

Abstraction for Concurrent Objects

Hongseok Yang (Queen Mary Univ. of London)

Joint work with
Ivana Filipovic, Peter O'Hearn, Noam Rinetzky

Data Abstraction for Concurrency

- Is it OK to replace a sequential queue in my program by Doug Lea's `ConcurrentLinkedQueue`?
- By OK, we mean: after the replacement,
 - my program does not generate new outputs;
 - its termination behavior does not change.
- Our aim is to develop a theory that answers such questions.

Linearizability

- Lea's ConcurrentLinkedQueue is proved to be linearizable:

Simple, Fast, and Practical Non-Blocking and Blocking Concurrent Queue Algorithms*

Maged M. Michael Michael L. Scott

3.2 Linearizability

The presented algorithms are linearizable because there is a specific point during each operation at which it is considered to “take effect” [5]. An enqueue takes effect when

Linearizability

- Widely used correctness condition for concurrent data structures (or objects).
- Usually expresses a relationship between concurrent object and sequential object.
- The goal of this talk is to show the connection between linearizability and the “can-we-replace” question.

Review of Linearizability

Object System

- A set of finite traces (i.e., sequences of “actions”).
- Gives trace semantics of an object.
- Concurrent queue:

Cqueue = {
 (a,enq(0))(a,ret),
 (a,enq(0))(a,ret)(b,deq)(b,ret(0)),
 (a,enq(0))(b,deq)(a,ret)(b,ret(0)), ... }

- Each element h in an obj. system is called history.

History

- A history h is a finite sequence of calls/returns:

$$h = (a, \text{enq}(0))(a, \text{ret})(b, \text{deq})(b, \text{ret}(0))$$
$$h = (a, \text{enq}(0))(b, \text{deq})(a, \text{ret})(b, \text{ret}(0))$$

- “a,b” in $(a, \text{enq}(0))$ and (b, deq) are thread ids.
- Each method invocation is split into call and return.

History

- A history h is a finite sequence of calls/returns:

$$h = (a, \text{enq}(0))(a, \text{ret})(b, \text{deq})(b, \text{ret}(0))$$
$$h = (a, \text{enq}(0))(b, \text{deq})(a, \text{ret})(b, \text{ret}(0))$$

- “a,b” in $(a, \text{enq}(0))$ and (b, deq) are thread ids.
- Each method invocation is split into call and return.
- We will consider only well-formed histories where:

History

- A history (a, ret) is a finite sequence of calls/returns:

$h = (a, \text{enq}(0))(a, \text{ret})(b, \text{deq})(b, \text{ret}(0))$ ✗

$h = (a, \text{enq}(0))(b, \text{deq})(a, \text{ret})(b, \text{ret}(0))$

- “a,b” in $(a, \text{enq}(0))$ and (b, deq) are thread ids.
- Each method invocation is split into call and return.
- We will consider only well-formed histories where:
 - calls have matching returns and vice versa;

History

- A history h is a finite sequence of calls/returns:

$h = (a, \text{enq}(0))(a, \text{ret})(b, \text{deq})(b, \text{ret}(0))$

$h = (a, \text{enq}(0))(b, \text{deq})(a, \text{ret})(b, \text{ret}(0))$ 

- “a,b” in $(a, \text{enq}(0))$ and $(a, \text{enq}(1))$ are thread ids.
- Each method invocation is split into call and return.
- We will consider only well-formed histories where:
 - calls have matching returns and vice versa;
 - after a call, each thread waits until the call returns.

Linearizability

- Binary relation on object systems.
- OS1 is linearizable wrt. OS2 iff

$$\forall h1 \in OS1. \exists h2 \in OS2. h1 [\text{LinR}] h2.$$

- Usually, OS2 is a sequential object (sequential queue) and OS1 is a concurrent object (concurrent queue).
- LinR is a relation on histories, and it is the key notion behind linearizability.

Relation LinR

- $h1 [\text{LinR}] h2$ holds iff $h2$ is a rearrangement of $h1$ s.t.
 1. $\text{proj}(h1, a) = \text{proj}(h2, a)$ for all thread-ids a , and
 2. the happen-before order of $h1$ is preserved by $h2$.
- Example:

$h1 = (a, \text{enq}(0)) (b, \text{sayHi}) (b, \text{ret}) (a, \text{ret}) (c, \text{deq}) (c, \text{ret}(0))$

$[\text{LinR}]$

$h2 = (a, \text{enq}(0)) (a, \text{ret}) (b, \text{sayHi}) (b, \text{ret}) (c, \text{deq}) (c, \text{ret}(0))$

Relation LinR

- $h1 [\text{LinR}] h2$ holds iff $h2$ is a rearrangement of $h1$ s.t.
 1. $\text{proj}(h1, a) = \text{proj}(h2, a)$ for all thread-ids a , and
 2. the happen-before order of $h1$ is preserved by $h2$.
- Example:

$h1 = (a, \text{enq}(0)) (b, \text{sayHi}) (b, \text{ret}) (a, \text{ret}) (c, \text{deq}) (c, \text{ret}(0)) \quad \checkmark$

$[\text{LinR}]$

$h2 = (a, \text{enq}(0)) (a, \text{ret}) (b, \text{sayHi}) (b, \text{ret}) (c, \text{deq}) (c, \text{ret}(0))$

Relation LinR

- $h1 [\text{LinR}] h2$ holds iff $h2$ is a rearrangement of $h1$ s.t.
 1. $\text{proj}(h1, a) = \text{proj}(h2, a)$ for all thread-ids a , and
 2. the happen-before order of $h1$ is preserved by $h2$.
- Example:

$h1' = (a, \text{enq}(0))(a, \text{ret})(b, \text{sayHi})(b, \text{ret})(c, \text{deq})(c, \text{ret}(4))$ ✗

$h1 = (a, \text{enq}(0))(b, \text{sayHi})(b, \text{ret})(a, \text{ret})(c, \text{deq})(c, \text{ret}(0))$ ✓

[LinR]

$h2 = (a, \text{enq}(0))(a, \text{ret})(b, \text{sayHi})(b, \text{ret})(c, \text{deq})(c, \text{ret}(0))$

Relation LinR

- $h1 [\text{LinR}] h2$ holds iff $h2$ is a rearrangement of $h1$ s.t.
 1. $\text{proj}(h1, a) = \text{proj}(h2, a)$ for all thread-ids a , and
 2. the happen-before order of $h1$ is preserved by $h2$.
 - Example:
 - $h1'' = (b, \text{sayHi})(b, \text{ret})(a, \text{enq}(0))(a, \text{ret})(c, \text{deq})(c, \text{ret}(0))$ ✗
 - $h1' = (a, \text{enq}(0))(a, \text{ret})(b, \text{sayHi})(b, \text{ret})(c, \text{deq})(c, \text{ret}(4))$ ✗
 - $h1 = (a, \text{enq}(0))(b, \text{sayHi})(b, \text{ret})(a, \text{ret})(c, \text{deq})(c, \text{ret}(0))$ ✓
- [LinR]**
- $h2 = (a, \text{enq}(0))(a, \text{ret})(b, \text{sayHi})(b, \text{ret})(c, \text{deq})(c, \text{ret}(0))$

Relation LinR

- $h1 [\text{LinR}] h2$ holds iff $h2$ is a rearrangement of $h1$ s.t.

1. $\text{proj}(h1, a) = \text{proj}(h2, a)$ for all thread-ids a , and

2. the happen-before order of $h1$ is preserved by $h2$.

- Example:

$h1'' = (b, \text{sayHi})(b, \text{ret})(a, \text{enq}(0))(a, \text{ret})(c, \text{deq})(c, \text{ret}(0))$ ✗

$h1' = (a, \text{enq}(0))(a, \text{ret})(b, \text{sayHi})(b, \text{ret})(c, \text{deq})(c, \text{ret}(4))$ ✗

$h1 = (a, \text{enq}(0))(b, \text{sayHi})(b, \text{ret})(a, \text{ret})(c, \text{deq})(c, \text{ret}(0))$ ✓

[LinR]

$h2 = (a, \text{enq}(0))(a, \text{ret})(b, \text{sayHi})(b, \text{ret})(c, \text{deq})(c, \text{ret}(0))$

Our Results

History viewed as Abstraction

- $\text{mean}(h) = \{ t \mid \text{ProjectObjectActions}(t) = h \}$.
- Says all traces whose interactions with the object are h .

History viewed as Abstraction

- $\text{mean}(h) = \{ t \mid \text{ProjectObjectActions}(t) = h \}$.
- Says all traces whose interactions with the object are h .
 $\text{mean}((a, \text{enq}(0))(a, \text{ret})(b, \text{sayHi})(b, \text{ret})) = \{$

History viewed as Abstraction

- $\text{mean}(h) = \{ t \mid \text{ProjectObjectActions}(t) = h \}$.
- Says all traces whose interactions with the object are h .

$\text{mean}((a, \text{enq}(0))(a, \text{ret})(b, \text{sayHi})(b, \text{ret})) = \{$
 $(a, \text{enq}(0))(a, \text{ret})(b, \text{sayHi})(a, x:=8)(b, \text{ret}),$

History viewed as Abstraction

- $\text{mean}(h) = \{ t \mid \text{ProjectObjectActions}(t) = h \}$.
- Says all traces whose interactions with the object are h .

