On Parallel Snapshot Isolation and Release/Acquire Consistency

Azalea Raad       Ori Lahav       Viktor Vafeiadis

MPI-SWS
Tel Aviv University

Thursday 19 April
ESOP 2018
Thessaloniki, Greece
What is STM?

Concurrency Control via *transactions*

- **atomic** unit of work (set of operations) on shared data
- **all-or-nothing**

\[
\begin{align*}
\text{T:} & \begin{bmatrix}
\text{x := 1;} \\
\text{y := 1;}
\end{bmatrix} \\
\text{// x = y = 0 OR x = y = 1}
\end{align*}
\]
Which STM?

**Strong** consistency - *inefficient*

- serialisability
- strong serialisability
- …

**Weak** consistency

- snapshot isolation (SI)
- parallel snapshot isolation (PSI)
- …
Which STM?

**Strong** consistency - inefficient

- serialisability
- strong serialisability
- ...

**Weak** consistency

- snapshot isolation (SI)
  - **parallel snapshot isolation (PSI)**
    - efficient, monotonic
- ...

STM Context

- Shared memory setting
  (with weak memory consistency)

- *Mixed* accesses to shared data
  (transactional and non-transactional)

- *Cannot instrument* non-transactional accesses
  (weak isolation)
PSI STM Desiderata
PSI STM Desiderata

- *Declarative* semantics

- Reference implementation (*operational* semantics)
  - **Sound**: $\text{Behaviours(imp)} \subseteq \text{Behaviours(spec)}$
  - **Complete**: $\text{Behaviours(spec)} \subseteq \text{Behaviours(imp)}$
PSI STM Desiderata

- Declarative semantics

- Reference implementation (operational semantics)
  - **Sound**: $\text{Behaviours(imp)} \subseteq \text{Behaviours(spec)}$
  - **Complete**: $\text{Behaviours(spec)} \subseteq \text{Behaviours(imp)}$

- Declarative semantics with mixed accesses

- Reference implementation with mixed accesses (operational semantics)
  - **Sound**: $\text{Behaviours(imp)} \subseteq \text{Behaviours(spec)}$
  - **Complete**: $\text{Behaviours(spec)} \subseteq \text{Behaviours(imp)}$
Declarative semantics

Reference implementation (operational semantics)
- **Sound**: \( \text{Behaviours(imp)} \subseteq \text{Behaviours(spec)} \)
- **Complete**: \( \text{Behaviours(spec)} \subseteq \text{Behaviours(imp)} \)

Declarative semantics with *mixed* accesses

Reference implementation with *mixed* accesses (operational semantics)
- **Sound**: \( \text{Behaviours(imp)} \subseteq \text{Behaviours(spec)} \)
- **Complete**: \( \text{Behaviours(spec)} \subseteq \text{Behaviours(imp)} \)
PSI STM Desiderata

✓ Declarative semantics

✓ Reference implementation (operational semantics)

→ **Sound**: Behaviours(imp) ⊆ Behaviours(spec)
→ **Complete**: Behaviours(spec) ⊆ Behaviours(imp)

✓ Declarative semantics with *mixed* accesses  ⇒  **RPSI** (‘robust’ PSI)

✓ Reference implementation with *mixed* accesses (operational semantics)

→ **Sound**: Behaviours(imp) ⊆ Behaviours(spec)
→ **Complete**: Behaviours(spec) ⊆ Behaviours(imp)
PSI STM Desiderata

✓ Declarative semantics

✓ Reference implementation (operational semantics)

→ Sound: Behaviours(imp) ≤ Behaviours(spec)
→ Complete: Behaviours(spec) ≤ Behaviours(imp)

✓ Declarative semantics with mixed accesses ⇒ RPSI (‘robust’ PSI)

✓ Reference implementation with mixed accesses (operational semantics)

→ Sound: Behaviours(imp) ≤ Behaviours(spec)
→ Complete: Behaviours(spec) ≤ Behaviours(imp)
What is PSI?

