Local Linearizability

Ana Sokolova

joint work with:

Andreas Haas  Google
Andreas Holzer  University of Toronto
Michael Lippautz  Google
Ali Sezgin  University of Cambridge

Tom Henzinger  IST Austria
Christoph Kirsch  University of Saarbrücken
Hannes Payer  Google
Helmut Veith  TU Wien
Rigorous methods for engineering of and reasoning about reactive systems
Rigorous methods for engineering of and reasoning about reactive systems.
Background big picture
Background big picture

Computer Science
Computer Science

Theoretical Computer Science
Background big picture

Computer Science

Theoretical Computer Science

Concurrency
Background big picture

- Computer Science
  - Theoretical Computer Science
  - Concurrency
  - Formal Methods
Computer Science

Theoretical Computer Science

Concurrency

Algebra and Coalgebra

Formal Methods
Background big picture

Computer Science

Theoretical Computer Science

Concurrency

Algebra and Coalgebra

Formal Methods

Probabilistic Systems

Ana Sokolova

CiiT 2018
Background big picture

Computer Science

Theoretical Computer Science

Concurrency

Security

Algebra and Coalgebra

Formal Methods

Probabilistic Systems

Ana Sokolova

CiiT 2018
Background big picture

Computer Science

Theoretical Computer Science

Concurrency

Security

Algebra and Coalgebra

Formal Methods

Probabilistic Systems

Real-Time Systems

Ana Sokolova
Background big picture

Computer Science

Theoretical Computer Science

Security

Probabilistic Systems

Formal Methods

Algebra and Coalgebra

Concurrency

Real-Time Systems

Memory Management Systems

Ana Sokolova

CiiT 2018
Background big picture

Computer Science

Theoretical Computer Science

Data Structures

Concurrency

Security

Algebra and Coalgebra

Formal Methods

Probabilistic Systems

Real-Time Systems

Memory Management Systems

Ana Sokolova
Background big picture

Computer Science

Theoretical Computer Science

Security

Formal Methods

Probabilistic Systems

Algebra and Coalgebra

Concurrency

Real-Time Systems

Memory Management

Systems

Data Structures

QAANTS

NQTA

SAQUN

TSNQ

ESTI

CiiT 2018
Current favourites

Computer Science

- Theoretical Computer Science
- Concurrency
- Formal Methods
- Algebra and Coalgebra
- Data Structures
- Memory Management
- Real-Time Systems
- Systems
- Probabilistic Systems
- Security

Ana Sokolova

CiiT 2018
Local Linearizability

Ana Sokolova

joint work with:

Andreas Haas
Andreas Holzer
Michael Lippautz
Ali Sezgin

Tom Henzinger
Christoph Kirsch
Hannes Payer
Helmut Veith
Concurrent Data Structures: Semantics and Relaxations
Concurrent Data Structures
Correctness and Performance
Concurrent Data Structures
Correctness and Performance

structure and power
Semantics of concurrent data structures
Semantics of concurrent data structures

e.g. pools, queues, stacks
Semantics of concurrent data structures

e.g. pools, queues, stacks

t1: enq(2) deq(1)
t2: enq(1) deq(2)
Semantics of concurrent data structures

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

- **Consistency condition** = e.g. linearizability / sequential consistency

\[ \begin{align*}
\text{t1:} & \quad \text{enq(2) deq(1)} \\
\text{t2:} & \quad \text{enq(1) deq(2)}
\end{align*} \]

E.g. pools, queues, stacks
Semantics of concurrent data structures

- **Sequential specification** = set of legal sequences
  
  - e.g. pools, queues, stacks

- **Consistency condition** = e.g. linearizability / sequential consistency
  
  - e.g. queue legal sequence
    enq(1)enq(2)deq(1)deq(2)
Semantics of concurrent data structures

- **Sequential specification** = set of legal sequences
  - e.g. queue legal sequence: `enq(1)enq(2)deq(1)deq(2)`

- **Consistency condition** = e.g. linearizability / sequential consistency
  - e.g. the concurrent history above is a linearizable queue concurrent history

- Examples:
  - Pools, queues, stacks
  - Queue legal sequence: `enq(1)enq(2)deq(1)deq(2)`

Example concurrent histories:

- $t_1$: `enq(2) deq(1)`
- $t_2$: `enq(1) deq(2)`
Consistency conditions

Linearizability  [Herlihy,Wing ’90]

Sequential Consistency  [Lamport’79]
Consistency conditions

there exists a legal sequence that preserves precedence

Linearizability [Herlihy, Wing '90]

