Relaxed Ordered Data Structures: Faster and Better

Andreas Haas
Thomas A. Henzinger
Christoph M. Kirsch
Michael Lippautz
Hannes Payer
Ali Sezgin
Ana Sokolova

University of Salzburg
IST Austria
University of Salzburg
University of Salzburg
University of Salzburg
IST Austria
University of Salzburg
Performance and scalability
Semantics of concurrent data structures

- Sequential specification – set of legal sequences
- Correctness condition – linearizability
Semantics of concurrent data structures

Stack - legal sequence

push(a) push(b) pop(b)

Sequential specification - set of legal sequences

Correctness condition - linearizability
Semantics of concurrent data structures

Stack - legal sequence
push(a)push(b)pop(b)

Sequential specification - set of legal sequences

Correctness condition - linearizability

Stack - concurrent history
begin-push(a)begin-push(b)end-push(a)end-push(b)begin-pop(b)end-pop(b)
Semantics of concurrent data structures

Sequential specification - set of legal sequences

Correctness condition - linearizability

Stack - legal sequence

push(a)push(b)pop(b)

Linearizable wrt seq.spec.

Stack - concurrent history

begin-push(a)begin-push(b)end-push(a)end-push(b)begin-pop(b)end-pop(b)
Semantics of concurrent data structures

- **Sequential specification** - set of legal sequences
  - Stack - legal sequence
    - `push(a) push(b) pop(b)`

- **Correctness condition** - linearizability
  - Stack - concurrent history
    - `begin-push(a) begin-push(b) end-push(a) end-push(b) begin-pop(b) end-pop(b)`

- We relax this

- Linearizable wrt seq.spec.
Relaxations (POPL, Thursday)

- May trade correctness for performance
- In a controlled way with quantitative bounds

measure the error from correct behavior
Relaxations (POPL, Thursday)

Stack - incorrect behavior

push(a)push(b)push(c)pop(a)pop(b)

- May trade correctness for performance
- In a controlled way with quantitative bounds

Correct in a relaxed stack

... 2-relaxed.. 3-relaxed

measure the error from correct behavior
Why relax?

- It is interesting
- Provides potential for better performing concurrent implementations

Stack

- top
- thread 1
- thread 2
- ... thread n

k-Relaxed stack

- top
- thread 1
- thread 2
- ... thread n

Ana Sokolova University of Salzburg

Concurrency Yak 21.1.2013
What we have (POPL)

- Framework
- Generic examples
- Concrete relaxation examples
- Efficient concurrent implementations
The big picture, briefly

\[ S \subseteq \Sigma^* \]

\( \Sigma \) - methods with arguments

semantics
sequential specification
legal sequences
The big picture, briefly

$S_k \subseteq \Sigma^*$

$S \subseteq \Sigma^*$

$\Sigma$ - methods with arguments

semantics
sequential specification
legal sequences
relaxed semantics
The big picture, briefly

\[ S_k \subseteq \Sigma^* \]

\[ S \subseteq \Sigma^* \]

- methods with arguments

\textit{distance!}

semantics
sequential specification
legal sequences

relaxed semantics

Ana Sokolova University of Salzburg

Concurrency Yak 21.1.2013
Out-of-order relaxation

... is a natural concrete one

Stack

Each \texttt{pop} pops one of the \((k+1)\)-youngest elements
Out-of-order relaxation

... is a natural concrete one

Stack

Each \textbf{pop} pops one of the \((k+1)\)-youngest elements

Queue

Each \textbf{deq} deques one of the \((k+1)\)-youngest elements
Out-of-order relaxation

... is a natural concrete one

Stack

Each **pop** pops one of the \((k+1)\)-youngest elements

Queue

Each **deq** deques one of the \((k+1)\)-youngest elements

**k-out-of-order relaxation**
Out-of-order relaxation

... is a natural concrete one

Stack
Each `pop` pops one of the \((k+1)\)-youngest elements

Queue
Each `deq` deques one of the \((k+1)\)-youngest elements

What is the distance?
Syntactic distances do not help

\[
\text{push}(a) \ [\text{push}(i)\text{pop}(i)]^n\text{push}(b) \ [\text{push}(j)\text{pop}(j)]^m\text{pop}(a)
\]

is a 1-out-of-order stack sequence

**Spoiler --- more about it on Thursday!**

its permutation distance is \( \min(n,m) \)
Framework for semantic distances (POPL)

- Identify states, build LTS(S)
- Add incorrect transitions with transition costs
- Fix a path cost function
Framework for semantic distances (POPL)

- Identify states, build LTS(S)
  - Add incorrect transitions with transition costs
  - Fix a path cost function
Framework for semantic distances (POPL)

- Identify states, build LTS(S)
- Add incorrect transitions with transition costs
- Fix a path cost function

- doable in a generic way !!!
  (also for out-of-order)
Out-of-order stack

- Canonical representative of a state
- Add incorrect transitions with costs
- Possible path cost functions $\text{max}$, $\text{sum}$,...
Out-of-order queue

- Canonical representative of a state
- Add incorrect transitions with costs
- Possible path cost functions max, sum,...