\( r_1 \)

\[ x = y = 0 \]

\( r_2 \)

\[ x = y = 0 \]

\( r_3 \)

\[ x = y = 0 \]
What is PSI?

r1

\[ x := 1; \]

r2

x = y = 0

r3

x = y = 0
What is PSI?

r1: \( x = y = 0 \)

r2: \( x = y = 0 \)

r3: \( x = y = 0 \)

S1: \( x = y = 0 \)

\[
\begin{bmatrix}
  x := 1;
\end{bmatrix}
\]
What is PSI?

\begin{align*}
  &x = y = 0 \\
  &x = y = 0 \\
  &x = y = 0 \\
  &x = y = 0 \\
\end{align*}

\begin{align*}
  &S_1: x = y = 0 \\
  &\begin{bmatrix}
    x := 1;
  \end{bmatrix} \\
  &C_1: x = 1
\end{align*}
What is PSI?

**r1**

x = y = 0

**r2**

x = y = 0

**r3**

x = y = 0

**S1:** x = y = 0

\[
x := 1;
\]

**C1:** x = 1

ww-conflict? no
What is PSI?

\[ x := 1; \]

\[ \begin{align*}
S1: & \quad x = y = 0 \\
C1: & \quad x = 1
\end{align*} \]

ww-conflict? no
What is PSI?

\[ x := 1; \]

**S1:** \( x = y = 0 \)

**C1:** \( x = 1 \)

- **r1:** \( x = 1; y = 0 \)
- **r2:** \( x = 1; y = 0 \)
- **r3:** \( x = 1; y = 0 \)
PSI Litmus Tests

Write skew / SB

T1:
\[
\begin{align*}
x & := 1; \\
a & := y; \quad \text{// 0}
\end{align*}
\]

T2:
\[
\begin{align*}
y & := 1; \\
b & := x; \quad \text{// 0}
\end{align*}
\]

All variables are initially 0; comments specify the values read.
PSI Litmus Tests

Write skew / SB

\[ T1: \begin{align*} x &:= 1; \\ a &:= y; \quad // 0 \end{align*} \quad | \quad T2: \begin{align*} y &:= 1; \\ b &:= x; \quad // 0 \end{align*} \]

\[ r1 \quad x=y=0 \]

\[ r2 \quad x=y=0 \]
PSI Litmus Tests

Write skew / SB

T1:
\[ \begin{align*}
x &:= 1; \\
a &:= y; \quad \text{// 0}
\end{align*} \]

T2:
\[ \begin{align*}
y &:= 1; \\
b &:= x; \quad \text{// 0}
\end{align*} \]

S1: \[ x = y = 0 \]
\[ \begin{align*}
x &:= 1; \\
a &:= y; \quad \text{// 0}
\end{align*} \]

C1: \( x = 1 \)
PSI Litmus Tests

Write skew / SB

\[
\begin{align*}
T1: & \quad x := 1; \\
& a := y; \quad // 0
\end{align*}
\quad || \quad
\begin{align*}
T2: & \quad y := 1; \\
& b := x; \quad // 0
\end{align*}
\]

**Test 1**

- **State 1 (r1)**: \(x = y = 0\)
- **Statement 1 (S1)**: \(x := 1;\)
  - \(x := 1;\)
  - \(a := y; \quad // 0\)
- **Constraint 1 (C1)**: \(x = 1\)

**Test 2**

- **State 2 (r2)**: \(x = y = 0\)
- **Statement 2 (S2)**: \(y := 1;\)
  - \(y := 1;\)
  - \(b := x; \quad // 0\)
- **Constraint 2 (C2)**: \(y = 1\)
PSI Litmus Tests

Write skew / SB

T1:
\[
\begin{align*}
x & := 1; \\
a & := y; \quad // 0
\end{align*}
\]

T2:
\[
\begin{align*}
y & := 1; \\
b & := x; \quad // 0
\end{align*}
\]