Sequential Consistency [Lamport '79]
Consistency conditions

Linearizability [Herlihy, Wing ’90]

There exists a legal sequence that preserves precedence

Sequential Consistency [Lamport’79]

\[
\begin{array}{cccc}
\text{t1:} & \text{enq(2)} & \text{deq(1)} \\
\text{t2:} & \text{enq(1)} & \text{deq(2)} \\
\end{array}
\]
Consistency conditions

- There exists a legal sequence that preserves precedence

**Linearizability**  [Herlihy,Wing ’90]

```
t1: enq(2) deq(1)
t2: enq(1) deq(2)
```

**Sequential Consistency**  [Lamport’79]
Consistency conditions

Linearizability  [Herlihy, Wing '90]

there exists a legal sequence that preserves precedence

Sequential Consistency  [Lamport '79]
Consistency conditions

- Linearizability [Herlihy, Wing ’90]
  - There exists a legal sequence that preserves precedence.

```
t1:  enq(2)  deq(1)  enq(1)  deq(2)
```

- Sequential Consistency [Lamport’79]
  - There exists a legal sequence that preserves per-thread precedence (program order).

```
t1:  enq(2)  deq(1)
```
```
t2:  enq(1)  deq(2)
```
Consistency conditions

Linearizability  [Herlihy,Wing ’90]

there exists a legal sequence that preserves precedence

t1: enq(2)
    deq(1)

t2: 1 enq(1)  deq(2)

Sequential Consistency  [Lamport’79]

there exists a legal sequence that preserves per-thread precedence (program order)

t1: enq(1)  deq(2)

t2: deq(1)  enq(2)
Consistency conditions

Linearizability [Herlihy, Wing '90]

Sequential Consistency [Lamport'79]

there exists a legal sequence that preserves precedence

there exists a legal sequence that preserves per-thread precedence (program order)
Consistency conditions

- **Linearizability** [Herlihy, Wing ’90]
  
  - There exists a legal sequence that preserves precedence.
  - Example: 
    - t1: enq(2) → deq(1) → enq(1) → deq(2)
    - t2: enq(1) → deq(2)

- **Sequential Consistency** [Lamport’79]
  
  - There exists a legal sequence that preserves per-thread precedence (program order).
  - Example: 
    - t1: enq(1) → deq(2)
    - t2: deq(1) → enq(2)
Performance and scalability

throughput

# of threads / cores

:-)))

:-)

:-(

:-\
Relaxations allow trading correctness for performance
Relaxations allow trading correctness for performance.
Relaxing the Semantics
Relaxing the Semantics

• Sequential specification = set of legal sequences

• Consistency condition = e.g. linearizability / sequential consistency
Relaxing the Semantics

- **Sequential specification** = set of legal sequences
- **Consistency condition** = e.g. linearizability / sequential consistency

Quantitative relaxations
Henzinger, Kirsch, Payer, Sezgin, S. POPL13
Relaxing the Semantics

- Sequential specification = set of legal sequences
- Consistency condition = e.g. linearizability / sequential consistency

Quantitative relaxations
Henzinger, Kirsch, Payer, Sezgin, S. POPL 13

not "sequentially correct"
Relaxing the Semantics

- Sequential specification = set of legal sequences
- Consistency condition = e.g. linearizability / sequential consistency

Quantitative relaxations
Henzinger, Kirsch, Payer, Sezgin, S. POPL13

Local linearizability
in this talk

not “sequentially correct”
Relaxing the Semantics

- **Sequential specification** = set of legal sequences
- **Consistency condition** = e.g. linearizability / sequential consistency

Quantitative relaxations
Henzinger, Kirsch, Payer, Sezgin, S. POPL13

Local linearizability in this talk

for queues only (feel free to ask for more)

not “sequentially correct”
Relaxing the Semantics

- **Sequential specification** = set of legal sequences
- **Consistency condition** = e.g. linearizability / sequential consistency

for queues only (feel free to ask for more)

not "sequentially correct"

Local linearizability in this talk

Quantitative relaxations
Henzinger, Kirsch, Payer, Sezgin, S. POPL 13

too weak
Relaxing the Consistency Condition
Relaxing the Consistency Condition
Local Linearizability
main idea
Local Linearizability

main idea

- Partition a history into a set of local histories
- Require linearizability per local history
Local Linearizability
main idea

• Partition a history into a set of local histories
• Require linearizability per local history

