Quantitatively Relaxed Data Structures

Thomas A. Henzinger
Christoph M. Kirsch
Hannes Payer
Ali Sezgin
Ana Sokolova

IST Austria
University of Salzburg
University of Salzburg
IST Austria
University of Salzburg

RISE 14.6.2012
The goal

- Trading correctness for performance
- In a controlled way with quantitative bounds
The goal

- Trading correctness for performance
- In a controlled way with quantitative bounds

measure the error from correct behavior
The goal

Trading 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

Stack - incorrect behavior

push(a)push(b)push(c)pop(a)pop(b)
Stack example

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

state evolution
Stack example

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

state evolution

top

a
Stack example

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

state evolution

top

b

a
Stack example

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

state evolution

top
↓
c
  b
  a
Stack example

\[
push(a) \quad push(b) \quad push(c) \quad pop(a) \quad pop(b)
\]

state evolution

top

???

???

???

Ana Sokolova  University of Salzburg
Stack example

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

How much does this error cost?

state evolution
Stack example

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

state evolution

top

\[ \text{Cost 2} \]

Ana Sokolova  University of Salzburg
Stack example

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

state evolution

???

Cost 2
Stack example

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

state evolution

top

c

b

Cost 1

a

Cost 2
Stack example

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

state evolution

top

c

Total cost?

Cost 1

Cost 2

Ana Sokolova University of Salzburg

RISE 14.6.2012
Stack example

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

state evolution

top

Total cost?

c

b

Cost 1

a

Cost 2

max = 2

sum = 3

Ana Sokolova  University of Salzburg

RiSE 14.6.2012
Why relax?

- It is theoretically interesting
- Provides potential for better performing concurrent implementations
Why relax?

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

Stack

k-Relaxed stack

Ana Sokolova University of Salzburg
What we have

- Framework
- Generic example
- Concrete relaxation examples
- Efficient concurrent implementations

for semantic relaxations
for ordered data structures
stacks, queues, priority queues,..
of relaxation instances

Ana Sokolova  University of Salzburg
Enough introduction
The big picture

$S \subseteq \Sigma^*$

$\Sigma$ - methods with arguments

semantics
sequential specification
legal sequences

Ana Sokolova  University of Salzburg
The big picture

$\Sigma \subseteq \Sigma^*$

$S_k \subseteq \Sigma^*$

$S \subseteq \Sigma^*$

$\Sigma$ - methods with arguments

semantics
sequential specification
legal sequences
relaxed semantics

Ana Sokolova  University of Salzburg
The big picture

\[ S_k \subseteq \Sigma^* \]

\[ S \subseteq \Sigma^* \]

- semantics
- sequential specification
- legal sequences
- relaxed semantics

\( \Sigma \) - methods with arguments

Ana Sokolova  University of Salzburg
The big picture

$S_k \subseteq \Sigma^*$

$S \subseteq \Sigma^*$

(semantics)

sequential specification

legal sequences

relaxed semantics

leads to relaxed linearizability

$\Sigma$ - methods with arguments

Ana Sokolova  University of Salzburg

RiSE 14.6.2012
Theoretical challenge

There are natural concrete relaxations...

Stack

Each **pop** pops one of the k-youngest elements
Each **push** pushes .....
Theoretical challenge

There are natural concrete relaxations...

Stack

Each **pop** pops one of the k-youngest elements
Each **push** pushes ..... k-out-of-order relaxation

Ana Sokolova  University of Salzburg
Theoretical challenge

There are natural concrete relaxations...

Stack

Each **pop** pops one of the $k$-youngest elements

Each **push** pushes .....  

makes sense also for queues, priority queues, .....
Theoretical challenge

There are natural concrete relaxations...