S1: \(x=y=0\)
\[
\begin{align*}
x & := 1; \\
a & := y; \quad // 0
\end{align*}
\]

C1: \(x=1\)

S2: \(x=y=0\)
\[
\begin{align*}
y & := 1; \\
b & := x; \quad // 0
\end{align*}
\]

C2: \(y=1\)

ww-conflict? no  ww-conflict? no
PSI Litmus Tests

Write skew / SB

\[ T1: \]
\[
\begin{align*}
  x &:= 1; \\
  a &:= y; \quad // 0
\end{align*}
\]

\[ T2: \]
\[
\begin{align*}
  y &:= 1; \\
  b &:= x; \quad // 0
\end{align*}
\]

\[ r1 \]
\[
\begin{align*}
  x &= 1; \\
  y &= 1
\end{align*}
\]

\[ \text{S1: } x=y=0 \\
   x := 1; \\
   a := y; \quad // 0 \\
\]

\[ \text{C1: } x=1 \]

\[ \text{ww-conflict? no} \]

\[ r2 \]
\[
\begin{align*}
  x &= 1; \\
  y &= 1
\end{align*}
\]

\[ \text{S2: } x=y=0 \\
   y := 1; \\
   b := x; \quad // 0 \\
\]

\[ \text{C2: } y=1 \]

\[ \text{ww-conflict? no} \]
### PSI Litmus Tests

**Long fork / IRIW**

| T1: | T2: | T3: | T4: |
|-----|-----|-----|-----|
| ![Image](image1.png) | ![Image](image2.png) | ![Image](image3.png) | ![Image](image4.png) |

All variables are initially 0; comments specify the values read.
PSI Litmus Tests

Long fork / IRIW

T1:
\[
\begin{align*}
x & := 1; \\
\end{align*}
\]

T2:
\[
\begin{align*}
a & := x; \quad \text{// 1} \\
b & := y; \quad \text{// 0}
\end{align*}
\]

T3:
\[
\begin{align*}
c & := y; \quad \text{// 1} \\
d & := x; \quad \text{// 0}
\end{align*}
\]

T4:
\[
\begin{align*}
y & := 1;
\end{align*}
\]

r1
\[
x = y = 0
\]

r2
\[
x = y = 0
\]

r3
\[
x = y = 0
\]

r4
\[
x = y = 0
\]

S1:
\[
\begin{align*}
x & := 1; \\
\end{align*}
\]

C1:
\[
\begin{align*}
x & = 1
\end{align*}
\]
PSI Litmus Tests

Long fork / IRIW

**T1:**
\[
\begin{align*}
x & := 1; \\
\end{align*}
\]

**T2:**
\[
\begin{align*}
a & := x; \quad // 1 \\
b & := y; \quad // 0 \\
\end{align*}
\]

**T3:**
\[
\begin{align*}
c & := y; \quad // 1 \\
d & := x; \quad // 0 \\
\end{align*}
\]

**T4:**
\[
\begin{align*}
y & := 1; \\
\end{align*}
\]

**r1**
\[
\begin{align*}
x & = y = 0 \\
\end{align*}
\]

**r2**
\[
\begin{align*}
x & = y = 0 \\
\end{align*}
\]

**r3**
\[
\begin{align*}
x & = y = 0 \\
\end{align*}
\]

**r4**
\[
\begin{align*}
x & = y = 0 \\
\end{align*}
\]

**S1:**
\[
\begin{align*}
x & := 1; \\
\end{align*}
\]

**C1:**
\[
\begin{align*}
x & = 1 \\
\end{align*}
\]

**S4:**
\[
\begin{align*}
y & := 1; \\
\end{align*}
\]

**C4:**
\[
\begin{align*}
y & = 1 \\
\end{align*}
\]
PSI Litmus Tests

Long fork / IRIW

\[ T_1 : \]
\[
x := 1;
\]
\[
x = y = 0
\]