Already present in some shared-memory consistency conditions (not in our form of choice)
Local Linearizability main idea

- **Partition** a history into a set of local histories
- **Require** linearizability per local history

Already present in some shared-memory consistency conditions (not in our form of choice)

Local sequential consistency… is also possible
Local Linearizability
main idea

- **Partition** a history into a set of local histories
- **Require** linearizability per local history

Already present in some shared-memory consistency conditions (not in our form of choice)

Local sequential consistency… is also possible

no global witness
Local Linearizability (queue) example

t1: enq(1)       deq(2)

t2: enq(2)       deq(1)
Local Linearizability (queue) example

(sequential) history not linearizable

<table>
<thead>
<tr>
<th>t1:</th>
<th>enq(1)</th>
<th>deq(2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>t2:</td>
<td>enq(2)</td>
<td>deq(1)</td>
</tr>
</tbody>
</table>
Local Linearizability (queue) example

(sequential) history not linearizable
Local Linearizability (queue) example

(sequential) history not linearizable

t1-induced history, linearizable
Local Linearizability (queue) example

(t1-induced history, linearizable)

(sequential) history not linearizable
Local Linearizability (queue) example

(t1-induced history, linearizable)
(t2-induced history, linearizable)

(sequential) history not linearizable
Local Linearizability (queue) example

\[ \text{t1: enq(1) deq(2) enq(2) deq(1)} \]

(sequential) history not linearizable

\[ \text{t2-induced history, linearizable} \]

\[ \text{t1-induced history, linearizable} \]

locally linearizable
Local Linearizability (queue) definition
Local Linearizability (queue) definition

Queue signature $\Sigma = \{\text{enq}(x) \mid x \in V\} \cup \{\text{deq}(x) \mid x \in V\} \cup \{\text{deq}(\text{empty})\}$
Local Linearizability (queue) definition

Queue signature $\Sigma = \{\text{enq}(x) \mid x \in V\} \cup \{\text{deq}(x) \mid x \in V\} \cup \{\text{deq}(\text{empty})\}$

For a history $h$ with a thread $T$, we put

$I_T = \{\text{enq}(x)^T \in h \mid x \in V\}$

$O_T = \{\text{deq}(x)^T \in h \mid \text{enq}(x)^T \in I_T\} \cup \{\text{deq}(\text{empty})\}$
Local Linearizability (queue) definition

Queue signature $\Sigma = \{\text{enq}(x) \mid x \in V\} \cup \{\text{deq}(x) \mid x \in V\} \cup \{\text{deq}(\text{empty})\}$

For a history $h$ with a thread $T$, we put

$I_T = \{\text{enq}(x)^T \in h \mid x \in V\}$

$O_T = \{\text{deq}(x)^T \in h \mid \text{enq}(x)^T \in I_T\} \cup \{\text{deq}(\text{empty})\}$

in-methods of thread $T$ are enqueues performed by thread $T$
Local Linearizability (queue) definition

Queue signature $\Sigma = \{\text{enq}(x) \mid x \in V\} \cup \{\text{deq}(x) \mid x \in V\} \cup \{\text{deq}(\text{empty})\}$

For a history $h$ with a thread $T$, we put

$$I_T = \{\text{enq}(x)^T \in h \mid x \in V\}$$

$$O_T = \{\text{deq}(x)^{T'} \in h \mid \text{enq}(x)^T \in I_T\} \cup \{\text{deq}(\text{empty})\}$$

- in-methods of thread $T$ are enqueues performed by thread $T$
- out-methods of thread $T$ are dequeues (performed by any thread) corresponding to enqueues that are in-methods

CiiT 2018
Local Linearizability (queue) definition

Queue signature \( \Sigma = \{\text{enq}(x) \mid x \in V\} \cup \{\text{deq}(x) \mid x \in V\} \cup \{\text{deq}(\text{empty})\} \)

For a history \( h \) with a thread \( T \), we put

\[ I_T = \{\text{enq}(x)^T \in h \mid x \in V\} \]
\[ O_T = \{\text{deq}(x)^T \in h \mid \text{enq}(x)^T \in I_T\} \cup \{\text{deq}(\text{empty})\} \]

\( h \) is locally linearizable iff every thread-induced history \( h_T = h \mid (I_T \cup O_T) \) is linearizable.
Generalizations of Local Linearizability
Generalizations of Local Linearizability