Stack

Each **pop** pops one of the k-youngest elements

Each **push** pushes ..... 

makes sense also for queues, priority queues, ....

k-out-of-order relaxation

How is it reflected by a distance between sequences?

one distance for all?
Syntactic distances do not help

push(a) [push(i) pop(i)]^npush(b) [push(j) pop(j)]^m pop(a)
Syntactic distances do not help

\[
push(a)[push(i)pop(i)]^npush(b)[push(j)pop(j)]^mpop(a)\]

is a 1-out-of-order stack sequence
Syntactic distances do not help

\[
push(a)[push(i)pop(i)]^npush(b)[push(j)pop(j)]^mpop(a)
\]

is a 1-out-of-order stack sequence

its permutation distance is unbounded

Ana Sokolova University of Salzburg
Semantic distances need a notion of state

- States are equivalence classes of sequences in $S$

- Two sequences in $S$ are equivalent if they have an indistinguishable future
Semantic distances need a notion of state

States are equivalence classes of sequences in $S$

- example: for stack
  \[
  \text{push(a)push(b)pop(b)push(c)} \equiv \text{push(a)push(c)}
  \]

- Two sequences in $S$ are equivalent if they have an indistinguishable future
Semantic distances need a notion of state

States are equivalence classes of sequences in $S$

Example: for stack

$\text{push}(a)\text{push}(b)\text{pop}(b)\text{push}(c) \equiv \text{push}(a)\text{push}(c)$

Two sequences in $S$ are equivalent if they have an indistinguishable future
Semantic distances need a notion of state

States are equivalence classes of sequences in $S$

Two sequences in $S$ are equivalent if they have an indistinguishable future

$$x \equiv y \iff \forall u \in \Sigma^*. (xu \in S \iff yu \in S)$$

example: for stack

$\text{push}(a) \text{push}(b) \text{pop}(b) \text{push}(c) \equiv \text{push}(a) \text{push}(c)$

Ana Sokolova  University of Salzburg
Semantics goes operational

\[ S \subseteq \Sigma^* \text{ is the sequential specification} \]

\[ \text{LTS}(S) = (S/\equiv, \Sigma, \rightarrow, [\varepsilon]_{\equiv}) \text{ with} \]

\[ [s]_{\equiv} \xrightarrow{m} [sm]_{\equiv} \iff sm \in S \]
Semantics goes operational

\[ S \subseteq \Sigma^* \text{ is the sequential specification} \]

\[ \text{LTS}(S) = (S/\equiv, \Sigma, \rightarrow, [\varepsilon]_\equiv) \text{ with} \]

- states
- labels
- initial state

transition relation

\[ [s]_\equiv \xrightarrow{\text{transition relation}} [sm]_\equiv \iff \text{sm} \in S \]
The framework

- Completion of LTS(S)
- Transition costs
- Path cost function
The framework

- Completion of LTS(S)
- Transition costs
- Path cost function

Σ - singleton

Ana Sokolova  University of Salzburg

RiSE 14.6.2012
The framework

- Completion of LTS(S)
- Transition costs
- Path cost function
The framework

- Completion of LTS(S)
- Transition costs
- Path cost function
The framework

- Completion of LTS(S)
- Transition costs
- Path cost function
The framework

- Completion of LTS(S)
- Transition costs
- Path cost function

(distance - minimal cost on all paths labelled by the sequence)
For the user

- Pick your favorite data structure S
- Add desired incorrect transitions and assign them transition costs
- Choose a path cost function
For the user

- Pick your favorite data structure $S$
- Add desired incorrect transitions and assign them transition costs
- Choose a path cost function

The framework clears the head, direct concrete relaxations are also possible.

distance and relaxation follow
Stack example

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

state evolution

top

c

Total cost

\begin{align*}
\text{Cost 1} & : \text{b} \\
\text{Cost 2} & : \text{a}
\end{align*}

\begin{align*}
\text{max} & = 2 \\
\text{sum} & = 3
\end{align*}
Stack example

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

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

- Canonical representative of a state
- Add incorrect transitions with costs
- Possible path cost functions $\text{max, sum, ...}$
Let's generalize
Generic out-of-order

\[ \text{segment\_cost}( q \xrightarrow{m} q') = |v| \]

where \( v \) is a sequence of minimal length s.t.