\[ T_2 : \]
\[
a := x; \quad // 1
\]
\[
b := y; \quad // 0
\]

\[ T_3 : \]
\[
c := y; \quad // 1
\]
\[
d := x; \quad // 0
\]

\[ T_4 : \]
\[
y := 1;
\]

\[ r_1 \]
\[
x = y = 0
\]

\[ r_2 \]
\[
x = y = 0
\]

\[ r_3 \]
\[
x = y = 0
\]

\[ r_4 \]
\[
x = y = 0
\]

\[ S_1 : x = y = 0 \]
\[
x := 1;
\]

\[ C_1 : x = 1 \]

ww-conflict? no

\[ S_4 : x = y = 0 \]
\[
y := 1;
\]

\[ C_4 : y = 1 \]

ww-conflict? no
## PSI Litmus Tests

### Long fork / IRIW

| T1: | T2: | T3: | T4: |
|-----|-----|-----|-----|
| \[x := 1;\] | \[a := x; \quad // 1\] | \[c := y; \quad // 1\] | \[y := 1;\] |
| \[b := y; \quad // 0\] | \[d := x; \quad // 0\] |

### Test Cases

- **r1**: \(x = 1; y = 0\)
- **r2**: \(x = y = 0\)
- **r3**: \(x = y = 0\)
- **r4**: \(x = 0; y = 1\)

### Conflicts

- **S1**: \(x = y = 0\)  
  \[x := 1;\]  
  **C1**: \(x = 1\)

  
  - ww-conflict? no

- **S4**: \(x = y = 0\)  
  \[y := 1;\]  
  **C4**: \(y = 1\)

  - ww-conflict? no
PSI Litmus Tests

Long fork / IRIW

\[ T1: \begin{cases} x := 1; \end{cases} \]
\[ T2: \begin{cases} a := x; & 1 \\ b := y; & 0 \end{cases} \]
\[ T3: \begin{cases} c := y; & 1 \\ d := x; & 0 \end{cases} \]
\[ T4: \begin{cases} y := 1; \end{cases} \]

\[ r1 \]
\[ x=1; y=0 \]

\[ r2 \]
\[ x=1; y=0 \]

\[ r3 \]
\[ x=y=0 \]

\[ r4 \]
\[ x=0; y=1 \]

\[ S1: x=y=0 \]
\[ C1: x=1 \]

ww-conflict? no

\[ S4: x=y=0 \]
\[ C4: y=1 \]

ww-conflict? no
PSI Litmus Tests

Long fork / IRIW

\[
\begin{align*}
T1: & \quad \begin{cases}
  x := 1; \\
\end{cases} \\
T2: & \quad \begin{cases}
  a := x; \quad \text{// 1} \\
  b := y; \quad \text{// 0}
\end{cases} \\
T3: & \quad \begin{cases}
  c := y; \quad \text{// 1} \\
  d := x; \quad \text{// 0}
\end{cases} \\
T4: & \quad \begin{cases}
  y := 1;
\end{cases}
\end{align*}
\]

\[\text{ww-conflict? no}\]
PSI Litmus Tests

Long fork / IRIW

T1:
\[ x := 1; \]

T2:
\[ a := x; \quad // 1 \\
    b := y; \quad // 0 \]

T3:
\[ c := y; \quad // 1 \\
    d := x; \quad // 0 \]

T4:
\[ y := 1; \]

r1

x=1; y=0

r2

x=1; y=0

r3

x=0; y=1

r4

x=0; y=1

S1: x=y=0

\[ x := 1; \]

S2: x=1; y=0

C1: x=1

S3: x=0; y=1

S4: x=y=0

\[ y := 1; \]

