert(1) : **true**; *ii*. *set*.empty? : **false**; *iii*. *set*.contains(1) : **true**

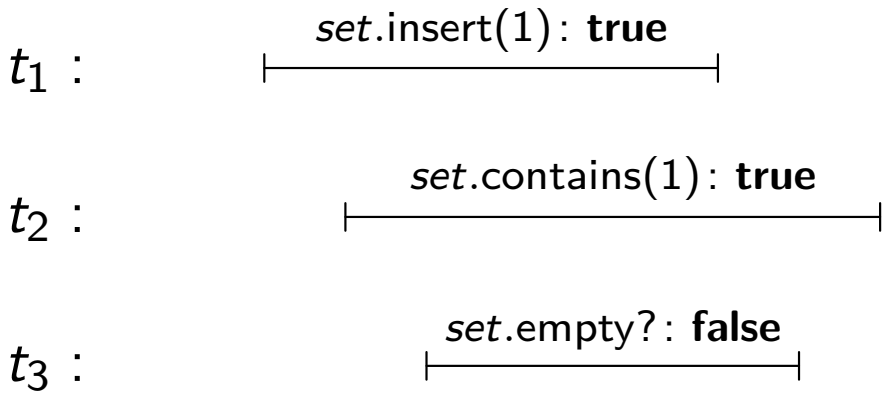
which satisfies the specification of a sequential set.

The problem (to be formalized next) is to automatically find such a reordering given any H and sequential specification.

Definition (History)

Let $E \triangleq \{\text{call}, \text{ret}\} \times \mathbb{N}$ be the set of **events**. For all n in $\mathbb{N}$, define $\text{call}_n \in E$ to be a **call** and $\text{ret}_n \in E$ to be a **return**. A **history** H is a sequence of such events, totally ordered by $\preceq_H$. For all events e in E, $obj(e)$ and $m(e)$ denotes the **object** and **method** of e, respectively. The **length** of H, written $|H|$, is the number of events in H.

Example



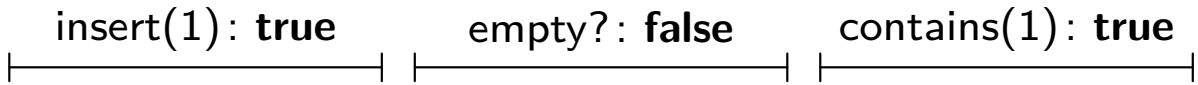
$H_1 \triangleq \langle \text{call}_1, \text{call}_2, \text{call}_3, \text{ret}_1, \text{ret}_3, \text{ret}_2 \rangle$ where $obj(\text{call}_1) =$ '*set*' and $m(\text{call}_1) =$ 'insert(1) : **true**', $|H_1| = 6$ and $\text{call}_3 \preceq_{H_1} \text{ret}_1$ etc.

Definition (Sequential history)

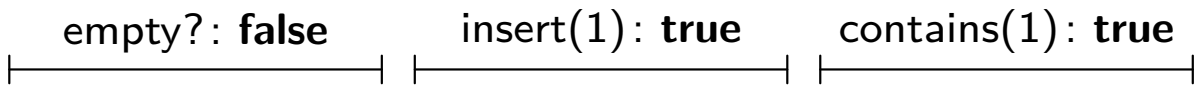
Let $e, e' \in E$ be events in history H. If e is a call and e' is a return in H, both are **matching** whenever $e \preceq_H e'$ and their objects and operations are equal, i.e. $obj(e) = obj(e')$ and $m(e) = m(e')$. A history is **complete** if every call has a matching return. A complete history H is **sequential** if it alternates between matching calls and returns (necessarily starting with a call).

