## Concurrent Data Structures

### Semantics and Quantitative Relaxations

<table>
<thead>
<tr>
<th>Name</th>
<th>University</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mike Dodds</td>
<td>University of York</td>
</tr>
<tr>
<td>Andreas Haas</td>
<td>University of Salzburg</td>
</tr>
<tr>
<td>Tom Henzinger</td>
<td>IST Austria</td>
</tr>
<tr>
<td>Andreas Holzer</td>
<td>TU Vienna</td>
</tr>
<tr>
<td>Christoph Kirsch</td>
<td>University of Salzburg</td>
</tr>
<tr>
<td>Michael Lippautz</td>
<td>University of Salzburg</td>
</tr>
<tr>
<td>Hannes Payer</td>
<td>University of Salzburg</td>
</tr>
<tr>
<td>Ali Sezgin</td>
<td>IST Austria</td>
</tr>
<tr>
<td>Ana Sokolova</td>
<td>University of Salzburg</td>
</tr>
</tbody>
</table>
Semantics of sequential data structures

- **Sequential specification** - set of legal sequences

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

There exists a sequential witness that preserves precedence.

- sequential consistency
- quiescent consistency

There exists a sequential witness that preserves per-thread precedence.

There exists a sequential witness that preserves precedence across quies.state.

- linearizability

```
T1
1 push(a) 3 pop(b)

T2
2 push(b)
```

```
T1
1 push(a) 3 pop(b)

T2
2 push(b)
```
Performance and scalability

Throughput vs. # threads/cores

Ana Sokolova University of Salzburg

FRIDA, VSL 23.7.2014
Relaxations allow trading correctness for performance in a controlled way with quantitative bounds.

Stack - incorrect behavior:

\[ \text{push}(a) \text{push}(b) \text{push}(c) \text{pop}(a) \text{pop}(b) \]

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
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
- 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
The big picture

\( S_k \subseteq \Sigma^* \)

\( S \subseteq \Sigma^* \)

\( \Sigma \) - methods with arguments

semantics
sequential specification
legal sequences

relaxed semantics

Quantitative relaxations (sequential specification)
The big picture

- Quantitative relaxations (sequential specification)

- The set $S_k \subseteq \Sigma^*$ represents legal sequences.
- $S \subseteq \Sigma^*$ represents relaxed semantics.
- The distance $k$ indicates the sequential specification.
- $\Sigma$ contains methods with arguments.

---

Ana Sokolova  University of Salzburg

FRIDA, VSL 23.7.2014
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?
Quantitative relaxations (sequential specification)

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.

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

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

\( S \subseteq \sum^* \) is the sequential specification

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

- states
- labels
- initial state
- transition relation

\[ [s]_\equiv \rightarrow [sm]_\equiv \iff sm \in S \]
The framework

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

1. Start from LTS(S)
2. Add transitions with transition costs
3. 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

Ana Sokolova  University of Salzburg

FRIDA, VSL 23.7.2014
Out-of-order stack

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

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

\begin{itemize}
  \item \textbf{push} \( a \), \textbf{push} \( b \), \textbf{push} \( c \)
  \item \textbf{pop} \( a \), \textbf{top} \( b \), \textbf{top} \( c \)
\end{itemize}

Also more advanced

Ana Sokolova  University of Salzburg

FRIDA, VSL 23.7.2014
Out-of-order queue

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

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

\begin{tikzpicture}
  \node at (0,0) (a) {a};
  \node at (1,0) (b) {b};
  \node at (2,0) (c) {c};
  \node at (3,0) (d) {a};
  \node at (4,0) (e) {b};
  \draw[->] (a) -- (b) node[midway, above] {\texttt{enq}(c)};
  \draw[->] (b) -- (c) node[midway, above] {2};
  \draw[->] (c) -- (d) node[midway, above] {\texttt{deq}(c)};
\end{tikzpicture}

- Possible path cost functions \texttt{max}, \texttt{sum}, ...

Ana Sokolova  University of Salzburg

FRIDA, VSL 23.7.2014
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

FRIDA, VSL 23.7.2014
Distributed queues

Performance and Scalability comparison

"80"-core machine

![Graph showing performance and scalability comparison for various queue systems](image)
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