Preserving Hyperproperties when using Concurrent Objects

Hagit Attiya (Technion)
Constantin Enea (Ecole Polytechnique)
Jennifer Welch (Texas A&M University)
Abstraction

Program P

Abstract object
E.g., Using an Abstract Multi-Writer Register

Program P

Multi-writer Register
Implemented from
Single-Writer Registers

Program P

Write(v,X)
read TS_0,..., read TS_{n-1}
TS_i = max TS_j + 1
write ⟨v,TS_i,i⟩ to R_i

Read(X)
read R_0,..., read R_{n-1}
return v_j with
   maximal <TS_j,j>

[Vitanyi, Awerbuch]
Or in Message-Passing

Program P

Write(v)
bcast ("query") & wait
  for > n/2 replies
t = largest timestamp
bcast ("update",v,t+1) & wait
  for > n/2 acks

Read()
bcast ("query") & wait
  for > n/2 replies
(v,t) = pair with largest t
bcast ("update",v,t) & wait
  for > n/2 acks
return v

[A, Bar-Noy, Dolev]
[Lynch, Schwarzmann]
When the abstract object is sequential, \( \equiv \) linearizability

Refinement (Trace Inclusion)

\[ \text{Obj} \leq \text{Spec} \iff \forall \text{ program } P, \; \text{Traces}(P \times \text{Obj}) \subseteq \text{Traces}(P \times \text{Spec}) \]
Refinement Preserves Trace Properties

\[ \text{Obj} \leq \text{Spec} \iff \forall \text{ program } P, \quad \text{Traces}(P \times \text{Obj}) \subseteq \text{Traces}(P \times \text{Spec}) \]

If \( \varphi \) is a property of a trace, it is preserved when the atomic object is replaced with a linearizable implementation.

\[ \models \varphi \]
Prove refinement by relating states of abstract and concrete objects

Forward simulations $\equiv$ proofs based on explicit linearization points, e.g., universal constructions using consensus objects or Compare&Swap
Prove refinement by relating states of **abstract** and **concrete** objects

In some cases, find an after-the-fact relation (e.g., based on timestamps)

**Linearizability can always be proved with forward & backward simulation**

[Lynch, Vaandrager]
Hyper (Safety) Properties

Security policies: e.g., noninterference (high clearance values cannot be observed by low clearance users)

Quantitative properties: termination w.h.p., mean response time, probability distributions

Hyperproperties are properties of sets of traces

Hyper safety properties are properties of sets of finite traces
Hyperproperties vs. Refinement

Refinement does not preserve hyperproperties
[McLean 1994]

Linearizability does not preserve probability distributions under strong / weak adversaries
[Golab, Higham, Woelfel, STOC 2011]
Example w/ MWSR Register

Initially $R = \perp$, $C = -1$

**Code for $p_0$, $p_1$:**

$R \leftarrow i$

if $i = 0$ then $C \leftarrow \text{flip 0 or 1}$

**Code for $p_2$:**

$r \leftarrow R$; $c \leftarrow C$

if $(c = 0 \land r = \perp) \lor (c = 1 \land r \neq \perp)$

then loop forever

else terminate

$p_2$ terminates with probability $> \frac{1}{2}$ w/ atomic mwsr register

Example by Noa Schiller
Example w/ MWSR Register

\[ p_0 \]
\[ p_1 \]
\[ p_2 \]

Initially \( R = \perp, \ C = -1 \)

Code for \( p_0, p_1 \):
\[ R \leftarrow i \]
\[ \text{if } i = 0 \text{ then } C \leftarrow \text{flip 0 or 1} \]

Code for \( p_2 \):
\[ r \leftarrow R; \ c \leftarrow C \]
\[ \text{if } (c = 0 \land r = \perp) \lor (c = 1 \land r \neq \perp) \]
\[ \text{then loop forever} \]
\[ \text{else terminate} \]

\( p_2 \) terminates with probability \( > \frac{1}{2} \) w/ atomic mwsr register
Example w/ MWSR Register

