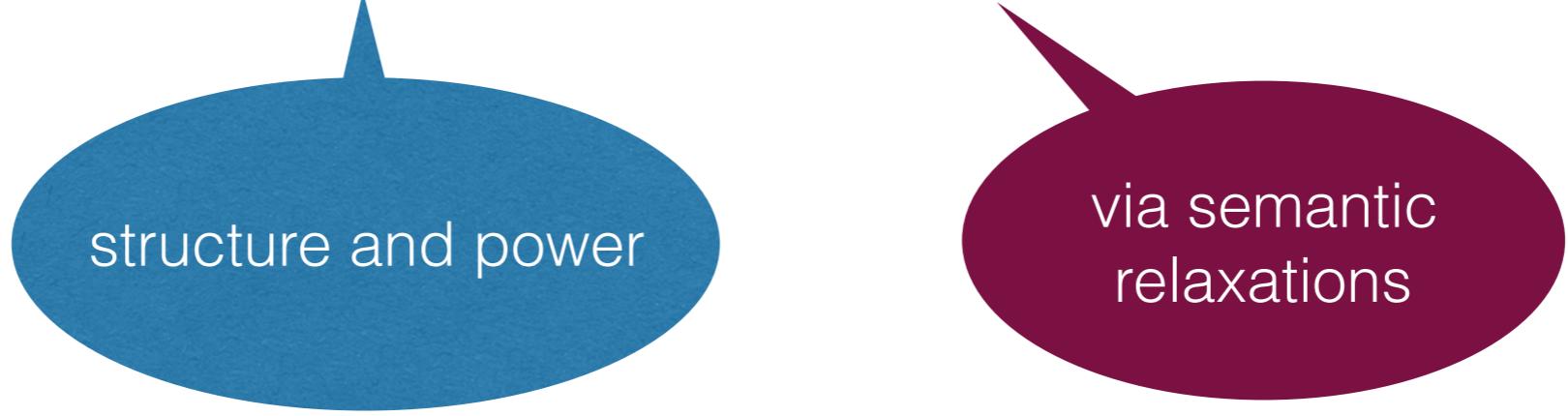


Semantics for Concurrency

Ana Sokolova  UNIVERSITY
of SALZBURG

Introduction - 5.3.19

- Intro: Concurrent data structures
correctness and performance



structure and power

via semantic
relaxations

Linearizability via Order Extension Theorems

joint work with



foundational results
for
verifying linearizability

Inspiration

Queue sequential specification (axiomatic)

s is a legal queue sequence
iff

1. **s** is a legal pool sequence, and
2. $\text{enq}(x) <_s \text{enq}(y) \wedge \text{deq}(y) \in s \Rightarrow \text{deq}(x) \in s \wedge \text{deq}(x) <_s \text{deq}(y)$

Queue linearizability (axiomatic)

Henzinger, Sezgin, Vafeiadis CONCUR13

h is queue linearizable
iff

1. **h** is pool linearizable, and
2. $\text{enq}(x) <_h \text{enq}(y) \wedge \text{deq}(y) \in h \Rightarrow \text{deq}(x) \in h \wedge \text{deq}(y) <_h \text{deq}(x)$

precedence order

As well as
Reducing Linearizability to
State Reachability
[Bouajjani, Emmi, Enea, Hamza]
ICALP15 + ...

Problems (stack)

Stack sequential specification (axiomatic)

s is a legal stack sequence
iff

1. **s** is a legal pool sequence, and
2. $\text{push}(x) <_s \text{push}(y) <_s \text{pop}(x) \Rightarrow \text{pop}(y) \in \mathbf{s} \wedge \text{pop}(y) <_s \text{pop}(x)$

Stack linearizability (axiomatic)

h is stack linearizable
iff

???

1. **h** is pool linearizable, and
2. $\text{push}(x) <_h \text{push}(y) <_h \text{pop}(x) \Rightarrow \text{pop}(y) \in \mathbf{h} \wedge \text{pop}(x) <_h \text{pop}(y)$

Problems (stack)

Stack sequential specification (axiomatic)

s is a legal stack sequence
iff

1. **s** is a legal pool sequence, and
2. $\text{push}(x) <_{\mathbf{s}} \text{push}(y) <_{\mathbf{s}} \text{pop}(x) \Rightarrow \text{pop}(y) \in \mathbf{s} \wedge \text{pop}(y) <_{\mathbf{s}} \text{pop}(x)$

Stack linearizability (axiomatic)

h is stack linearizable
iff

1. **h** is pool linearizable, and
2. $\text{push}(x) <_{\mathbf{h}} \text{push}(y) <_{\mathbf{h}} \text{pop}(x) \Rightarrow \text{pop}(y) \in \mathbf{h} \wedge \text{pop}(x) <_{\mathbf{h}} \text{pop}(y)$

Problems (stack)

t1: push(1)

pop(1)

t2: push(2)

pop(2)

t3: push(3)

pop(3)

not stack
linearizable

Stack linearizability (axiomatic)