Signature $\Sigma$
Generalizations of Local Linearizability

Signature $\Sigma$

For a history $h$ with $n$ threads, choose

$\text{In}_h(i)$

$\text{Out}_h(i)$
Generalizations of Local Linearizability

Signature $\Sigma$

For a history $h$ with $n$ threads, choose

$\text{In}_h(i)$

$\text{Out}_h(i)$

in-methods of thread $i$, methods that go in $h_i$
Generalizations of Local Linearizability

Signature $\Sigma$

For a history $h$ with $n$ threads, choose:

- $\text{In}_h(i)$: in-methods of thread $i$
- $\text{Out}_h(i)$: out-methods of thread $i$

- methods that go in $h_i$
- dependent methods on the methods in $\text{In}_h(i)$ (performed by any thread)
Generalizations of Local Linearizability

Signature $\Sigma$

For a history $h$ with $n$ threads, choose

$\text{In}_h(i)$

$\text{Out}_h(i)$

$\text{in-methods of thread } i$, methods that go in $h_i$

$\text{out-methods of thread } i$, dependent methods on the methods in $\text{In}_h(i)$ (performed by any thread)

$h$ is locally linearizable iff every thread-induced history $h_i = h \mid (\text{In}_h(i) \cup \text{Out}_h(i))$ is linearizable.
Generalizations of Local Linearizability

Signature $\Sigma$

For a history $h$ with $n$ threads, choose:

- $\text{In}_h(i)$
- $\text{Out}_h(i)$

by increasing the in-methods, LL gradually moves to linearizability.

Out-methods of thread $i$, dependent methods on the methods in $\text{In}_h(i)$ (performed by any thread)

$h$ is locally linearizable iff every thread-induced history $h_i = h | (\text{In}_h(i) \cup \text{Out}_h(i))$ is linearizable.
Where do we stand?

In general

Local Linearizability

Linearizability

Sequential Consistency
Where do we stand?

For queues (and most container-type data structures)

Local Linearizability

Linearizability

Sequential Consistency
Properties
Properties

Local linearizability is compositional
Local linearizability is compositional

like linearizability
unlike sequential consistency
Properties

Local linearizability is compositional

$h$ (over multiple objects) is locally linearizable
iff
each per-object subhistory of $h$ is locally linearizable

like linearizability
unlike sequential consistency
Properties

Local linearizability is compositional

\( h \) (over multiple objects) is locally linearizable iff each per-object subhistory of \( h \) is locally linearizable

Local linearizability is modular / “decompositional”

like linearizability

unlike sequential consistency
Local linearizability is compositional

\( h \) (over multiple objects) is locally linearizable

iff

each per-object subhistory of \( h \) is locally linearizable

Local linearizability is modular / “decompositional”

like linearizability

unlike sequential consistency

uses decomposition into smaller histories, by definition
Local linearizability is compositional

\( h \) (over multiple objects) is locally linearizable

iff

each per-object subhistory of \( h \) is locally linearizable

Local linearizability is modular / “decompositional”

uses decomposition into smaller histories, by definition

may allow for modular verification

like linearizability

unlike sequential consistency

Ana Sokolova University of Salzburg

CiiT 2018
Verification (queue)

**Queue sequential specification (axiomatic)**

\[ s \text{ is a legal queue sequence}
\quad \text{iff}
\]

1. \( s \text{ is a legal pool sequence, and} \)
2. \( \text{enq}(x) <_s \text{enq}(y) \land \text{deq}(y) \in s \quad \Rightarrow \quad \text{deq}(x) \in s \land \text{deq}(x) <_s \text{deq}(y) \)
Verification (queue)

Queue sequential specification (axiomatic)

\[ \textbf{s} \text{ is a legal queue sequence iff} \]
\[ \begin{align*}
1. & \quad \textbf{s} \text{ is a legal pool sequence, and} \\
2. & \quad \text{enq}(x) <_s \text{enq}(y) \land \text{deq}(y) \in \textbf{s} \quad \Rightarrow \quad \text{deq}(x) \in \textbf{s} \land \text{deq}(x) <_s \text{deq}(y)
\end{align*} \]

Queue linearizability (axiomatic)

\[ \textbf{h} \text{ is queue linearizable iff} \]
\[ \begin{align*}
1. & \quad \textbf{h} \text{ is pool linearizable, and} \\
2. & \quad \text{enq}(x) <_h \text{enq}(y) \land \text{deq}(y) \in \textbf{h} \quad \Rightarrow \quad \text{deq}(x) \in \textbf{h} \land \text{deq}(y) <_h \text{deq}(x)
\end{align*} \]
Verification (queue)