C2: null

C3: null

C4: y=1

ww-conflict? no

ww-conflict? no
PSI Litmus Tests

Long fork / IRIW

T1:
\[
\begin{align*}
x & := 1; \\
\end{align*}
\]
T2:
\[
\begin{align*}
a & := x; // 1 \\
b & := y; // 0 \\
\end{align*}
\]
T3:
\[
\begin{align*}
c & := y; // 1 \\
d & := x; // 0 \\
\end{align*}
\]
T4:
\[
\begin{align*}
y & := 1; \\
\end{align*}
\]

r1
\[
x = 1; y = 0
\]

r2
\[
x = 1; y = 0
\]

r3
\[
x = 0; y = 1
\]

r4
\[
x = 0; y = 1
\]

S1: \(x=y=0\)
\[
\begin{align*}
x & := 1; \\
\end{align*}
\]

C1: \(x=1\)

ww-conflict? no

S2: \(x=1; y=0\)

C2: null

S3: \(x=0; y=1\)

C3: null

S4: \(x=y=0\)
\[
\begin{align*}
y & := 1; \\
\end{align*}
\]

C4: \(y=1\)

ww-conflict? no
Reasoning about replicas and propagation order is difficult.
Reasoning about replicas and propagation order is difficult

⇒

Lock-based PSI reference implementation in which memory model?
PSI in \textit{which} Memory Model?

- Sequential Consistency (SC)?
PSI in **which** Memory Model?

❌ Sequential Consistency (SC) — too strong!

| Long fork / IRIW |
|------------------|
| **T1:**          |
| \[ x := 1; \]    |
| **T2:**          |
| \[ a := x; // 1 \] |
| \[ b := y; // 0 \] |
| **T3:**          |
| \[ c := y; // 1 \] |
| \[ d := x; // 0 \] |
| **T4:**          |
| \[ y := 1; \]    |
Reasoning about replicas and propagation order is difficult.

⇒

Lock-based
PSI reference implementation in
C11 release/acquire fragment
Which Locks for PSI?
Which Locks for PSI?

❌ Global lock

→ disjoint accesses allowed
Which Locks for PSI?

✗ Global lock
   ➔ disjoint accesses allowed

• Per-location locks
Which Locks for PSI?

❌ **Global** lock
  ➡ disjoint accesses allowed

❌ **Per-location** locks
  ➡ concurrent reads allowed
Which Locks for PSI?

❌ Global lock
  ➡ disjoint accesses allowed

❌ Per-location locks
  ➡ concurrent reads allowed

• Per-location MRSW (multiple-readers-single-writer) locks
Which Locks for PSI?

✗ **Global** lock
   → *disjoint* accesses allowed

✗ **Per-location** locks
   → concurrent reads allowed

✗ **Per-location MRSW** (multiple-readers-single-writer) locks
   → readers should *not* synchronise (e.g. IRIW)

---

Long fork / IRIW

\[
\begin{align*}
\text{T1:} & & \text{T2:} & & \text{T3:} & & \text{T4:} \\
\begin{array}{l}
\begin{align*}
x & := 1; \\
\end{align*}
\end{array}
& & \begin{array}{l}
\begin{align*}
a & := x; \quad \text{◆ 1} \\
b & := y; \quad \text{◆ 0}
\end{align*}
\end{array}
& & \begin{array}{l}
\begin{align*}
c & := y; \quad \text{◆ 1} \\
d & := x; \quad \text{◆ 0}
\end{align*}
\end{array}
& & \begin{array}{l}
\begin{align*}
y & := 1;
\end{align*}
\end{array}
\end{align*}
\]
Which Locks for PSI?

✗ **Global** lock
  ➔ *disjoint* accesses allowed

✗ **Per-location** locks
  ➔ concurrent reads allowed

✗ **Per-location MRSW** (multiple-readers-single-writer) locks
  ➔ readers should *not* synchronise (e.g. IRIW)

✓ **Per-location sequence** locks
Sequence Locks

\texttt{x\_lock\ version\ even} : lock free
\texttt{x\_lock\ version\ odd} : lock taken
Sequence Locks

