Grokking linearizability checker

AriesDevil@Datafuselabs

A NP-complete problem

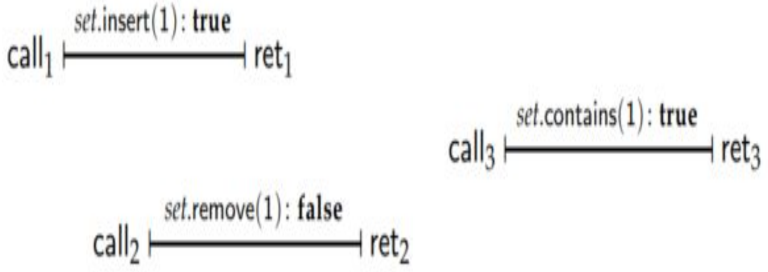
Definition 1 (History)

Call: Let E = {call, ret} × N. For all natural numbers n in N,

$call_n$ = <call, n> in E is called a **call.**

Return: Let E = {call, ret} × N. For all natural numbers n in N,

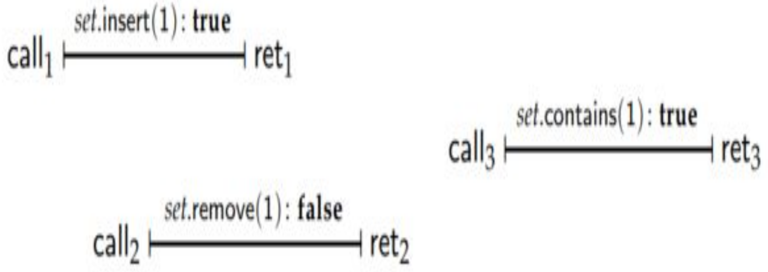
ret_n = <ret, n> in E is called a **return**



Definition 1 (History)

Operation: The invocation of a procedure with input and output arguments is called an **operation.**

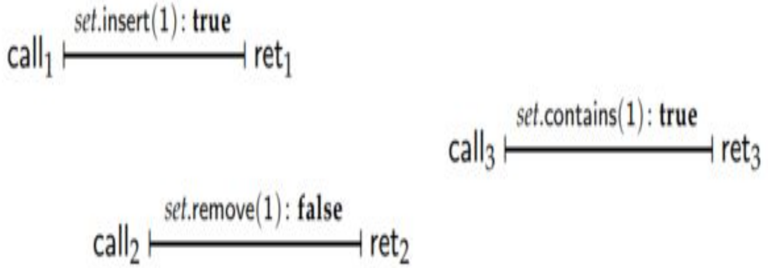
Object: An **object** comprises a finite set of such operations.



Definition 1 (History)

For all e in E, *obj*(e) and *op*(e) denote the object and operation of e.

History: A history is a tuple <H, obj, op> where H is a finite sequence of calls and returns, totally ordered by ordered set H.



Definition 2（Complete and sequential history）

Definition 2 (Complete and sequential history). *Let $e, e' \in E$ and H be a history. 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 $op(e) = op(e')$. A history is called **complete** if every call has a unique matching return. A complete history is called **sequential** whenever it alternates between matching calls and returns (necessarily starting with a call).*

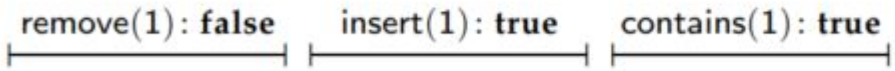
H2 remove(1) : **false** insert(1) : **true** contains(1) : **true**

H3 insert(1) : **true** remove(1) : **false** contains(1) : **true**

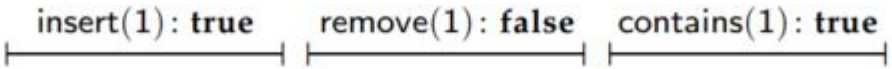
Definition 3 (Specification)

Specification: A specification, denoted by φ , is a unary predicate on sequential histories.

φ(H2) = true



φ(H3) = false



Definition 4
(happens-before & happens concurrently)

Definition 4 (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 e's matching return, denoted by* $\mathsf{ret}(e)$, *precedes e' in H, i.e.* $\mathsf{ret}(e) \preceq_H e'$. *We say that two calls e and e'* ***happen concurrently*** *whenever $e \nless_H e'$ and $e' \nless_H e$.*

Example 4. For the history H_1 in Fig. 1, we get:

- $\mathsf{call}_1 <_{H_1} \mathsf{call}_3$ and $\mathsf{call}_2 <_{H_1} \mathsf{call}_3$, i.e. call_1 and call_2 happen-before call_3;
- $\mathsf{call}_1 \nless_{H_1} \mathsf{call}_2$ and $\mathsf{call}_2 \nless_{H_1} \mathsf{call}_1$, i.e. call_1 and call_2 happen concurrently.

Definition 5（Linearizability）

Definition 5 (Linearizability). *Let ϕ be a specification. 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) and there is a ϕ-sequential history S with the same obj and op functions as H' such that*

L1 *H' and S are equal when seen as two sets of calls and returns;*

L2 $<_H \subseteq <_S$*, i.e. for all calls e, e' in H, if e happens-before e', the same is true in S.*

Definition 6 (P-compositionality)

Definition 6 (*P*-**compositionality**). *Let P be a function that maps a history H to a non-trivial partition of H, i.e. P satisfies $P(H) \neq \{H\}$. A specification ϕ is called P-**compositional** whenever any history H is linearizable with respect to ϕ if and only if, for every history $H' \in P(H)$, H' is linearizable with respect to ϕ. When this equivalence holds we speak of P-**compositionality**.*

Linearizability checker main logic

```
Require: head_entry is such that head_entry.next points to the beginning of history H.
Require: N = 0.5 × |H| is half of the total number of entries reachable from head_entry.
Require: linearized is a bitset (array of bits) such that linearized[k] = 0 for all 0 ≤ k < N.
Require: For all entries e in H, 0 ≤ entry_id(e) < N.
Require: For all entries e and e′ in H, if entry_id(e) = entry_id(e′), then e = e′.
Require: cache is an empty set and calls is an empty stack.
while head_entry.next ≠ null do
    if entry.match ≠ null then                                   ▷ Is call entry?
        ⟨is_linearizable, s′⟩ ← apply(entry, s)                  ▷ Simulate entry's operation
        cache′ ← cache                                           ▷ Copy set
        if is_linearizable then
            linearized′ ← linearized                             ▷ Copy bitset
            linearized′[entry_id(entry)] ← 1                     ▷ Insert entry_id(entry) into bitset
            cache ← cache ∪ {⟨linearized′, s′⟩}                  ▷ Update configuration cache
        if cache′ ≠ cache then
            calls ← push(calls, ⟨entry, s⟩)                      ▷ Provisionally linearize call entry and state
            s ← s′                                               ▷ Update state of persistent data type
            linearized[entry_id(entry)] ← 1                      ▷ Keep track of linearized entries
            LIFT(entry)                                          ▷ Provisionally remove the entry from the history
            entry ← head_entry.next                              ▷ Continue search in shortened history
        else                                                     ▷ Cannot linearize call entry
            entry ← entry.next                                   ▷ Continue search in unmodified history
    else                                                         ▷ Handle "return entry"
        if is_empty(calls) then
            return false                                         ▷ Cannot linearize entries in history
        ⟨entry, s⟩ ← top(calls)                                  ▷ Revert to earlier state
        linearized[entry_id(entry)] ← 0
        calls ← pop(calls)
        UNLIFT(entry)                                            ▷ Undo provisional linearization
        entry ← entry.next
return true
```

Q&A