Initially $R = \perp$, $C = -1$

**Code for $p_0$, $p_1$:**

$R \leftarrow i$

if $i = 0$ then $C \leftarrow$ flip 0 or 1

**Code for $p_2$:**

$r \leftarrow R; c \leftarrow C$

if $(c = 0 \land r = \perp) \lor (c = 1 \land r \neq \perp)$ then loop forever
else terminate

$p_2$ terminates with probability $> \frac{1}{2}$ w/ atomic mwsr register
Using VA Implementation

Write \( \langle v, R \rangle \)
read \( TS_0, \ldots, read \ TS_{n-1} \)
\( TS_i = \max TS_j + 1 \)
write \( \langle v, TS_i, i \rangle \) to \( R_i \)

Read \( \langle R \rangle \)
read \( R_0, \ldots, read \ R_{n-1} \)
return \( v_j \) with maximal \( TS_j, j \)

Initially \( R = \bot, C = -1 \)

Code for \( p_0, p_1 \):
\( R \leftarrow i \)
if \( i = 0 \) then \( C \leftarrow \) flip \( 0 \) or \( 1 \)

Code for \( p_2 \):
\( r \leftarrow R; c \leftarrow C \)
if \( (c = 0 \land r = \bot) \lor (c = 1 \land r \neq \bot) \)
then loop forever
else terminate

A strong adversary can make \( p_2 \) always loop forever
Using VA Implementation

\[ p_0 \quad \text{Write } R \quad R_0 \leftarrow \langle 0,1,0 \rangle \]
\[ p_1 \quad \text{Write } R \]
\[ p_2 \quad \text{Read } R \quad \langle \bot,0,0 \rangle \leftarrow R_0 \]

**Write**\( (v,R) \)
- read \( TS_0, \ldots, read \ TS_{n-1} \)
- \( TS_i = \max TS_j + 1 \)
- write \( \langle v,TS_i,i \rangle \) to \( R_i \)

**Read**\( (R) \)
- read \( R_0, \ldots, read \ R_{n-1} \)
- return \( v_j \) with maximal \( \langle TS_j,j \rangle \)

**Code for** \( p_0, p_1 \):

- Initially \( R = \bot, C = -1 \)
- \( R \leftarrow i \)
- if \( i = 0 \) then \( C \leftarrow \) flip \( 0 \) or \( 1 \)

**Code for** \( p_2 \):

- \( r \leftarrow R; c \leftarrow C \)
- if \( (c = 0 \land r = \bot) \lor (c = 1 \land r \neq \bot) \)
- then loop forever
- else terminate

A strong adversary can make \( p_2 \) **always loop forever**
A strong adversary can make $p_2$ always loop forever.

**Example w/ VA: Coin = 1**

- **Write $v, R$**
  - read $T S_0, ..., read T S_{n-1}$
  - $T S_i = \text{max } T S_j + 1$
  - write $\langle v, T S_i, i \rangle$ to $R_i$

- **Read $R$**
  - read $R_0, ..., read R_{n-1}$
  - return $v_j$ with maximal $\langle T S_j, j \rangle$

**Code for $p_0, p_1$:**
- $R \leftarrow i$
- if $i = 0$ then $C \leftarrow \text{flip 0 or 1}$

**Code for $p_2$:**
- $r \leftarrow R; c \leftarrow C$
- if $(c = 0 \land r = \bot) \lor (c = 1 \land r \neq \bot)$
  - then loop forever
- else terminate

Initially $R = \bot, C = -1$
Example w/ VA: Coin = 1

A strong adversary can make $p_2$ always loop forever

Write $v, R$
- read $T_{S_0}, ..., read T_{S_{n-1}}$
- $T_{S_i} = \max T_{S_j} + 1$
- write $\langle v, T_{S_i}, i \rangle$ to $R_i$

Read($R$)
- read $R_0, ..., read R_{n-1}$
- return $v_j$ with maximal $\langle T_{S_j}, j \rangle$

