Linearizability via Order Extension Theorems

Ana Sokolova

Dagstuhl, 25.5.2018
• Part I: Concurrent data structures correctness and performance

• Part II: Order extension results for verifying linearizability

via semantic relaxations

structure and power
Concurrent Data Structures
Correctness and Relaxations
Data structures

- Queue FIFO
  - enq
  - f e d c b a
  - deq

- Stack LIFO
  - push
  - x
  - y
  - z
  - pop

- Pool unordered
  - ins
  - k
  - n
  - m
  - j
  - o
  - l
  - rem
Concurrent data structures

- Queue FIFO
- Stack LIFO
- Pool unordered
Semantics of concurrent data structures

- **Sequential specification** = set of legal sequences
  
  e.g. queues

- **Consistency condition** = e.g. linearizability / sequential consistency

  e.g. queue legal sequence
  enq(1)enq(2)deq(1)deq(2)

  e.g. the concurrent history above is a linearizable queue concurrent history
Consistency conditions

there exists a legal sequence that preserves precedence order

Linearizability [Herlihy, Wing ’90]

\[
t_1: \text{enq}(2) \rightarrow \text{deq}(1) \\
_2: \text{enq}(1) \rightarrow \text{deq}(2)
\]

there exists a legal sequence that preserves per-thread precedence (program order)

Sequential Consistency [Lamport’79]

\[
t_1: \text{enq}(1) \rightarrow \text{deq}(2) \\
_2: \text{deq}(1) \rightarrow \text{enq}(2)
\]

consistency is about extending partial orders to total orders
Performance and scalability

throughput

# of threads / cores

Dagstuhl 25.5.18
Relaxations allow trading correctness for performance.
Relaxing the Semantics

- **Sequential specification** = set of legal sequences
- **Consistency condition** = e.g. linearizability / sequential consistency

- Quantitative relaxations
  Henzinger, Kirsch, Payer, Sezgin, S. POPL13

- Local linearizability
  Haas, Henzinger, Holzer, ..., S, Veith CONCUR16
Lead to scalable implementations

e.g. k-FIFO, k-Stack

k-out-of-order queue

locally linearizable distributed implementation

local inserts / global removes

LLD Φ
LL+D Φ
Performance

Figure 8: Performance and scalability of producer-consumer microbenchmarks with an increasing number of threads on a 40-core (2 hyperthreads per core) machine.

(a) Queues, LL queues, and “queue-like” pools
Performance

![Performance graph](image)

(a) Queues, LL queues, and “queue-like” pools

- MS
- LCRQ
- k-FIFO
- LL+D MS
- LLD LCRQ
- LLD k-FIFO
- 1-RA DQ

**LLD \(\Phi\) performs significantly better than \(\Phi\)**
Performance

(a) Queues, LL queues, and “queue-like” pools
Linearizability via Order Extension Theorems

foundational results for verifying linearizability

joint work with

Harald Woracek
Inspiration

Queue sequential specification (axiomatic)

- \( s \) is a legal queue sequence
  
  iff
  
  1. \( s \) is a legal pool sequence, and
  2. \( \text{enq}(x) <_s \text{enq}(y) \land \text{deq}(y) \in s \implies \text{deq}(x) \in s \land \text{deq}(x) <_s \text{deq}(y) \)

Queue linearizability (axiomatic)

- \( h \) is queue linearizable
  
  iff
  
  1. \( h \) is pool linearizable, and
  2. \( \text{enq}(x) <_h \text{enq}(y) \land \text{deq}(y) \in h \implies \text{deq}(x) \in h \land \text{deq}(y) <_h \text{deq}(x) \)

Henzinger, Sezgin, Vafeiadis CONCUR13

As well as

Reducing Linearizability to State Reachability
[Bouajjani, Emmi, Enea, Hamza]
ICALP15 + …
Concurrent Queues

Data independence $\Rightarrow$ verifying executions where each value is enqueued at most once is sound

Reduction to assertion checking $=$ exclusion of "bad patterns"

Value $v$ dequeued without being enqueued
$$\text{deq} \Rightarrow v$$

Value $v$ dequeued before being enqueued
$$\text{deq} \Rightarrow v \quad \text{enq}(v)$$

Value $v$ dequeued
$$\text{deq} \Rightarrow v \quad \text{deq} \Rightarrow v$$

Value $v_1$ and $v_2$ dequeued in the wrong order
$$\text{enq}(v_1) \quad \text{enq}(v_2) \quad \text{deq} \Rightarrow v_2 \quad \text{deq} \Rightarrow v_1$$
Concurrent Queues

Data independence $\Rightarrow$ verifying executions where each value is enqueued at most once is sound

Reduction to assertion checking $=$ exclusion of "bad patterns"