~~**h** is stack linearizable
iff~~

1. **h** is pool linearizable, and
2. $\text{push}(x) <_{\mathbf{h}} \text{push}(y) <_{\mathbf{h}} \text{pop}(x) \Rightarrow \text{pop}(y) \in \mathbf{h} \wedge \text{pop}(x) \not<_{\mathbf{h}} \text{pop}(y)$

Linearizability verification

Data structure

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

identify sequences with total orders

Sequential specification via violations

Extract a set of violations V . relations on Σ . such that $\mathbf{s} \in S$ iff \mathbf{s} has no violations

it is easy to find a large CV ,
but difficult to find a small representative

$$\mathcal{P}(\mathbf{s}) \cap V = \emptyset$$

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

legal sequence

concurrent history

Pool without empty removals

Pool sequential specification (axiomatic)

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

1. $\text{rem}(x) \in \mathbf{s} \Rightarrow \text{ins}(x) \in \mathbf{s} \wedge \text{ins}(x) <_{\mathbf{s}} \text{rem}(x)$

V violations
 $\text{rem}(x) <_{\mathbf{s}} \text{ins}(x)$

Pool linearizability (axiomatic)

h is pool (without empty removals) linearizable
iff

1. $\text{rem}(x) \in \mathbf{h} \Rightarrow \text{ins}(x) \in \mathbf{h} \wedge \text{rem}(x) <_{\mathbf{h}} \text{ins}(x)$

CV violations
= V violations

Queue without empty removals

Queue sequential specification (axiomatic)

s is a legal queue (without empty removals) sequence iff

1. $\text{deq}(x) \in \mathbf{s} \Rightarrow \text{enq}(x) \in \mathbf{s} \wedge \text{enq}(x) <_{\mathbf{s}} \text{deq}(x)$
2. $\text{enq}(x) <_{\mathbf{s}} \text{enq}(y) \wedge \text{deq}(y) \in \mathbf{s} \Rightarrow \text{deq}(x) \in \mathbf{s} \wedge \text{deq}(x) <_{\mathbf{s}} \text{deq}(y)$

V violations
 $\text{deq}(x) <_{\mathbf{s}} \text{enq}(x)$
and
 $\text{enq}(x) <_{\mathbf{s}} \text{enq}(y) \wedge$
 $\text{deq}(y) <_{\mathbf{s}} \text{deq}(x)$

Queue linearizability (axiomatic)

h is queue (without empty removals) linearizable iff

1. $\text{rem}(x) \in \mathbf{h} \Rightarrow \text{ins}(x) \in \mathbf{h} \wedge \text{rem}(x) \not<_{\mathbf{h}} \text{ins}(x)$
2. $\text{enq}(x) <_{\mathbf{h}} \text{enq}(y) \wedge \text{deq}(y) \in \mathbf{h} \Rightarrow \text{deq}(x) \in \mathbf{h} \wedge \text{deq}(y) \not<_{\mathbf{h}} \text{deq}(x)$

CV violations
= V violations

Pool

infinite
inductive
violations

Pool sequential specification (axiomatic)

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

$$1. \text{rem}(x) \in \mathbf{s} \Rightarrow \text{ins}(x) \in \mathbf{s} \wedge \text{ins}(x) <_{\mathbf{s}} \text{rem}(x)$$

$$2. \text{rem}(\perp) <_{\mathbf{s}} \text{rem}(x) \Rightarrow \text{rem}(\perp) <_{\mathbf{s}} \text{ins}(x) \wedge \text{ins}(x) <_{\mathbf{s}} \text{rem}(\perp) \Rightarrow \text{rem}(x) <_{\mathbf{s}} \text{rem}(\perp)$$

V violations
 $\text{rem}(x) <_{\mathbf{s}} \text{ins}(x)$

and

$\text{ins}(x) <_{\mathbf{s}} \text{rem}(\perp) <_{\mathbf{s}} \text{rem}(x)$

Pool linearizability (axiomatic)

h is pool (with empty removals) linearizable
iff

$$1. \text{rem}(x) \in \mathbf{h} \Rightarrow \text{ins}(x) \in \mathbf{h} \wedge \text{rem}(x) <_{\mathbf{h}} \text{ins}(x)$$

2.