Initially $R = \bot, C = -1$

Code for $p_0, p_1$:
- $R \leftarrow i$
  - if $i = 0$ then $C \leftarrow$ flip $0$ or $1$

Code for $p_2$:
- $r \leftarrow R; c \leftarrow C$
  - if $(c = 0 \wedge r = \bot) \vee (c = 1 \wedge r \neq \bot)$
    - then loop forever
  - else terminate
A strong adversary can make $p_2$ always loop forever.
Example w/ VA: Coin = 0

Write \( R \)
- \( R_0 \leftarrow \langle 0,1,0 \rangle \)
- \( R \leftarrow \langle 1,2,1 \rangle \leftarrow R_1 \)

Read \( R \)
- \( \langle \perp, 0,0 \rangle \leftarrow R_0 \)
- \( \langle \perp, 0,1 \rangle \leftarrow R_1 \)

Write \((v, R)\)
- read \( T S_0, ..., read \ T S_{n-1} \)
- \( T S_i = \max \ T S_j + 1 \)
- write \( \langle v, T S_i, i \rangle \) to \( R_i \)

Read \((R)\)
- read \( R_0, ..., read \ R_{n-1} \)
- return \( v_j \) with maximal \( <T S_j, j> \)

Initially \( R = \perp, C = -1 \)

Code for \( p_0, p_1 \):
- \( R \leftarrow i \)
- if \( i = 0 \) then \( C \leftarrow \text{flip 0 or 1} \)

Code for \( p_2 \):
- \( r \leftarrow R; c \leftarrow C \)
- if \( (c = 0 \wedge r = \perp) \vee (c = 1 \wedge r \neq \perp) \)
  then loop forever
- else terminate

A strong adversary can make \( p_2 \) always loop forever
Another Example

When R is an atomic register, p₂ terminates with probability > ½

Initially: R = ⊥, C = −1

Code for pi, i = 0, 1:
R ← i
if (i == 1) then C ← flip fair coin (0 or 1)

Code for p₂:
u1 ← R; u2 ← R; c ← C
if ((u1 ≠ c) or (u2 ≠ 1 − c)) then loop forever
else terminate

[A, Enea, Welch, arxiv 2106.15554]
Distilled from [Hadzilacos, Hu, Toueg, PODC 2021]
Example w/ ABD

When R is implemented in message-passing, a strong adversary can make \( p_2 \) always loop forever.

- **Write(v)**
  - `bcast ⟨"query"⟩` and wait for \( > n/2 \) replies \((v, t)\)
  - \( t \) = largest timestamp
  - `bcast ⟨"update",v,t+1⟩` and wait for \( > n/2 \) acks

- **Read()**
  - `bcast ⟨"query"⟩` and wait for \( > n/2 \) replies \((v, t)\) = pair with largest timestamp
  - `bcast ⟨"update",v,t⟩` and wait for \( > n/2 \) acks
  - return \( v \)

Initially: \( R = ⊥, C = −1 \)

**Code for \( p_i \), \( i = 0, 1 \):**
- \( R \leftarrow i \)
- if \( i == 1 \) then \( C \leftarrow \) flip fair coin (0 or 1)

**Code for \( p_2 \):**
- \( u1 \leftarrow R; u2 \leftarrow R; c \leftarrow C \)
- if \( ((u1 ≠ c) \) or \( (u2 ≠ 1 − c) \)) then loop forever
- else terminate

[A, Enea, Welch, arxiv 2106.15554]
Distilled from [Hadzilacos, Hu, Toueg, PODC 2021]
Strong Linearizability

Linearization points are **prefix preserving**

Preserves probability distributions under strong adversaries

[Golab, Higham, Woelfel, STOC 2011]
More Generally, Strong Refinement

\[ \text{Obj} \preceq \text{Spec} \iff \forall \text{ program } P, \forall \text{ deterministic scheduler } S_1 \text{ of } P \times \text{Obj}, \]
\[ \exists \text{ deterministic scheduler } S_2 \text{ of } P \times \text{Spec}, \]
\[ \text{Traces}(P \times \text{Obj} \times S_1) = \text{Traces}(P \times \text{Spec} \times S_2) \]

