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;
  - my program does not change termination behavior.
- 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:

  Cstack = {

  (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,enq(0))(a,ret)(b,deq)(b,ret(0))

  h = (a,enq(0))(b,deq)(a,ret)(b,ret(0))

- "a,b" in (a,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 (possibly except the last one) have matching returns.

Linearizability

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

$$\forall h1 \in OS1.\ \exists h2 \in OS2.\ \ h1\,[LinR]\,h2.$$

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

h1 = (a,enq(0))(b,sayHi)(b,ret)(a,ret)(c,deq)(c,ret(0))

[LinR]

h2 = (a,enq(0))(a,ret)(b,sayHi)(b,ret)(c,deq)(c,ret(0))

Relation LinR

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

h1 = (a,enq(0))(b,sayHi)(b,ret)(a,ret)(c,deq)(c,ret(0)) ∨

[LinR]

h2 = (a,enq(0))(a,ret)(b,sayHi)(b,ret)(c,deq)(c,ret(0))

Relation LinR

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

h1' = (a,enq(0))(a,ret)(b,sayHi)(b,ret)(c,deq)(c,ret(4)) ✗

h1 = (a,enq(0))(b,sayHi)(b,ret)(a,ret)(c,deq)(c,ret(0)) ✓

[LinR]

h2 = (a,enq(0))(a,ret)(b,sayHi)(b,ret)(c,deq)(c,ret(0))

Relation LinR

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

h1'' = (b,sayHi)(b,ret)(a,enq(0))(a,ret)(c,deq)(c,ret(0)) ✗

h1' = (a,enq(0))(a,ret)(b,sayHi)(b,ret)(c,deq)(c,ret(4)) ✗

h1 = (a,enq(0))(b,sayHi)(b,ret)(a,ret)(c,deq)(c,ret(0)) ✓

[LinR]

h2 = (a,enq(0))(a,ret)(b,sayHi)(b,ret)(c,deq)(c,ret(0))

Relation LinR

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

h1'' = (b,sayHi)(b,ret)(a,enq(0))(a,ret)(c,deq)(c,ret(0)) ✗

h1' = (a,enq(0))(a,ret)(b,sayHi)(b,ret)(c,deq)(c,ret(4)) ✗

h1 = (a,enq(0))(b,sayHi)(b,ret)(a,ret)(c,deq)(c,ret(0)) ✓

[LinR]

h2 = (a,enq(0))(a,ret)(b,sayHi)(b,ret)(c,deq)(c,ret(0))

Relation LinR

- h1[LinR]h2 holds iff h2 is a rearrangement of h1 s.t.

  1. proj(h1,a) = proj(h2,a) for all thread-ids a, and

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

- Example:

  h1'' = (b,sayHi)(b,ret)(a,enq(0))(a,ret)(c,deq)(c,ret(4)) ✗

  h1' = (a,enq(0))(a,ret)(b,sayHi)(b,ret)(c,deq)(c,ret(4)) ✗

  h1 = (a,enq(0))(b,sayHi)(b,ret)(a,ret)(c,deq)(c,ret(0)) ✓

[LinR]

h2 = (a,enq(0))(a,ret)(b,sayHi)(b,ret)(c,deq)(c,ret(0))