Example

H_2 is a sequential history:



And so is H_3 (albeit not maybe what we expect):

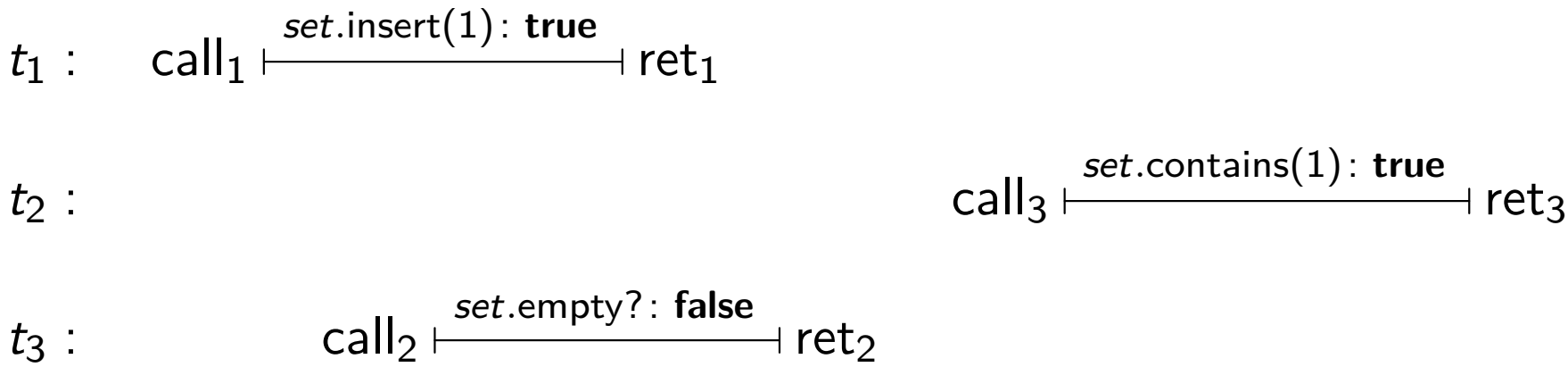


Definition (Happens-before)

Given a history H, the **happens-before** relation is defined to be a partial order $<_H$ over calls e and e' such that $e <_H e'$ whenever $\mathsf{ret}(e) \preceq_H e'$, i.e. e's matching return occurs before e' in H.

Example

Let H_5 be the following history:



- $\mathsf{call}_1 \not<_H \mathsf{call}_2$ and $\mathsf{call}_2 \not<_H \mathsf{call}_1$, i.e. both **happen concurrently**;
- $\mathsf{call}_1 <_H \mathsf{call}_3$ and $\mathsf{call}_2 <_H \mathsf{call}_3$, i.e. call_1 **happens-before** call_3 etc.

Problem statement

Definition (Linearizability)

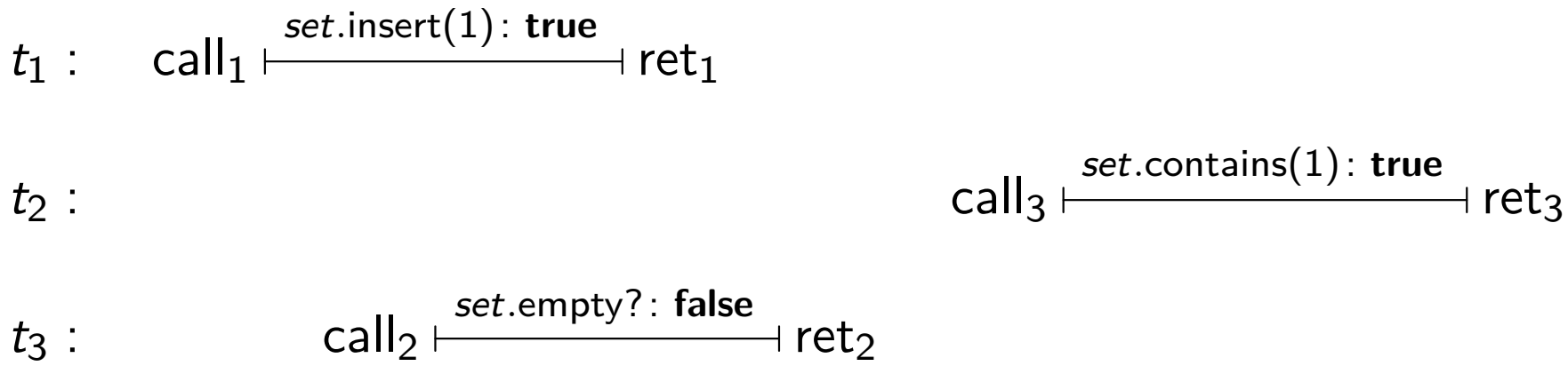
Denote with ϕ a specification of a sequential data type, i.e. ϕ is a unary predicate on sequential histories. A ϕ-**sequential history** is a sequential history H that satisfies $\phi(H)$. A history H is **linearizable with respect to** ϕ if it can be extended to a complete history H' (by appending zero or more returns to H) and there is a ϕ-sequential history S such that

