Concurrent Data Structures

Semantics and Quantitative Relaxations

Mike Dodds
Andreas Haas
Tom Henzinger
Andreas Holzer
Christoph Kirsch
Michael Lippautz
Hannes Payer
Ali Sezgin
Ana Sokolova

University of York
University of Salzburg
IST Austria
TU Vienna
University of Salzburg
University of Salzburg
University of Salzburg
IST Austria
University of Salzburg

Universitaet des Saarlandes 21.11.2014
Semantics of sequential data structures

Sequential specification - set of legal sequences

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

Examples of sequential data structures:
- Pools
- Queues
- Stacks
Semantics of concurrent data structures

- Sequential specification - set of legal sequences
- Consistency condition - e.g. linearizability

Stack - concurrent history
begin-push(a) begin-push(b) end-push(a) end-push(b) begin-pop(b) end-pop(b)

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

linearizable wrt seq.spec.
There exists a sequential witness that preserves precedence across quiesce.state

There exists a sequential witness that preserves per-thread precedence

linearizability

sequential consistency

quiescent consistency

There exists a sequential witness that preserves precedence
Performance and scalability

throughput

# threads/cores
Relaxations allow

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
- c
- b
- a

k-Relaxed stack

- top
- thread 1
- thread 2
- thread n
- k
- c
- b
- a
Relaxations of concurrent data structures

- Sequential specification - set of legal sequences
- Consistency condition - e.g. linearizability

- Quantitative relaxations
  Henzinger, Kirsch, Payer, Sezgin, S.
  POPL 2013

- (Quantitative) relaxations
  Dodds, Sezgin, S.
  work in progress
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

$S \subseteq \Sigma^*$

$\Sigma$ - methods with arguments

semantics
sequential specification
legal sequences

Quantitative relaxations (sequential specification)
The big picture

$S_k \subseteq \Sigma^*$

$s \subseteq \Sigma^*$

semantics
sequential specification
legal sequences

relaxed semantics

$\Sigma$ - methods with arguments

Quantitative relaxations (sequential specification)

Ana Sokolova  University of Salzburg

Universitaet des Saarlandes 21.11.2014
Quantitative relaxations (sequential specification)

The big picture

\[ S_k \subseteq \Sigma^* \]

Semantics
Sequential specification
Legal sequences

\[ S \subseteq \Sigma^* \]

Relaxed semantics

\[ \Sigma \] - methods with arguments

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, .... 

k-out-of-order relaxation

How is it reflected by a distance between sequences?

one distance for all?
Syntactic distances do not help

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

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.

$\equiv$ example: for stack

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

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

$m$

$[s]_\equiv \rightarrow [sm]_\equiv \iff sm \in S$

Ana Sokolova  
University of Salzburg

Quantitative relaxations (sequential specification)

Universitaet des Saarlandes 21.11.2014
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

\( \Sigma \) – singleton
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
Out-of-order stack

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

Quantitative relaxations (sequential specification)
Out-of-order queue

Sequence of enq’s with no matching deq

Canonical representative of a state

Add incorrect transitions with segment-costs

Possible path cost functions max, sum,...

Also more advanced

Ana Sokolova  University of Salzburg

Universitaet des Saarlandes 21.11.2014
Implementations and Performance
Relaxed implementations

k-Stack
Henzinger, Kirsch, Payer, Sezgin, S.
POPL 2013

Distributed queues / stacks
Haas, Henzinger, Kirsch, Lippautz, Payer, Sezgin, S.
CF 2013
k-Stack

Performance and Scalability comparison

"80"-core machine

lock-free segment stack

Ana Sokolova University of Salzburg

Universitaet des Saarlandes 21.11.2014
Distributed queues

Performance and Scalability comparison

"80"-core machine

operations per ms (more is better)

number of threads

LB
MS
FC
WF
RD
SQ
BS k-FIFO (k=80)
US k-FIFO (k=80)
ED
BAG
RP
1-RR DQ (p=80)
2-RR DQ (p=80)
TL-RR DQ (p=80)
1-RA DQ (p=80)
2-RA DQ (p=80)
LRU DQ (p=80)
Bad performance also relaxes semantics

Linearizability revisited

The slower the implementation, the more nondeterminism

Semantics vs. performance comparison (Con²Colic testing)
Haas, Henzinger, Holzer, Kirsch, … S. work in progress