Our Results

History viewed as Abstraction

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

History viewed as Abstraction

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

mean((a,enq(0))(a,ret)(b,sayHi)(b,ret)) = {

History viewed as Abstraction

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

mean((a,enq(0))(a,ret)(b,sayHi)(b,ret)) = {

(a,enq(0))(a,ret)(b,sayHi)(a,x:=8)(b,ret),

History viewed as Abstraction

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

mean((a,enq(0))(a,ret)(b,sayHi)(b,ret)) = {

(a,enq(0))(a,ret)(b,sayHi)(a,x:=8)(b,ret),

(a,enq(0))(b,y:=0)(a,ret)(b,sayHi)(a,x:=8)(b,ret),

History viewed as Abstraction

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

mean((a,enq(0))(a,ret)(b,sayHi)(b,ret)) = {

(a,enq(0))(a,ret)(b,sayHi)(a,x:=8)(b,ret),

(a,enq(0))(b,y:=0)(a,ret)(b,sayHi)(a,x:=8)(b,ret),

(a,enq(0))(a,ret)(a,x:=8)(b,assume(x=8))(b,sayHi)(b,ret),

...}

History viewed as Abstraction

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

mean((a,enq(0))(a,ret)(b,sayHi)(b,ret)) = {

(a,enq(0))(a,ret)(b,sayHi)(a,x:=8)(b,ret),

(a,enq(0))(b,y:=0)(a,ret)(b,sayHi)(a,x:=8)(b,ret),

(a,enq(0))(a,ret)(a,x:=8)(b,assume(x=8))(b,sayHi)(b,ret),

...}

- (a,enq(0))(a,x:=0)(a,ret)(b,sayHi)(b,ret) is not in the set.

Main Result I

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

HappenBefore(i,j,h) iff ∃t∈mean(h). Depend(i,j,t).

- Says "HappenBefore" is an abstraction of dependency.
- only-if by example: h = (a,enq(0))(a,ret)(b,sayHi)(b,ret)

Main Result I

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

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

- Says "HappenBefore" is an abstraction of dependency.
- only-if by example: h = (a,enq(0))(a,ret)(b,sayHi)(b,ret)
  (a,enq(0))(a,ret)(a,x:=1)(b,assume(x=1))(b,sayHi)(b,ret)

Main Result I

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

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

- Says "HappenBefore" is an abstraction of dependency.
- only-if by example: h = (a,enq(0))(a,ret)(b,sayHi)(b,ret)
  (a,enq(0))(a,ret)(a,x:=1)(b,assume(x=1))(b,sayHi)(b,ret)
- if by example: h = (a,enq(0))(b,sayHi)(a,ret)(b,ret)

Main Result I

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

HappenBefore(i,j,h) iff ∃t∈mean(h). Depend(i,j,t).

- Says "HappenBefore" is an abstraction of dependency.
- only-if by example: h = (a,enq(0))(a,ret)(b,sayHi)(b,ret)
  (a,enq(0))(a,ret)(a,x:=1)(b,assume(x=1))(b,sayHi)(b,ret)
- if by example: h = (a,enq(0))(b,sayHi)(a,ret)(b,ret)
  (a,enq(0))(b,sayHi)(a,ret)(a,x:=1)(b,assume(x=1))(b,ret)

Trace Equivalence

- t1 ~ t2 iff t2 can be obtained from t1 by swapping independent actions.
- (a,x:=4)(b,y:=11) ~ (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,enq(0))(a,ret)(b,sayHi) ~ (a,enq(0))(b,sayHi)(a,ret)

Main Result 2

[Theorem 2]

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

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

h1 = (a,enq(0))(b,sayHi)(b,ret)(a,ret)

t1 = (a,enq(0))(b,sayHi)(b,ret)(b,x:=1)(a,ret)

h2 = (a,enq(0))(a,ret)(b,sayHi)(b,ret)

Main Result 2

[Theorem 2]

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

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

h1 = (a,enq(0))(b,sayHi)(b,ret)(a,ret)

t1 = (a,enq(0))(b,sayHi)(b,ret)(b,x:=1)(a,ret)

h2 = (a,enq(0))(a,ret)(b,sayHi)(b,ret)

t2 = (a,enq(0))(a,ret)(b,sayHi)(b,ret)(b,x:=1)

Main Result 2

[Theorem 2]

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

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

- if by example:

h1 = (b,sayHi)(b,ret)(a,enq(0))(a,ret)

t1 = (b,sayHi)(b,ret)(b,x:=1)(a,assume(x=1))(a,enq(0))(a,ret)

h2 = (a,enq(0))(a,ret)(b,sayHi)(b,ret)

Main Result 2

[Theorem 2]

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

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

- if by example:

h1 = (b,sayHi)(b,ret)(a,enq(0))(a,ret)

t1 = (b,sayHi)(b,ret)(b,x:=1)(a,assume(x=1))(a,enq(0))(a,ret)

h2 = (a,enq(0))(a,ret)(b,sayHi)(b,ret)

t2 = (a,assume(x=1))(a,enq(0))(a,ret)(b,sayHi)(b,ret)(b,x:=1)

Main Result 2

[Theorem 2]

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

[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`?