- **L1** H' is equivalent to S, i.e. H' and S are are equal as sets;
- **L2** $<_H \subseteq <_S$, i.e. if e precedes e' in H, the same is true in S.

Decision problem: Given H and ϕ, decide whether H is linearizable with respect to ϕ.

Example: linearizability (with our formalization)

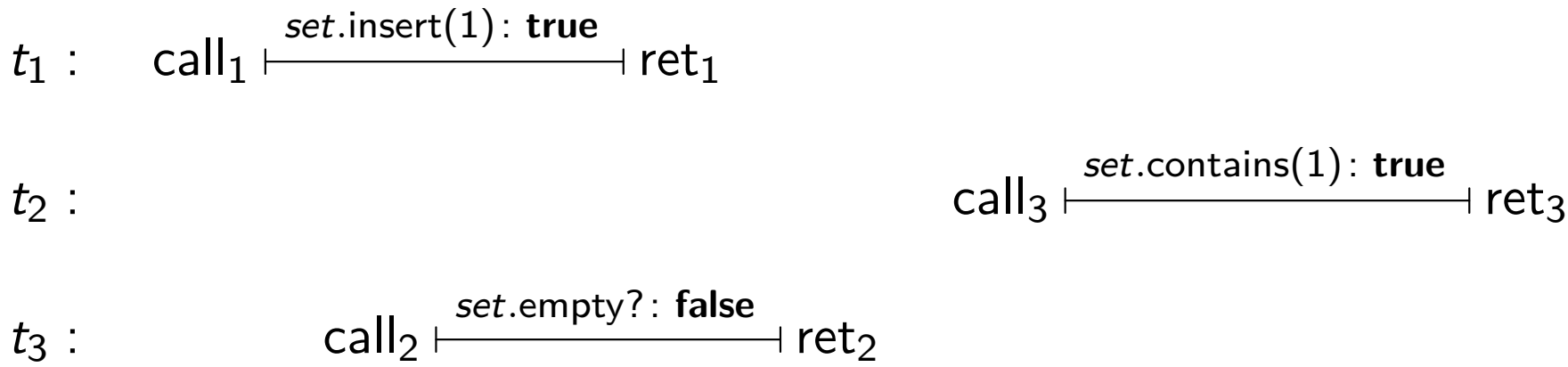
Recall H_5:



Question: Is H_5 linearizable with respect to ϕ_{set}?

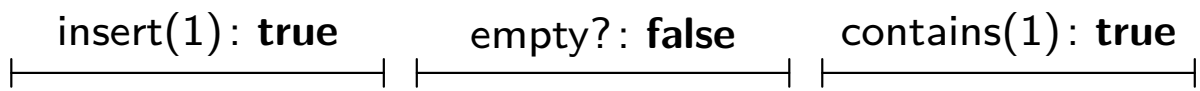
Example: linearizability (with our formalization)

Recall H_5:



Question: Is H_5 linearizable with respect to ϕ_{set}? Yes!

H_2 is a witness for a ϕ_{set}-sequential history that respects $<_{H_5}$:



In general, to check linearizability, we need to consider all permutations of a history H. That's $O(|H|!)$.

