# Linearizability via Order Extension Theorems

a glimpse into
unpublished results and
some open problems

foundational results
for
verifying linearizability

# Inspiration

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

**Queue sequential specification (axiomatic)**

**s** is a legal queue sequence

iff

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

**Queue linearizability (axiomatic)**

Henzinger, Sezgin, Vafeiadis CONCUR13

**h** is queue linearizable

iff

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

precedence order

# Problems (stack)

**Stack sequential specification (axiomatic)**

**s** is a legal stack sequence

iff

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

**Stack linearizability (axiomatic)**

**h** is stack linearizable

iff

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

???

# Problems (stack)

**Stack sequential specification (axiomatic)**

**s** is a legal stack sequence

iff

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

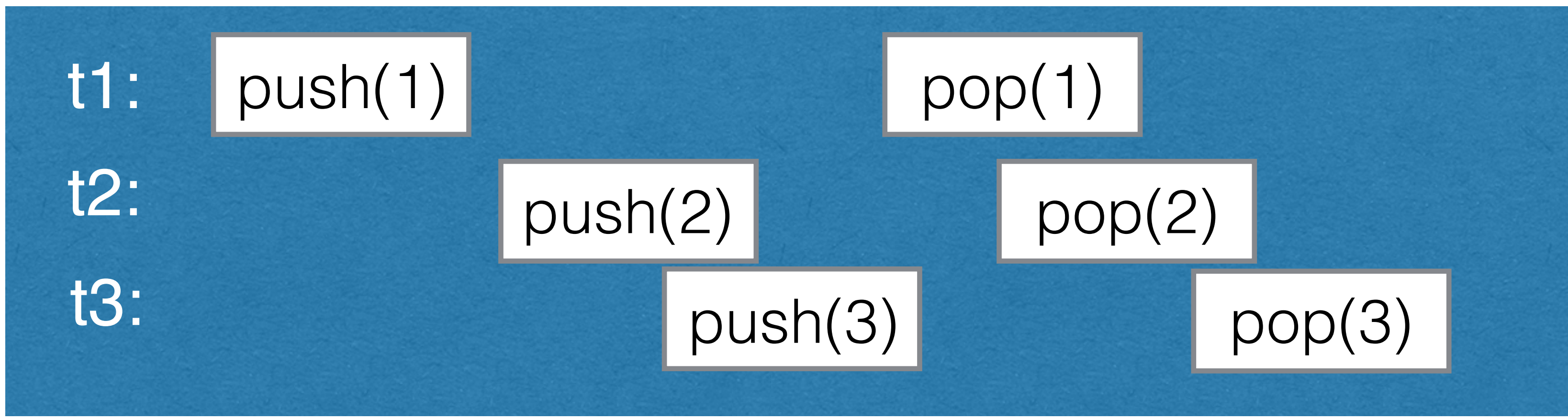
**Stack linearizability (axiomatic)**

**h** is stack linearizable

iff

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

# Problems (stack)




**not** stack linearizable

**Stack linearizability (axiomatic)**

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

~~iff~~

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

# Linearizability verification

**Data structure**

- signature $\Sigma$ - 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 $\Sigma$, such that $\mathbf{s} \in S$ iff $\mathbf{s}$ has no violations

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

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

**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.

concurrent history

legal sequence

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. rem(x) ∈ **s** $\Rightarrow$ ins(x) ∈ **s** $\wedge$ ins(x) $<_{\mathbf{s}}$ rem(x)

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

**Pool linearizability (axiomatic)**

**h** is pool (without empty removals) linearizable
iff
1. rem(x) ∈ **h** $\Rightarrow$ ins(x) ∈ **h** $\wedge$ rem(x) $\not<_{\mathbf{h}}$ 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. $deq(x) \in \mathbf{s} \Rightarrow enq(x) \in \mathbf{s} \wedge enq(x) <_{\mathbf{s}} deq(x)$
2. $enq(x) <_{\mathbf{s}} enq(y) \wedge deq(y) \in \mathbf{s} \Rightarrow deq(x) \in \mathbf{s} \wedge deq(x) <_{\mathbf{s}} deq(y)$

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

## Queue linearizability (axiomatic)

**h** is queue (without empty removals) linearizable
iff

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

CV violations
= V violations

# Pool

infinite inductive violations

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

**Pool sequential specification (axiomatic)**

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

1. rem(x) ∈ **s** ⇒ ins(x) ∈ **s** ∧ ins(x) $<_\mathbf{s}$ rem(x)
2. rem(⊥) $<_\mathbf{s}$ rem(x) ⇒ rem(⊥) $<_\mathbf{s}$ ins(x) ∧ ins(x) $<_\mathbf{s}$ rem(⊥) ⇒ rem(x) $<_\mathbf{s}$ rem(⊥)

**Pool linearizability (axiomatic)**

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

1. rem(x) ∈ **h** ⇒ ins(x) ∈ **h** ∧ rem(x) $\not<_\mathbf{h}$ ins(x)
2. … …

infinitely many CV violations
ins($x_1$) $<_\mathbf{h}$ rem(⊥) ∧ ins($x_2$) $<_\mathbf{h}$ rem($x_1$) ∧ … ∧ ins($x_{n+1}$) $<_\mathbf{h}$ rem($x_n$) ∧ rem(⊥) $<_\mathbf{h}$ rem($x_{n+1}$)

infinite inductive violations

# Queue

V violations
rem(x) $<_s$ ins(x)
and
ins(x) $<_s$ rem(⊥) $<_s$ rem(x)
and
enq(x) $<_s$ enq(y) ∧
deq(y) $<_s$ deq(x)