Queue sequential specification (axiomatic)

\[ s \text{ is a legal queue sequence iff 1. } s \text{ is a legal pool sequence, and 2. } \text{enq}(x) <_s \text{enq}(y) \land \text{deq}(y) \in s \implies \text{deq}(x) \in s \land \text{deq}(x) <_s \text{deq}(y) \]

Queue linearizability (axiomatic)

\[ h \text{ is queue linearizable iff 1. } h \text{ is pool linearizable, and 2. } \text{enq}(x) <_h \text{enq}(y) \land \text{deq}(y) \in h \implies \text{deq}(x) \in h \land \text{deq}(y) <_h \text{deq}(x) \]
Verification (queue)

Queue sequential specification (axiomatic)

\( s \) is a legal queue sequence
iff
1. \( s \) is a legal pool sequence, and
2. \( \text{enq}(x) \prec_s \text{enq}(y) \land \text{deq}(y) \in s \) \( \Rightarrow \) \( \text{deq}(x) \in s \land \text{deq}(x) \prec_s \text{deq}(y) \)

Queue local linearizability (axiomatic)

\( h \) is queue locally linearizable
iff
1. \( h \) is pool locally linearizable, and
2. \( \text{enq}(x) \prec_h \text{enq}(y) \land \text{deq}(y) \in h \) \( \Rightarrow \) \( \text{deq}(x) \in h \land \text{deq}(y) \preceq_h \text{deq}(x) \)
Verification (queue)

Queue sequential specification (axiomatic)

$s$ is a legal queue sequence
iff
1. $s$ is a legal pool sequence, and
2. $\text{enq}(x) \prec_s \text{enq}(y) \land \text{deq}(y) \in s \implies \text{deq}(x) \in s \land \text{deq}(x) \prec_s \text{deq}(y)$

Queue local linearizability (axiomatic)

$h$ is queue locally linearizable
iff
1. $h$ is pool locally linearizable, and
2. $\text{enq}(x) \prec_h \text{enq}(y) \land \text{deq}(y) \in h \implies \text{deq}(x) \in h \land \text{deq}(y) \prec_h \text{deq}(x)$

thread-local precedence order
Generic Implementations
Generic Implementations

Your favorite linearizable data structure implementation
Generic Implementations

Your favorite linearizable data structure implementation
Generic Implementations

Your favorite linearizable data structure implementation

turns into a locally linearizable implementation by:
Generic Implementations

Your favorite linearizable data structure implementation

turns into a locally linearizable implementation by:

\[ t_1 \quad t_2 \quad \ldots \quad t_n \]

\[ \Phi \quad \Phi \quad \Phi \]
Generic Implementations

Your favorite linearizable data structure implementation

turns into a locally linearizable implementation by:

segment of possibly dynamic size (n)
Generic Implementations

Your favorite linearizable data structure implementation

turns into a locally linearizable implementation by:

- segment of possibly dynamic size (n)
- local inserts / global (randomly distributed) removes
Generic Implementations

Your favorite linearizable data structure implementation

turns into a locally linearizable implementation by:

segment of possibly dynamic size (n)

local inserts / global (randomly distributed) removes

LLD \( \Phi \)
(locally linearizable)

Ana Sokolova
CiiT 2018
Generic Implementations

Your favorite linearizable data structure implementation

turns into a locally linearizable implementation by:

LLD Φ (locally linearizable)

LL+D Φ (also pool linearizable)

segment of possibly dynamic size (n)

local inserts / global (randomly distributed) removes
Performance

![Performance graph](image)

(a) Queues, LL queues, and “queue-like” pools
Performance

LL+D MS queue performs significantly better than MS queue

(a) Queues, LL queues, and “queue-like” pools
Performance

(a) Queues, LL queues, and “queue-like” pools
Performance

LL+D MS queue performs better than the best known pools

(a) Queues, LL queues, and “queue-like” pools
Local Linearizability

Ana Sokolova

joint work with:

Andreas Haas
Andreas Holzer
Michael Lippautz
Ali Sezgin

Tom Henzinger
Christoph Kirsch
Hannes Payer
Helmut Veith
Local Linearizability

Ana Sokolova

joint work with:

Andreas Haas, Google
Andreas Holzer, University of Toronto
Michael Lippautz, Google
Ali Sezgin, University of Cambridge

Tom Henzinger, IST Austria
Christoph Kirsch, University of Salzburg
Hannes Payer, Google
Helmut Veith, TU Wien