\[ [uvw] = q, \text{minimal}, \quad uw \text{ minimal} \]

\[ (1) \quad [uvw] \equiv q' \]

\[ (1.1) \quad [uw] \equiv [u'w], \quad [u'vw] = q' \]

\[ (1.2) \quad [uw] \equiv [uw'], \quad [uvw'] = q' \]

\[ (2) \quad [uw] = q, \text{minimal}, \quad uvw \text{ minimal} \]

\[ (1.1) \quad [uvw] = [u'vw], \quad [uvw'] = q' \]

\[ (1.2) \quad [uvw] = [uvw'], \quad [uw] = q' \]

transition cost

removing \( v \) enables a transition

inserting \( v \) enables a transition

 goes with different path costs
Generic out-of-order

\[ \text{segment\_cost}(q \xrightarrow{m} q') = |v| \]

where \( v \) is a sequence of minimal length s.t.

\[ (1) \quad [uvw] = q, \quad uvw \text{ is minimal, } uw \text{ is minimal} \]
\[ (1.1) \quad [uw] \xrightarrow{m} [uw'] = q' \]
\[ (1.2) \quad [uw] \xrightarrow{m} [uvw'] = q' \]

\[ (2) \quad [uw] = q, \quad uw \text{ is minimal, } uvw \text{ is minimal} \]
\[ (1.1) \quad [uvw] \xrightarrow{m} [u'vw] = q' \]
\[ (1.2) \quad [uvw] \xrightarrow{m} [uvw'] = q' \]
Generic out-of-order

\[
\text{segment\_cost}( q \xrightarrow{m} q' ) = |v| 
\]

where \( v \) is a sequence of minimal length s.t.

\[ (1) \]
\[
[uvw] = q', \; uvw \text{ is minimal, } uw \text{ is minimal}
\]

\[
(1.1) \quad [uw] \xrightarrow{m} [uw'] = q', \; [uvw'] = q'
\]

\[
(1.2) \quad [uvw] \xrightarrow{m} [uvw'] = q', \; [uw'] = q'
\]

\[ (2) \]
\[
[uw] = q', \; uw \text{ is minimal, } uvw \text{ is minimal}
\]

\[
(1.1) \quad [uvw] \xrightarrow{m} [uvw'] = q', \; [u']w = q'
\]

\[
(1.2) \quad [uvw] \xrightarrow{m} [uvw'] = q', \; [uw'] = q'
\]

transition cost

removing \( v \) enables a transition

inserting \( v \) enables a transition

Ana Sokolova University of Salzburg

RiSE 14.6.2012
Out-of-order stack

- Canonical representative of a state
- Add incorrect transitions with segment-costs

- Possible path cost functions $\text{max, sum,...}$
Out-of-order stack

Sequence of push's with no matching pop

- Canonical representative of a state
- Add incorrect transitions with segment-costs

- Possible path cost functions max, sum,...
Out-of-order stack

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

Sequence of push's with no matching pop
Out-of-order stack

- Sequence of push's with no matching pop
- Canonical representative of a state
- Add incorrect transitions with segment-costs
- Possible path cost functions max, sum,...

Also "shrinking window" restricted out-of-order

Ana Sokolova  University of Salzburg
RiSE 14.6.2012
Out-of-order queue

- Canonical representative of a state
- Add incorrect transitions with segment-costs

- Possible path cost functions \textit{max, sum,...}
Out-of-order queue

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

Ana Sokolova  University of Salzburg  RiSE 14.6.2012
Out-of-order queue

Sequence of \texttt{enq}'s with no matching \texttt{deq}

\begin{itemize}
  \item Canonical representative of a state
  \item Add incorrect transitions with segment-costs
  \item Possible path cost functions \texttt{max}, \texttt{sum}, ...
\end{itemize}
Out-of-order queue

- Canonical representative of a state
- Add incorrect transitions with segment-costs
- Possible path cost functions \( \text{max, sum,...} \)

Also "shrinking window" restricted out-of-order
Out-of-order variants

Queue

head

a b c d e ...

tail

z
Out-of-order variants

Queue

out-of-order k=3

restricted
out-of-order k=3

head

\[ a \quad b \quad c \quad d \quad e \quad \ldots \quad z \]

tail

lateness k=3
How about implementations? Performance?
Short-term history

- SCAL queues [KPRS’11]
- Quasi linearizability theory and implementations [AKY’10]
- Some straightforward implementations [HKPSS’12]
- Efficient lock-free segment queue [KLP’12]