## Queue sequential specification (axiomatic)

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

iff

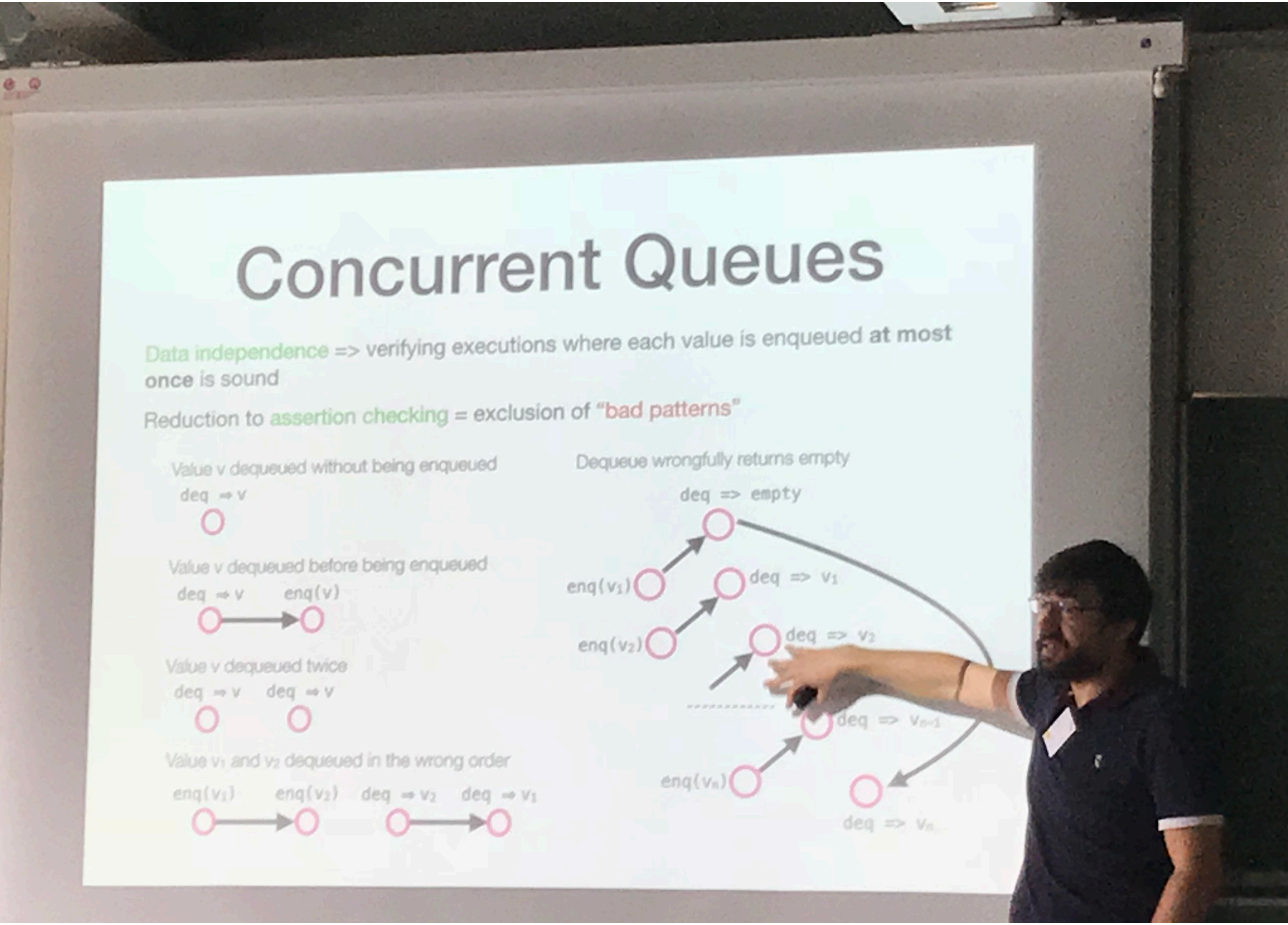
1. deq(x) ∈ **s** ⇒ enq(x) ∈ **s** ∧ enq(x) $<_s$ deq(x)
2. deq(⊥) $<_s$ deq(x) ⇒ deq(⊥) $<_s$ enq(x) ∧ enq(x) $<_s$ deq(⊥) ⇒ deq(x) $<_s$ deq(⊥)
3. enq(x) $<_s$ enq(y) ∧ deq(y) ∈ **s** ⇒ deq(x) ∈ **s** ∧ deq(x) $<_s$ deq(y)

## Queue linearizability (axiomatic)

**h** is queue (with empty removals) linearizable

iff

1. deq(x) ∈ **h** ⇒ enq(x) ∈ **h** ∧ deq(x) $\not<_h$ enq(x)
2. infinitely many CV violations
   enq($x_1$) $<_h$ deq(⊥) ∧ enq($x_2$) $<_h$ deq($x_1$) ∧ … ∧ enq($x_{n+1}$) $<_h$ deq($x_n$) ∧ deq(⊥) $<_h$ deq($x_{n+1}$)
3. enq(x) $<_h$ enq(y) ∧ deq(y) ∈ **h** ⇒ deq(x) ∈ **h** ∧ deq(y) $\not<_h$ 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
deq ⇒ v
Value v dequeued before being enqueued
deq ⇒ v
enq(v)
Value v dequeued twice
deq ⇒ v
deq ⇒ v
Value $v_1$ and $v_2$ dequeued in the wrong order
enq($v_1$)
enq($v_2$)
deq ⇒ $v_2$
deq ⇒ $v_1$
Dequeue wrongfully returns empty
deq => empty
enq($v_1$)
deq => $v_1$
enq($v_2$)
deq => $v_2$
deq => $v_{n-1}$
enq($v_n$)
deq => $v_n$

# 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