Theorem (Complexity)

Given a history H and specification ϕ, the linearizability problem — i.e. whether H is linearizable with respect to ϕ — is NP-complete.

Proof.

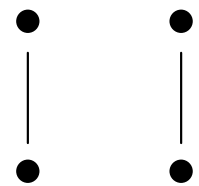
The problem is clearly in NP whenever the specification of the sequential data type is deterministic and can be run in polynomial time. For NP-hardness, by reduction from SAT. Let ψ be a propositional logic formula. Let M_ψ be a model for a partial Boolean assignment of the variables in ψ. Let H be a history where every operation sets a variable to either true or false, and all operations happen concurrently, i.e. are unordered by $<_H$. The specification is satisfied whenever all variables in M_ψ have been assigned a value without causing a contradiction (any later assignments are ignored). Then H is linearizable with respect to M_ψ if and only if ψ is SAT. Hence linearizability checking is NP-complete. □

About our decision procedure

Definition (Interval order)

A partially ordered set $(P, \leq)$ is an **interval order** whenever, for all x, y, u, v in P, if $x \leq y$ and $u \leq v$, then $x \leq v$ or $u \leq y$.

Note that $(P, \leq)$ is an interval order if and only if no restriction of $(P, \leq)$ is isomorphic to the following Hasse diagram:[1]



[1] Rabinovitch, I., *The dimension of semiorders*. J. Comb. Theory (Series A) 25 1978 50–61

About our decision procedure

Theorem

For every complete history H, $<_H$ is an interval order.

Proof.

Assume $x <_H y$ and $u <_H v$. Let $\mathrm{ret}(x)$ and $\mathrm{ret}(u)$ be the matching return of x and u, respectively. Since H is complete, both returns are in H. Recall that $\preceq_H$ denotes the total ordering of events in H. By definition of $<_H$, $\mathrm{ret}(x) \preceq_H y$ and $\mathrm{ret}(u) \preceq_H v$. Since $\preceq_H$ is total, either $\mathrm{ret}(x) \preceq_H \mathrm{ret}(u)$ or $\mathrm{ret}(u) \preceq_H \mathrm{ret}(x)$. The former implies $x <_H v$, whereas the latter implies $u <_H y$, proving the claim. □

Put differently, this talk is about a **decision procedure for a certain class of partial orders**.

Idea: Partitioning scheme

Theorem

A history H is linearizable with respect to a specification if and only if, for each object obj, the restriction of H to obj is linearizable.[2]

We use this theorem to partition a history of operations on an associative data type (e.g. a set or hash map) into sub-histories that only contain operations on the same datum. We call this the **compositional technique**. Note that this technique cannot be applied when there are zero-arg operations such as 'empty?'.

Example

set.insert(1) and *set*.insert(2) are in different sub-histories even though both operations are on the same set. Put differently, we interpret the datum to be *obj*.

[2] This is well-known as "compositionality", see Theorem 3.6.1 in *The Art of Multiprocessor Programming* (Revised Ed.) by Herlihy and Shavit.

Idea: Optimizations

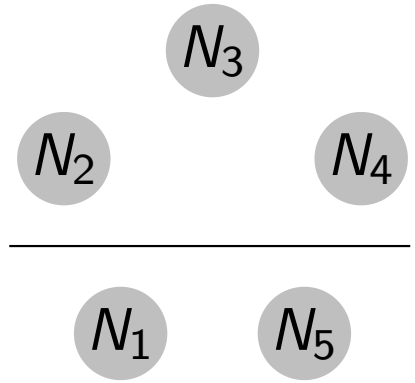
As in SAT solvers, optimizations matter. We've experimentally evaluated many different ways of implementing the decision procedure, including non-chronological backtracking, parallelization, and cache eviction strategies. We also looked at hashing: we exploit the fact that XOR forms an albelian group and by this we get a constant-time, $O(1)$, hash function on bitsets.