(almost) all implement restricted out-of-order
Short-term history

- SCAL queues [KPRS’11]
- Quasi linearizability theory and implementations [AKY’10]
- Some straightforward implementations [HKPSS’12]
- Efficient lock-free segment queue [KLP’12]

(almost) all implement restricted out-of-order
Short-term history

- SCAL queues [KPRS’11]
- Quasi linearizability theory and implementations [AKY’10]
- Some straightforward implementations [HKPSS’12]
- Efficient lock-free segment queue [KLP’12]

Distributed, one k-queue

Syntactic, does not work for stacks

(almost) all implement restricted out-of-order
Short-term history

- SCAL queues [KPRS’11]
- Quasi linearizability theory and implementations [AKY’10]
- Some straightforward implementations [HKPSS’12]
- Efficient lock-free segment queue [KLP’12]

- Distributed, one k-queue
- Syntactic, does not work for stacks
- Not too well performing

(almost) all implement restricted out-of-order
Short-term history

- SCAL queues [KPRS’11]
- Quasi linearizability theory and implementations [AKY’10]
- Some straightforward implementations [HKPSS’12]
- Efficient lock-free segment queue [KLP’12]

- distributed, one k-queue
- syntactic, does not work for stacks
- not too well performing
- not too well performing
- (almost) all implement restricted out-of-order
Short-term history

- SCAL queues [KPRS’11]
- Quasi linearizability theory and implementations [AKY’10]
- Some straightforward implementations [HKPSS’12]
- Efficient lock-free segment queue [KLP’12]

- Distributed, one k-queue
- Syntactic, does not work for stacks
- Not too well performing
- Not too well performing
- Performs very well
- (almost) all implement restricted out-of-order
Lessons learned
Lessons learned

The way from sequential specification to concurrent implementation is hard
Lessons learned

The way from sequential specification to concurrent implementation is hard

Being relaxed not necessarily means better 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!
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!

Let’s see them!
Queue

Scalability comparison

Operations/ms (more is better)

Number of threads

Lock-based FIFO
Michael-Scott lock-free FIFO
Flat-Combining lock-based FIFO
Lock-free k-FIFO (k=64)
Queue

Scalability comparison

80-core machine

Operations/ms (more is better)

Number of threads

Lock-based FIFO
Michael-Scott lock-free FIFO
Flat-Combining lock-based FIFO
Lock-free k-FIFO (k=64)
Queue

The more relaxed, the better

lock-free segment queue
Stack

Scalability comparison

![Graph showing scalability comparison of lock-based stack, lock-free stack, and k-stack (k=64). The x-axis represents the number of threads, and the y-axis represents operations per ms (more is better). The data points show that lock-based stack performs the best, followed by lock-free stack, and then k-stack (k=64).]
Stack

Scalability comparison

Operations/ms (more is better)

Number of threads

Lock-based stack

Lock-free stack

k-stack (k=64)

80-core machine

Ana Sokolova  University of Salzburg
Stack

The more relaxed, the better

lock-free segment stack

![Graph showing the performance of stack operations with different thread counts and k values on a log scale. The x-axis represents the number of operations per second (more is better), and the y-axis represents the number of operations per second (more is better). The graph shows how the performance varies with different thread counts and k values.](image)
Contributions

Framework for quantitative relaxations
generic relaxation, concrete examples,
efficient implementations exist
Contributions

Framework for quantitative relaxations
- generic relaxation, concrete examples,
- efficient implementations exist

- all kinds of
Final remarks

Contributions
Framework for quantitative relaxations
generic relaxation, concrete examples,
efficient implementations exist

Difficult open problem
From practice to theory it works...
How to get from theory to practice?

all kinds of
Contributions

Framework for quantitative relaxations
generic relaxation, concrete examples,
efficient implementations exist

Difficult open problem

From practice to theory it works...
How to get from theory to practice?

all kinds of

THANK YOU