infinitely many CV violations

$$\text{ins}(x_1) <_{\mathbf{h}} \text{rem}(\perp) \wedge \text{ins}(x_2) <_{\mathbf{h}} \text{rem}(x_1) \wedge \dots \wedge \text{ins}(x_{n+1}) <_{\mathbf{h}} \text{rem}(x_n) \wedge \text{rem}(\perp) <_{\mathbf{h}} \text{rem}(x_{n+1})$$

infinite
inductive
violations

Queue

Queue sequential specification (axiomatic)

s is a legal queue (with empty removals) sequence
iff

1. $\text{deq}(x) \in \mathbf{s} \Rightarrow \text{enq}(x) \in \mathbf{s} \wedge \text{enq}(x) <_{\mathbf{s}} \text{deq}(x)$
2. $\text{deq}(\perp) <_{\mathbf{s}} \text{deq}(x) \Rightarrow \text{deq}(\perp) <_{\mathbf{s}} \text{enq}(x) \wedge \text{enq}(x) <_{\mathbf{s}} \text{deq}(\perp) \Rightarrow \text{deq}(x) <_{\mathbf{s}} \text{deq}(\perp)$
3. $\text{enq}(x) <_{\mathbf{s}} \text{enq}(y) \wedge \text{deq}(y) \in \mathbf{s} \Rightarrow \text{deq}(x) \in \mathbf{s} \wedge \text{deq}(x) <_{\mathbf{s}} \text{deq}(y)$

Queue linearizability (axiomatic)

h is queue (with empty removals) linearizable
iff

1. $\text{deq}(x) \in \mathbf{h} \Rightarrow \text{enq}(x) \in \mathbf{h} \wedge \text{deq}(x) \not<_{\mathbf{h}} \text{enq}(x)$
2. infinitely many CV violations
 $\text{enq}(x_1) <_{\mathbf{h}} \text{deq}(\perp) \wedge \text{enq}(x_2) <_{\mathbf{h}} \text{deq}(x_1) \wedge \dots \wedge \text{enq}(x_{n+1}) <_{\mathbf{h}} \text{deq}(x_n) \wedge \text{deq}(\perp) <_{\mathbf{h}} \text{deq}(x_{n+1})$
3. $\text{enq}(x) <_{\mathbf{h}} \text{enq}(y) \wedge \text{deq}(y) \in \mathbf{h} \Rightarrow \text{deq}(x) \in \mathbf{h} \wedge \text{deq}(y) \not<_{\mathbf{h}} \text{deq}(x)$

V violations
 $\text{rem}(x) <_{\mathbf{s}} \text{ins}(x)$
and
 $\text{ins}(x) <_{\mathbf{s}} \text{rem}(\perp) <_{\mathbf{s}} \text{rem}(x)$
and
 $\text{enq}(x) <_{\mathbf{s}} \text{enq}(y) \wedge$
 $\text{deq}(y) <_{\mathbf{s}} \text{deq}(x)$

Concurrent Queues

Data independence => 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

$\text{deq} \Rightarrow v$ $\text{deq} \Rightarrow v$



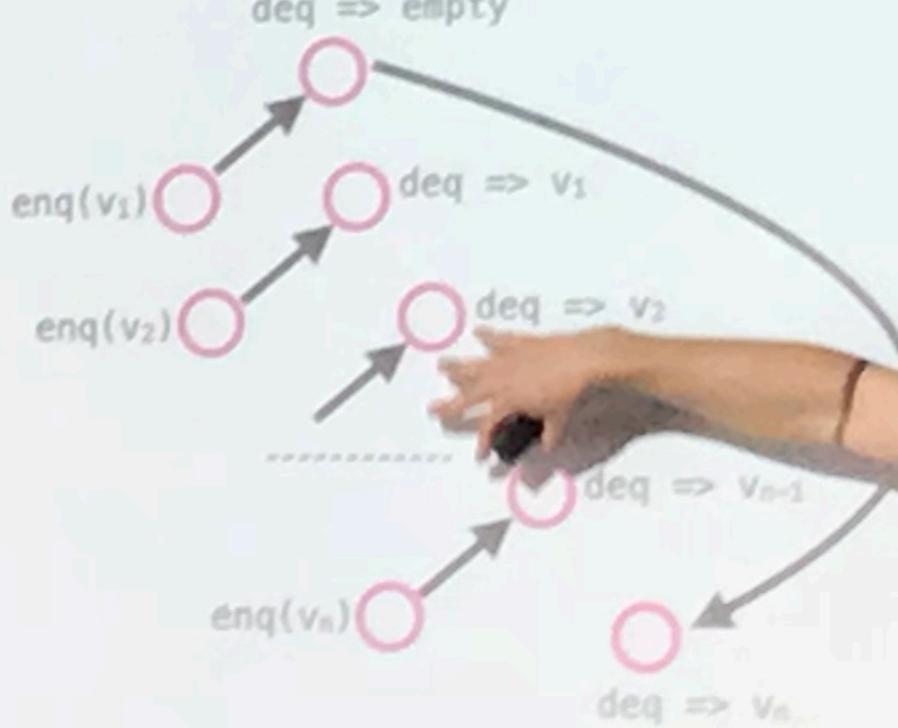
Value v_1 and v_2 dequeued in the wrong order

$\text{enq}(v_1)$ $\text{enq}(v_2)$ $\text{deq} \Rightarrow v_2$ $\text{deq} \Rightarrow v_1$



Dequeue wrongfully returns empty

$\text{deq} \Rightarrow \text{empty}$



It works for

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

infinite
inductive
violations

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