Preserves Hyperproperties

\[ \equiv \text{Forward Simulation} \]

[\text{A, Enea, DISC 2019}]
$\Rightarrow$ Strong Refinement
Composes

Locality (horizontal composition)

Parametrized objects (hierarchical composition)
Many Objects Don’t Have Strongly Linearizable Implementations

Counter-example $\Rightarrow$ V&A MW register is not strongly linearizable

In fact, there is no \textit{wait-free strongly-linearizable} MW register implementation from SW registers

Also, no \textit{wait-free strongly-linearizable} snapshot implementation

[Helmi, Higham, Woelfel, PODC 2012]

No message-passing simulation of a register
Given a **SL message-passing** implementation of a **multi-writer multi-reader register**

\( n \) processes, \( f << n \) possible failures

Obtain a **SL shared-memory** implementation of a **multi-writer multi-reader register**, using only **single-writer multi-reader registers**

\( f+1 \) processes, \( f \) possible failures

(Strong simulation)

Which is impossible by [Helmi et al. 2012]

[A, Enea, Welch, DISC 2021]

A direct proof in [Chan, Hadzilacos, Hu, Toueg, arxiv 2108.01651]
Take inspiration from program indistinguishable obfuscation [Barak et al. 2012] and oblivious RAMs [Goldreich and Ostrovsky, 1996]

Perturb the concrete object to blunt the adversary, while keeping its functionality indistinguishable

Use randomization...
Use Randomization to Blunt the Adversary

Implement MW register from SW registers

Atomic MW

P

termination with high probability
E.g., Blunting w/ One Coin Flip

\( p_2 \) terminates with constant probability with \( VA^2 \)

**Write**(v,X)
read TS\(_0\),...,read TS\(_{n-1}\)
TS\(_i\) = max TS\(_j\) +1
write \( \langle v,TS_i,i \rangle \) to Ri

**Read**(X)
read R\(_0\),...,read R\(_{n-1}\)
v\(_1\) = v\(_j\) with maximal \( \langle TS_j,j \rangle \)
read R\(_0\),...,read R\(_{n-1}\)
v\(_2\) = v\(_j\) with maximal \( \langle TS_j,j \rangle \)
return v\(_1\) or v\(_2\) with prob. \( \frac{1}{2} \)

Initially \( R = \perp \), \( C = -1 \)

**Code for** \( p_0, p_1 \):
\( R \leftarrow i \)
if \( i = 0 \) then \( C \leftarrow \) flip 0 or 1

**Code for** \( p_2 \):
\( r \leftarrow R; \ c \leftarrow C \)
if \( (c = 0 \land r = \perp) \lor (c = 1 \land r \neq \perp) \)
then loop forever
else terminate
Tail Strong Linearizability

Identify a **preamble** of the operation, after which, it is mapped in a prefix-preserving manner (“**strongly linearizable**”)

**Write**(v,X)
read TS₀,..., read TSₙ₋₁
TSᵢ = max TSⱼ + 1
write ⟨v, TSᵢ, i⟩ to Rᵢ

**Read**(X)
read R₀,..., read Rₙ₋₁
v = vⱼ with maximal ⟨TSⱼ, j⟩

return v

**Effect-free** preamble doesn’t impact concurrently-running processes
Blunting

Tail strongly linearizable objects with an effect-free preamble

- Repeat the preamble $k$ times
- Randomly pick one of the iterations to continue with

\[
\text{Write}(v,X) \\
\begin{array}{l}
\text{read } TS_0, \ldots, \text{read } TS_{n-1} \\
TS_i = \max TS_j + 1 \\
\text{write } \langle v, TS_i, i \rangle \text{ to } R_i
\end{array}
\]

\[
\text{Read}(X) \\
\begin{array}{l}
\text{read } R_0, \ldots, \text{read } R_{n-1} \\
v1 = v_j \text{ with maximal } <TS_j, j> \\
\text{read } R_0, \ldots, \text{read } R_{n-1} \\
v2 = v_j \text{ with maximal } <TS_j, j> \\
\text{return } v1 \text{ or } v2 \text{ with prob. } \frac{1}{2}
\end{array}
\]
Blunting, Specifically

