Blunting an Adversary Against Randomized Concurrent Programs with Linearizable Implementations

Hagit Attiya (Technion)
Constantin Enea (Ecole Polytechnique)
Jennifer Welch (Texas A&M University)
Using an Abstract Multi-Writer Register
Write\((v, X)\)
read \(TS_0, \ldots, \text{read } TS_{n-1}\)
\(TS_i = \max TS_j + 1\)
write \(\langle v, TS_i, i \rangle\) to \(R_i\)

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

[\text{Vitanyi, Awerbuch}]
Linearizability Preserves Trace Properties

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

[Herlihy, Wing]
Linearizability does not preserve properties of sets of traces. E.g., probability distributions.

[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 $C \leftarrow$ 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, based on [A, Enea, Welch] and [Hadzilacos, Hu, Toueg, PODC 2021]
Example w/ MWSR Register

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

Code for $p_0, p_1$:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

A strong adversary can make $p_2$ always loop forever
Example w/ VA: Coin = 1

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

Write(v,R)
\[
\text{Read(TS}_0\ldots, \text{read TS}_{n-1}
\]
\[
\text{TS}_i = \max \text{TS}_j + 1
\]
write \( \langle v, \text{TS}_i, i \rangle \) to \( R_i \)

Read(R)
\[
\text{Read R}_0\ldots, \text{read R}_{n-1}
\]
return \( v_j \) with maximal \( \langle \text{TS}_j, j \rangle \)

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

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 \( (c = 0 \land r = \bot) \lor (c = 1 \land r \neq \bot) \) then loop forever
else terminate
A strong adversary can make $p_2$ always loop forever.
Example w/ VA: Coin = 0

A strong adversary can make p₂ always loop forever

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

Read(R)
read R₀,...,read Rₙ₋₁
return vⱼ with maximal ⟨TSⱼ,j⟩

Initially R =⊥, C = -1
Code for p₀, p₁:
R ← i
if i = 0 then C ← flip 0 or 1

Code for p₂:
r ← R; c ← C
if (c = 0 ∧ r =⊥) ∨ (c = 1 ∧ r ≠⊥)
then loop forever
else terminate
Example w/ VA: Coin = 0

Write(v,R)
read TS₀,...,read TSₙ₋₁
TSᵢ = max TSⱼ +1
write \( \langle v,TSᵢ,i \rangle \) to Rᵢ

Read(R)
read R₀,...,read Rₙ₋₁
return vᵢ with maximal \( <TSⱼ,j> \)

Initially \( R = \bot, C = -1 \)
Code for \( p₀, p₁ \):
\( R \leftarrow i \)
if \( i = 0 \) then \( C \leftarrow \) flip 0 or 1

Code for \( p₂ \):
\( r \leftarrow R; c \leftarrow C \)
if \( (c = 0 \land r = \bot) \lor (c = 1 \land r \neq \bot) \)
then loop forever
else terminate

A strong adversary can make \( p₂ \) always loop forever
Strong Linearizability

Linearization points are prefix preserving.

Preserves probability distributions under strong adversaries.

[Golab, Higham, Woelfel, STOC 2011]
Many Objects Don’t Have Strongly Linearizable Implementations

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

In fact, there is no wait-free strongly-linearizable MW register implementation from SW registers
Also, no wait-free strongly-linearizable snapshot implementation

[Helmi, Higham, Woelfel, PODC 2012]

No message-passing simulation of a register

[A, Enea, Welch, DISC 2021] [Chan, Hadzilacos, Hu, Toueg]
Use Randomization to Blunt the Adversary

Implement MW register from SW registers

Atomic MW

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

\( p_2 \) terminates with constant probability with \( \text{VA}^2 \)

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

**Read**(\( X \))
- read \( R_0, \ldots, \text{read } R_{n-1} \)
- \( v_1 = \text{v}_j \) with maximal \( \langle TS_j, j \rangle \)
- read \( R_0, \ldots, \text{read } R_{n-1} \)
- \( v_2 = \text{v}_j \) with maximal \( \langle TS_j, j \rangle \)
- return \( v_1 \) or \( v_2 \) with prob. \( \frac{1}{2} \)

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

**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 \( (c = 0 \land r = \bot) \lor (c = 1 \land r \neq \bot) \) 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")

\[
\begin{array}{l}
\text{Write}(v,X) \\
\text{read } T_{S_0},...,\text{read } T_{S_n-1} \\
T_{S_i} = \text{max } T_{S_j} + 1 \\
\text{write } \langle v, T_{S_i}, i \rangle \text{ to } R_i \\
\text{Read}(X) \\
\text{read } R_0,...,\text{read } R_{n-1} \\
v = v_j \text{ with maximal } \langle T_{S_j}, j \rangle \\
\end{array}
\]

**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

```
<table>
<thead>
<tr>
<th>Write(v,X)</th>
</tr>
</thead>
<tbody>
<tr>
<td>read TS_0, ..., read TS_{n-1}</td>
</tr>
<tr>
<td>TS_i = max TS_j + 1</td>
</tr>
<tr>
<td>write \langle v, TS_i, i \rangle to R_i</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Read(X)</th>
</tr>
</thead>
<tbody>
<tr>
<td>read R_0, ..., read R_{n-1}</td>
</tr>
<tr>
<td>v1 = v_j with maximal \langle TS_j, j \rangle</td>
</tr>
<tr>
<td>read R_0, ..., read R_{n-1}</td>
</tr>
<tr>
<td>v2 = v_j with maximal \langle TS_j, j \rangle</td>
</tr>
<tr>
<td>return v1 or v2 with prob. \frac{1}{2}</td>
</tr>
</tbody>
</table>
```
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 $k \geq r$,

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

- 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 $O$
E.g., in our Example

\[ p_2 \text{ terminates with probability } > \frac{1}{8} \text{ with \ } \text{VA}^2 \]

and \( > \frac{2}{9} \text{ with \ } \text{VA}^3 \)

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

- 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 \( O \)

\[
1 - \left(\frac{3 - 1}{3}\right)^2 = \frac{5}{9}
\]

\[
1 - \left(\frac{2 - 1}{2}\right)^2 = \frac{3}{4}
\]
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

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

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

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

[Hadzilacos, Hu, Toueg, PODC 2021]

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