$\text{mean}((a, \text{enq}(0))(a, \text{ret})(b, \text{sayHi})(b, \text{ret})) = \{$
 $(a, \text{enq}(0))(a, \text{ret})(b, \text{sayHi})(a, x:=8)(b, \text{ret}),$
 $(a, \text{enq}(0))(b, y:=0)(a, \text{ret})(b, \text{sayHi})(a, x:=8)(b, \text{ret}),$

History viewed as Abstraction

- $\text{mean}(h) = \{ t \mid \text{ProjectObjectActions}(t) = h \}$.
- Says all traces whose interactions with the object are h .

$\text{mean}((a, \text{enq}(0))(a, \text{ret})(b, \text{sayHi})(b, \text{ret})) = \{$
 $(a, \text{enq}(0))(a, \text{ret})(b, \text{sayHi})(a, x:=8)(b, \text{ret}),$
 $(a, \text{enq}(0))(b, y:=0)(a, \text{ret})(b, \text{sayHi})(a, x:=8)(b, \text{ret}),$
 $(a, \text{enq}(0))(a, \text{ret})(a, x:=8)(b, \text{assume}(x=8))(b, \text{sayHi})(b, \text{ret}),$
 $\dots \}$

History viewed as Abstraction

- $\text{mean}(h) = \{ t \mid \text{ProjectObjectActions}(t) = h \}$.
- Says all traces whose interactions with the object are h .

$\text{mean}((a, \text{enq}(0))(a, \text{ret})(b, \text{sayHi})(b, \text{ret})) = \{$
 $(a, \text{enq}(0))(a, \text{ret})(b, \text{sayHi})(a, x:=8)(b, \text{ret}),$
 $(a, \text{enq}(0))(b, y:=0)(a, \text{ret})(b, \text{sayHi})(a, x:=8)(b, \text{ret}),$
 $(a, \text{enq}(0))(a, \text{ret})(a, x:=8)(b, \text{assume}(x=8))(b, \text{sayHi})(b, \text{ret}),$
 $\dots \}$

- $(a, \text{enq}(0))(a, x:=0)(a, \text{ret})(b, \text{sayHi})(b, \text{ret})$ is not in the set.

Main Result I

[Theorem I] For all actions i, j of h ,

$\text{HappenBefore}(i, j, h)$ iff $\exists t \in \text{mean}(h). \text{Depend}(i, j, t)$.

- Says “HappenBefore” is an abstraction of dependency.
- only-if by example: $h = (a, \text{enq}(0))(a, \text{ret})(b, \text{sayHi})(b, \text{ret})$

Main Result I

[Theorem I] For all actions i, j of h ,

$\text{HappenBefore}(i, j, h)$ iff $\exists t \in \text{mean}(h). \text{Depend}(i, j, t)$.

- Says “HappenBefore” is an abstraction of dependency.
- only-if by example: $h = (a, \text{enq}(0))(a, \text{ret})(b, \text{sayHi})(b, \text{ret})$
 $\vee (a, \text{enq}(0))(a, \text{ret})(a, x := 1)(b, \text{assume}(x = 1))(b, \text{sayHi})(b, \text{ret})$

Main Result I

[Theorem I] For all actions i, j of h ,

$\text{HappenBefore}(i, j, h)$ iff $\exists t \in \text{mean}(h). \text{Depend}(i, j, t)$.

- Says “HappenBefore” is an abstraction of dependency.
- only-if by example: $h = (a, \text{enq}(0))(a, \text{ret})(b, \text{sayHi})(b, \text{ret})$
- $\checkmark (a, \text{enq}(0))(a, \text{ret})(a, x := 1)(b, \text{assume}(x = 1))(b, \text{sayHi})(b, \text{ret})$
- if by example: $h = (a, \text{enq}(0))(b, \text{sayHi})(a, \text{ret})(b, \text{ret})$

Main Result I

[Theorem I] For all actions i, j of h ,

$\text{HappenBefore}(i, j, h)$ iff $\exists t \in \text{mean}(h). \text{Depend}(i, j, t)$.

- Says “HappenBefore” is an abstraction of dependency.
- only-if by example: $h = (a, \text{enq}(0))(a, \text{ret})(b, \text{sayHi})(b, \text{ret})$
 $\checkmark (a, \text{enq}(0))(a, \text{ret})(a, x := 1)(b, \text{assume}(x = 1))(b, \text{sayHi})(b, \text{ret})$
- if by example: $h = (a, \text{enq}(0))(b, \text{sayHi})(a, \text{ret})(b, \text{ret})$
 $\times (a, \text{enq}(0))(b, \text{sayHi})(a, \text{ret})(a, x := 1)(b, \text{assume}(x = 1))(b, \text{ret})$

Trace Equivalence

- $t1 \sim t2$ iff $t2$ can be obtained from $t1$ by swapping independent actions.
- $(a, x:=4)(b, y:=11) \sim (b, y:=11)(a, x:=4)$
- $(a, x:=4)(b, x:=11) \not\sim (b, x:=11)(a, x:=4)$
- $(a, x:=4)(a, y:=11) \not\sim (a, y:=11)(a, x:=4)$
- $(a, \text{enq}(0))(a, \text{ret})(b, \text{sayHi}) \sim (a, \text{enq}(0))(b, \text{sayHi})(a, \text{ret})$

Main Result 2

[Theorem 2]

$h1 [\text{LinR}] h2$ iff $\forall t1 \in \text{mean}(h1). \exists t2 \in \text{mean}(h2). t1 \sim t2$.