- Value $v$ dequeued without being enqueued
  - $\text{deq} \Rightarrow v$

- Value $v$ dequeued before being enqueued
  - $\text{deq} \Rightarrow v$, $\text{enq}(v)$

- Value $v$ dequeued twice
  - $\text{deq} \Rightarrow v$, $\text{deq} \Rightarrow v$

- Value $v_1$ and $v_2$ dequeued in the wrong order
  - $\text{enq}(v_1)$, $\text{enq}(v_2)$, $\text{deq} \Rightarrow v_2$, $\text{deq} \Rightarrow v_1$

- Dequeue wrongfully returns empty
  - $\text{deq} \Rightarrow \text{empty}$

- $\text{enq}(v_1)$, $\text{enq}(v_2)$, $\text{deq} \Rightarrow v_1$, $\text{deq} \Rightarrow v_2$, $\text{deq} \Rightarrow v_{n-3}$, $\text{deq} \Rightarrow v_n$
Linearizability verification

Data structure

- signature $\Sigma$ - set of method calls including data values
- sequential specification $S \subseteq \Sigma^*$, prefix closed

Sequential specification via violations

Extract a set of violations $V$, relations on $\Sigma$, such that $s \in S$ iff $s$ has no violations $P(s) \cap V = \emptyset$

Linearizability verification

Find a set of violations $CV$ such that: every interval order with no $CV$ violations extends to a total order with no $V$ violations.

We build $CV$ iteratively from $V$
Pool without empty removals

**Pool sequential specification (axiomatic)**

\( s \) is a legal pool (without empty removals) sequence iff

1. \( \text{rem}(x) \in s \Rightarrow \text{ins}(x) \in s \land \text{ins}(x) <_s \text{rem}(x) \)

**Pool linearizability (axiomatic)**

\( h \) is pool (without empty removals) linearizable iff

1. \( \text{rem}(x) \in h \Rightarrow \text{ins}(x) \in h \land \text{rem}(x) <_h \text{ins}(x) \)

V violations
\( \text{rem}(x) <_s \text{ins}(x) \)

CV violations
\( = V \text{ violations} \)
Queue without empty removals

Queue sequential specification (axiomatic)

\( s \) is a legal queue (without empty removals) sequence
iff
1. \( \text{deq}(x) \in s \implies \text{enq}(x) \in s \land \text{enq}(x) \less s \text{ deq}(x) \)
2. \( \text{enq}(x) \less s \text{ enq}(y) \land \text{deq}(y) \in s \implies \text{deq}(x) \in s \land \text{deq}(x) \less s \text{ deq}(y) \)

Queue linearizability (axiomatic)

\( h \) is queue (without empty removals) linearizable
iff
1. \( \text{rem}(x) \in h \implies \text{ins}(x) \in h \land \text{rem}(x) \less h \text{ ins}(x) \)
2. \( \text{enq}(x) \less h \text{ enq}(y) \land \text{deq}(y) \in h \implies \text{deq}(x) \in h \land \text{deq}(y) \less h \text{ deq}(x) \)

\( V \) violations
\( \text{deq}(x) \less s \text{ enq}(x) \)
and
\( \text{enq}(x) \less s \text{ enq}(y) \land \text{deq}(y) \less s \text{ deq}(x) \)

\( CV \) violations
\( = V \) violations
Pool sequential specification (axiomatic)

\(s\) is a legal pool (with empty removals) sequence iff

1. \(\text{rem}(x) \in s \Rightarrow \text{ins}(x) \in s \land \text{ins}(x) <_s \text{rem}(x)\)
2. \(\text{rem}(⊥) <_s \text{rem}(x) \Rightarrow \text{rem}(⊥) <_s \text{ins}(x) \land \text{ins}(x) <_s \text{rem}(⊥) \Rightarrow \text{rem}(x) <_s \text{rem}(⊥)\)

Pool linearizability (axiomatic)

\(h\) is pool (with empty removals) linearizable iff

1. \(\text{rem}(x) \in h \Rightarrow \text{ins}(x) \in h \land \text{rem}(x) <_h \text{ins}(x)\)
2. \(\ldots \ldots\)

infinitely many CV violations

\(\text{ins}(x_1) <_h \text{rem}(⊥) \land \text{ins}(x_2) <_h \text{rem}(x_1) \land \ldots \land \text{ins}(x_{n+1}) <_h \text{rem}(x_n) \land \text{rem}(⊥) <_h \text{rem}(x_{n+1})\)
It works for

- Pool without empty removals
- Queue without empty removals
- Priority queue without empty removals
- Pool
- Queue
- Priority queue

But not yet for Stack: infinite CV violations without clear inductive structure

Exploring the space of data structures as well as new ideas for problematic cases

Thank You!