Tail strongly linearizable objects with a read-only preamble, e.g.,

- Multi-reader registers from single-reader registers [Israeli, Li 1993]
- Multi-writer registers from single-writer registers [Vitanyi, Awerbuch 1986]
- ABD, Snapshots [Afek et al.]

For an n-process program \( P \) with \( r \) coin flips, using tail-strongly-linearizable objects \( O \) with effect-free preambles & any \( k \geq r \),

\[
\Pr[O^k] \leq \Pr[O_a] + (\Pr[O] - \Pr[O_a]) \left( 1 - \left( \frac{k - r}{k} \right)^{n-1} \right)
\]
E.g., in Our Example

\( p_2 \) terminates with probability greater than \( \frac{1}{8} \) with \( \text{VA}^2 \) and greater than \( \frac{2}{9} \) with \( \text{VA}^3 \)

\[
\begin{array}{c|c}
\text{Non-termination probability w/ } \text{VA}^2 & \frac{1}{2} \\
\hline
\text{Pr}[O^k] \leq \text{Pr}[O_a] + (\text{Pr}[O] - \text{Pr}[O_a]) \left(1 - \left(\frac{k - r}{k}\right)^{n-1}\right)
\end{array}
\]

\[
\begin{align*}
1 - \left(\frac{3-1}{3}\right)^2 &= \frac{5}{9} \\
1 - \left(\frac{2-1}{2}\right)^2 &= \frac{3}{4}
\end{align*}
\]

- probability of a **bad** outcome \( B \) when \( P \) uses \( k \)-preamble-iterated versions of objects in \( O \)
- probability of \( B \) when \( P \) uses **atomic** versions of objects in \( O \)
- probability of \( B \) when \( P \) uses objects in \( O \)
Let $X$ be the event that all random choices in $O^k$ objects return an iteration of the preamble that does NOT overlap any random step of the program, then

$$
\Pr[O^k] = \Pr[O^k|X] \cdot \Pr[X] + \Pr[O^k|\neg X] \cdot (1 - \Pr[X])
$$

$$
\leq \Pr[O_a] \cdot \Pr[X]
$$

when chosen preambles don’t overlap any program random steps, $O^k$ objects behave like atomic objects

when chosen preamble overlaps program random step, $O^k$ objects are no worse than $O$ objects

$$
+ \Pr[O] \cdot (1 - \Pr[X])
$$

Since $\Pr[X] \geq \left(\frac{k-r}{k}\right)^{n-1}$, rearrangement gives that

$$
\Pr[O^k] \leq \Pr[O_a] + (\Pr[O] - \Pr[O_a]) \left(1 - \left(\frac{k-r}{k}\right)^{n-1}\right)
$$

probability of a bad outcome $B$ when $P$ uses $k$-preamble-iterated versions of objects in $O$

probability of $B$ when $P$ uses atomic versions of objects in $O$

probability of $B$ when $P$ uses objects in $O$
Wrap-Up

Write strong linearizability does not help with our example

Tradeoff between # iterations and decreased prob. of bad outcome
Reduce # random steps considered in the analysis, based on program structure (e.g., communication-closed layers)

Object implementations w/o effect-free preambles

Transactions & Cryptographic protocols
References

• Attiya, Enea, Welch: Blunting an Adversary Against Randomized Concurrent Programs with Linearizable Implementations. PODC 2022

• Attiya, Enea, Welch: Impossibility of Strongly-Linearizable Message-Passing Objects via Simulation by Single-Writer Registers. DISC 2021

• Attiya, Enea: Putting Strong Linearizability in Context: Preserving Hyperproperties in Programs that Use Concurrent Objects. DISC 2019