Sequence of enq's with no matching deq
How useful are these relaxations? Performance?
Lessons learned

The way from sequential specification to concurrent implementation is hard

Being relaxed not necessarily means better performance

Well-performing implementations of relaxed specifications do exist!
Our current interests

- Study applicability
- Learn from efficient implementations
Our current interests

- Study applicability
  - which applications tolerate relaxation?
  - maybe there is nothing to tolerate!
- Learn from efficient implementations
Our current interests

- Study applicability
- Learn from efficient implementations

- which applications tolerate relaxation?
- maybe there is nothing to tolerate!
- towards synthesis
- lock-free universal construction?
Observed non-determinism

Input sequence: enq(a)enq(b)enq(c)deq(x)
Observed non-determinism

Input sequence: enq(a) enq(b) enq(c) deq(x)

Ana Sokolova University of Salzburg
Observed non-determinism

Input sequence: enq(a)enq(b)enq(c)deq(x)

ideal, $x = a$

slow and strict $x \in \{a, b, c\}$
Observed non-determinism

Input sequence: enq(a)enq(b)enq(c)deq(x)
Observed non-determinism

Two reasons

- Relaxation (the more relaxed, the more...)
- Linearizability (the slower, the more...)
Observed non-determinism

Two reasons

- Relaxation (the more relaxed, the more...)
- Linearizability (the slower, the more...)

Connection between relaxation and performance
Observed non-determinism

Two reasons

- **Relaxation** (the more relaxed, the more...)
- **Linearizability** (the slower, the more...)

Connection between relaxation and performance

What is it really? Measure for performance?
Relaxation vs. performance

Fixed input sequence $w$

$R : \mathbb{N} \rightarrow \mathbb{N}$

$R(n) = \min k \text{ s.t. a linearization of a concurrent history with input } w$

and performance index $n$ is in $S_k$

$P : \mathbb{N} \rightarrow \mathbb{N}$

$P(k) = \min n \text{ s.t. a linearization of a concurrent history with input } w$

and performance index $n$ is in $S_k$

Performance index (of a concurrent history) = number of overlaps
R vs. P graph

Fixed input sequence \( w \)

\[
\{(n, R(n)) \mid n \in \mathbb{N}\} \cup \{(P(k), k) \mid k \in \mathbb{N}\}
\]
R vs. P graph

Fixed input sequence w

\{(n, R(n)) \mid n \in \mathbb{N}\} \cup \{(P(k), k) \mid k \in \mathbb{N}\}

w's relaxation

Ana Sokolova  University of Salzburg

Concurrency Yak  21.1.2013
\[(n, R(n)) \mid n \in \mathbb{N}\} \cup \{(P(k), k) \mid k \in \mathbb{N}\}\]

Fixed input sequence \(w\)

\(w\) can be generated by an implementation with relaxation \(\pi_1(\star)\) and performance index \(\pi_2(\star)\).
One of P or R is sufficient for the P vs. R graph

\[ R(n) = \min \{k \mid P(k) \leq n\} \]

\[ P(k) = \min \{n \mid R(n) \leq k\} \]
Back to measuring observed non-determinism
Implementations around...

- SCAL queues [KPRS’11]
- Quasi linearizability (SQ, RDQ) theory and implementations [AKY’10]
- Some straightforward implementations [HKPSS’12]
- Efficient lock-free segment queue k-FIFO [KLP’12]
- Efficient lock-free segment stack k-Stack [POPL]
- Efficient distributed queues DQ (relatives to SCAL)
Back to measuring observed non-determinism

Actual-time sequence:  \text{enq}(b)\text{enq}(c)\text{enq}(a)\text{deq}(b)

Zero-time sequence:  \text{enq}(a)\text{enq}(b)\text{enq}(c)\text{deq}(b)
Back to measuring observed non-determinism

Actual-time sequence: enq(b)enq(c)enq(a)deq(b)
Zero-time sequence: enq(a)enq(b)enq(c)deq(b)

one possibility - strict
Back to measuring observed non-determinism

Actual-time sequence: \text{enq}(b)\text{enq}(c)\text{enq}(a)\text{deq}(b)

Zero-time sequence: \text{enq}(a)\text{enq}(b)\text{enq}(c)\text{deq}(b)

Ana Sokolova University of Salzburg
The experiments look good

- Relaxed efficient implementations perform/scale well (also better than pools)
  DQs are the best

- Performance index is a reasonable indicator of performance

- All show comparable observed non-determinism (also strict implementations)
The experiments look good

- Relaxed efficient implementations perform/scale well (also better than pools)
  DQs are the best

- Performance index is a reasonable indicator of performance

- All show comparable observed non-determinism (also strict implementations)

Any real applications that use concurrent queues / stacks?
The experiments look good

- Relaxed efficient implementations perform/scale well (also better than pools)
  - DQs are the best
- Performance index is a reasonable indicator of performance
- All show comparable observed non-determinism (also strict implementations)

Any real applications that use concurrent queues / stacks?