- Says 1) $\text{mean}(h1)$ is a subset of $\text{mean}(h2)$ in a sense, and 2) we can always replace $h2$ by $h1$.
- only-if by example:

$h1 = (a, \text{enq}(0))(b, \text{sayHi})(b, \text{ret})(a, \text{ret})$

$t1 = (a, \text{enq}(0))(b, \text{sayHi})(b, \text{ret})(b, x := 1)(a, \text{ret})$

$h2 = (a, \text{enq}(0))(a, \text{ret})(b, \text{sayHi})(b, \text{ret})$

Main Result 2

[Theorem 2]

$h1 [\text{LinR}] h2$ iff $\forall t1 \in \text{mean}(h1). \exists t2 \in \text{mean}(h2). t1 \sim t2$.

- Says 1) $\text{mean}(h1)$ is a subset of $\text{mean}(h2)$ in a sense, and 2) we can always replace $h2$ by $h1$.
- only-if by example:

$h1 = (a, \text{enq}(0))(b, \text{sayHi})(b, \text{ret})(a, \text{ret})$

$t1 = (a, \text{enq}(0))(b, \text{sayHi})(b, \text{ret})(b, x := 1)(a, \text{ret})$

$h2 = (a, \text{enq}(0))(a, \text{ret})(b, \text{sayHi})(b, \text{ret})$

$t2 = (a, \text{enq}(0))(a, \text{ret})(b, \text{sayHi})(b, \text{ret})(b, x := 1)$

Main Result 2

[Theorem 2]

$h1 [\text{LinR}] h2$ iff $\forall t1 \in \text{mean}(h1). \exists t2 \in \text{mean}(h2). t1 \sim t2$.

- Says 1) $\text{mean}(h1)$ is a subset of $\text{mean}(h2)$ in a sense, and 2) we can always replace $h2$ by $h1$.

- if by example:

$h1 = (b, \text{sayHi})(b, \text{ret})(a, \text{enq}(0))(a, \text{ret})$

$t1 = (b, \text{sayHi})(b, \text{ret})(b, x := 1)(a, \text{assume}(x = 1))(a, \text{enq}(0))(a, \text{ret})$

$h2 = (a, \text{enq}(0))(a, \text{ret})(b, \text{sayHi})(b, \text{ret})$

Main Result 2

[Theorem 2]

$h1 [\text{LinR}] h2$ iff $\forall t1 \in \text{mean}(h1). \exists t2 \in \text{mean}(h2). t1 \sim t2$.

- Says 1) $\text{mean}(h1)$ is a subset of $\text{mean}(h2)$ in a sense, and 2) we can always replace $h2$ by $h1$.

- if by example:

$h1 = (b, \text{sayHi})(b, \text{ret})(a, \text{enq}(0))(a, \text{ret})$

$t1 = (b, \text{sayHi})(b, \text{ret})(b, x:=1)(a, \text{assume}(x=1))(a, \text{enq}(0))(a, \text{ret})$

$h2 = (a, \text{enq}(0))(a, \text{ret})(b, \text{sayHi})(b, \text{ret})$

$t2 = (a, \text{assume}(x=1))(a, \text{enq}(0))(a, \text{ret})(b, \text{sayHi})(b, \text{ret})(b, x:=1)$

Main Result 2

[Theorem 2]

$h1 [\text{LinR}] h2 \text{ iff } \forall t1 \in \text{mean}(h1). \exists t2 \in \text{mean}(h2). t1 \sim t2.$

[Corollary]

If we ignore the termination issue, the answer to the following question is yes:

- Is it OK to replace a sequential queue in my program by Doug Lea's `ConcurrentLinkedQueue`?

Main Result 2

[Theorem 2]

$h1 [\text{LinR}] h2$ iff $\forall t1 \in \text{mean}(h1). \exists t2 \in \text{mean}(h2). t1 \sim t2.$

[Corollary]

If we ignore the termination issue, the answer to the following question is yes:

- Is it OK to replace a sequential queue in my program by Doug Lea's ConcurrentLinkedQueue?

[Theorem 3] Linearizability is the same as obs. refinement.