Abelian group axioms:

- Closure: for all $x, y \in \mathbb{V}$, $x \oplus y \in \mathbb{V}$;
- Associativity: for all $x, y, z \in \mathbb{V}$, $(x \oplus y) \oplus z = x \oplus (y \oplus z)$;
- Identity: for all $x \in \mathbb{V}$, $x \oplus 0 = x = 0 \oplus x$;
- Inverse: for all $x \in \mathbb{V}$, $x \oplus \tilde{x} = 0$ (here, $\tilde{x} \triangleq x$);
- Commutativity: for all $x, y \in \mathbb{V}$, $x \oplus y = y \oplus x$.

Experiments with distributed systems

For our experiments with distributed systems, we use *Jepsen*, a testing framework that simulates network failures.[3] For example, Jepsen sporadically introduces network partitions in a cluster (on purpose):



Using Jepsen, we have collected histories from *etcd*, a distributed key/value data store.[4] Due to network failures, the etcd histories are rather different from those generated by shared memory programs.

[3] https://github.com/aphyr/jepsen

[4] https://github.com/coreos/etcd

Etcd experiments, comparison with Knossos

Jepsen ships with a linearizability tester called Knossos.[5] Out of 100 etcd histories collected with Jepsen, 80% are non-linearizable (this is harder to check than linearizability, i.e. SAT versus UNSAT).

Crucially, Knossos times out on benchmarks 7 and 99, and runs out of memory on benchmarks 40, 57, 85 and 97. All these benchmarks (except 7) are non-linearizable. In contrast, our linearizability tester completes all 100 benchmarks in around 7 s, averaging $\mathbf{0.07}\,s$ per benchmark. Memory usage of our tool is negligible.

[5] https://github.com/aphyr/knossos

TBB experiments

We tested `concurrent_unordered_set` in Intel's *TBB* library.[6] In our experiments, we start up four processes that randomly insert and remove items. Each process produces $70\,K$ operations. In other words, the length of histories is $4 \times 2 \times 70K = 560\,K$. Those histories are significantly longer than in previous experiments where the limit is $2^{13} \approx 8\,K$ operations per process.[7] For our tool, we get the following results when histories are linearizable:

- LRU=on, O(1) hash: **54 s** [**646 MiB**]
- LRU=off, O(1) hash: $131\,s$ [$9,763\,MiB \approx 10GiB$]
- Compositional: **7 s** [$9,763\,MiB \approx 10GiB$]

The first two times remain stable even if allow 'empty?' checks, but then we cannot use the compositional technique (as noted before).

---

[6] https://www.threadingbuildingblocks.org/

[7] http://www.cs.ox.ac.uk/people/gavin.lowe/LinearizabiltyTesting/

EMBB experiments

We tested `LockFreeStack` in Siemens' EMBB library.[8] Again, we start up four processes that randomly push and pop items. Each process produces $70\,K$ operations, resulting in histories of length $560\,K$. We get the following results when histories are linearizable:

- LRU=on, O(1) hash: $73\,s$ [**704 MiB**]
- LRU=off, O(1) hash: **59 s** [$13,135\,MiB \approx 13\,GiB$]

Things to improve: Currently, we artificially induce non-linearizable histories by changing the initial state. Ideally, we would find real bugs in TBB or EMBB. It would be also interesting to extract histories form Gavin's experiments on hash sets implemented in Scala.[9]

[8] https://github.com/siemens/embb

[9] http://www.cs.ox.ac.uk/people/gavin.lowe/LinearizabiltyTesting/

Concluding remarks

We have presented a linearizability tester with promising results. We beat Knossos and our partitioning scheme shows significant algorithmic improvements over Gavin's linearizability tester. In addition, we have shown that LRU cache eviction can reduce memory consumption by one order of magnitude. In fact, in the TBB experiments, this even decreases runtime.

As future work, it would be nice to extend the idea behind our partitioning scheme to a wider class of operations. I am planning to support other consistency models.

You may download our tool now!
https://github.com/ahorn/linearizability-tester

(Our source code repository contains not only our experiments but also many unit tests to simplify development.)