Quantitatively Relaxed Concurrent 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
Semantics of concurrent data structures

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

- Stack - legal sequence
  \[ \text{push}(a) \text{push}(b) \text{pop}(b) \]

- 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

**Stack - legal sequence**

- $\text{push}(a)\text{push}(b)\text{pop}(b)$

**Sequential specification** - set of legal sequences

**Correctness condition** - linearizability

**Stack - concurrent history**

- $\text{begin-push}(a)\text{begin-push}(b)\text{end-push}(a)\text{end-push}(b)\text{begin-pop}(b)\text{end-pop}(b)$

linearizable wrt seq.spec.
Semantics of concurrent data structures

Sequential specification - set of legal sequences

Correctness condition - linearizability

Stack - legal sequence

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

Stack - concurrent history

begin-push(a)begin-push(b)end-push(a)end-push(b)begin-pop(b)end-pop(b)
Performance and scalability

Throughput vs. # threads/cores graph.
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

Stack - incorrect behavior

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

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
What we have

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

...for semantic relaxations
...out-of-order / stuttering
...stacks, queues, priority queues,.. / CAS, shared counter
...of relaxation instances
The big picture

\( S \subseteq \Sigma^* \)

- methods with arguments

semantics
sequential specification
legal sequences
The big picture

\[ S_k \subseteq \Sigma^* \]

\[ S \subseteq \Sigma^* \]

\( \Sigma \) - methods with arguments

semantics

sequential specification

legal sequences

relaxed semantics

Ana Sokolova  University of Salzburg

Dagstuhl RiSE 16.11.2012
The big picture

\[ \Sigma k \subseteq \Sigma^* \]

- methods with arguments

- semantics
- sequential specification
- legal sequences
- relaxed semantics

\[ \text{distance?} \]
Challenge

There are natural concrete relaxations...

Stack

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

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

**k-out-of-order relaxation**
Challenge

There are natural concrete relaxations...

Stack

Each **pop** pops one of the (k+1)-youngest elements
Each **push** pushes .....  

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

How is it reflected by a distance between sequences?

one distance for all?
Syntactic distances do not help

\[\text{push}(a) [\text{push}(i) \text{pop}(i)]^n \text{push}(b) [\text{push}(j) \text{pop}(j)]^m \text{pop}(a)\]
Syntactic distances do not help

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

is a \text{1-out-of-order} stack sequence
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

its permutation distance is \( \min(n,m) \)
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 $S$

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

example: for stack

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

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

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

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

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

$S \subseteq \Sigma^*$ is the sequential specification

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

- states
- labels
- initial state

transition relation

$[s]_{\equiv} \xrightarrow{m} [sm]_{\equiv} \iff sm \in S$

Stack
top
$a$
push(c)
c
$a$

d transitions are 1-D cells
The framework

- Start from LTS(S)
- Add transitions with transition costs
- Fix a path cost function
The framework

- Start from $\text{LTS}(S)$
- Add transitions with transition costs
- Fix a path cost function
The framework

- Start from LTS(S)
- Add transitions with transition costs
- Fix a path cost function
The framework

- Start from LTS(S)
- Add transitions with transition costs
- Fix a path cost function
The framework

- Start from LTS(S)

- Add transitions with transition costs

- Fix a path cost function

**distance** - minimal cost on all paths labelled by the sequence
Generic out-of-order

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

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

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

removing \( v \) enables a transition

(1.1) 
\[
[uvw] = q \quad \Rightarrow \quad [uvw'] = q'
\]

(1.2) 
\[
[uw] = m \quad \Rightarrow \quad [uw'] = q'
\]

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

inserting \( v \) enables a transition

(1.1) 
\[
[uvw] = q \quad \Rightarrow \quad [uvw'] = q'
\]

(1.2) 
\[
[uw] = m \quad \Rightarrow \quad [uw'] = q'
\]

goes with different path costs

Ana Sokolova  University of Salzburg

Dagstuhl RiSE 16.11.2012
Out-of-order stack

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

Sequence of \textbf{push}'s with no matching \textbf{pop}

Also more advanced
Out-of-order queue

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

Sequence of enq's with no matching deq

Also more advanced
How about implementations? 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!
Stack

Scalability comparison

"80"-core machine

lock-free segment stack
k-Stack

The more relaxed, the better

lock-free segment stack

operations/ms (more is better)
Conclusions

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

Difficult open problem
How to get from theory to practice?

THANK YOU
For the future

- Study applicability
- Learn from efficient implementations
For the future

- Study applicability
- Learn from efficient implementations

which applications tolerate relaxation?

maybe there is nothing to tolerate!
For the future

- Study applicability
  - which applications tolerate relaxation?
  - maybe there is nothing to tolerate!

- Learn from efficient implementations
  - towards synthesis
  - lock-free universal construction?
For the future

- Study applicability
  - which applications tolerate relaxation?
  - maybe there is nothing to tolerate!

- Learn from efficient implementations
  - towards synthesis
  - lock-free universal construction?

THANK YOU

Ana Sokolova  University of Salzburg  Dagstuhl RiSE 16.11.2012