\texttt{x\_lock version even} : lock free
\texttt{x\_lock version odd} : lock taken

\begin{verbatim}
\textbf{wlock(x)}{
\textbf{retry:}
\texttt{vx := x\_lock;}
\texttt{if (is-odd(vx))}
\texttt{goto retry;}
\texttt{if(!CAS(x\_lock, vx, vx+1))}
\texttt{goto retry;}
}
\textbf{wunlock(x)}{
\texttt{vx := x\_lock;}
\texttt{x\_lock := vx + 1;}
}
\end{verbatim}
Sequence Locks

\[x\text{\_lock version } \text{even} : \text{lock free}\]
\[x\text{\_lock version } \text{odd} : \text{lock taken}\]

\[
\text{snapshot}(x)\{
\text{retry:}
\hspace{1em} \text{// tentative version}
\hspace{2em} v_x := x\text\_lock;
\hspace{2em} \text{while (is-odd}(v_x))
\hspace{3em} \text{skip;}
\hspace{2em} \text{// read x}
\hspace{2em} s_x := x;
\hspace{2em} \text{// validate version}
\hspace{2em} \text{if (v}_x \text{ != x\_lock)}
\hspace{3em} \text{goto retry;}
\}
\]
**Sequence Locks**

\[ \text{x\_lock version even : lock free} \]
\[ \text{x\_lock version odd : lock taken} \]

```
snapshot(x){
    retry:
        // tentative version
        vx := x\_lock;
        while (is-odd(vx))
            skip;
        // read x
        sx := x;
        // validate version
        if (vx != x\_lock)
            goto retry;
}
```

```
snapshot(S){
    retry:
        // tentative versions
        for (x in S) {
            vx := x\_lock;
            while (is-odd(vx))
                skip;
        }
        // read S
        for (x in S) sx := x;
        // validate versions
        for (x in S)
            if (vx != x\_lock)
                goto retry;
}
```
PSI(T) {  
  // lock write set  
  for(x ∈ WS) {  
    wlock(x);  
    if (x ∈ RS) sx := x;  
  }  
  snapshot(RS\WS);  
  [ T ];  
  // unlock write set  
  for(x ∈ WS) wunlock(x);  
}
PSI Reference Implementation

```c
PSI(T) {  // lock write set
    for(x ∈ WS) {
        wlock(x);
        if (x ∈ RS) sx := x;
    }
    snapshot(RS\WS);
    [[ T ]];
    // unlock write set
    for(x ∈ WS) wunlock(x);
}
```

\[ [a := x] = a := sx \]
\[ [x := a] = x := a; sx := a \]
\[ [S1;S2] = [S1] ; [S2] \]
\[ ... \]

\[ \rightarrow \text{Sound: } \text{Behaviours(imp)} \subseteq \text{Behaviours(PSI\_spec)} \]
\[ \rightarrow \text{Complete: } \text{Behaviours(PSI\_spec)} \subseteq \text{Behaviours(imp)} \]
What about \textbf{mixed} accesses?

\[ \text{RPSI} = \text{PSI} + \text{mixed accesses} \]
What about \textbf{mixed} accesses?

\[
\text{acyclic}(\text{rpsi-hb}_{\text{loc}} \cup \text{mo} \cup \text{rb})
\]

with

\[
\begin{align*}
\text{rpsi-hb; rpsi-hb} & \subseteq \text{rpsi-hb} & \text{(TRANS)} \\
\text{po} \cup \text{rf} \cup \text{mo}_T & \subseteq \text{rpsi-hb} & \text{(PSI-HB)} \\
[E \setminus \mathcal{T}]; \text{rf; st} & \subseteq \text{rpsi-hb} & \text{(NT-RF)} \\
\text{st; } ([\mathcal{W}] ; \text{st; } (\text{rpsi-hb \setminus st}) ; \text{st; } [\mathcal{R}] )_{\text{loc}} ; \text{st} & \subseteq \text{rpsi-hb} & \text{(T-RF)}
\end{align*}
\]

\[
\text{RPSI} = \text{PSI} + \text{mixed accesses}
\]
What about **mixed** accesses?

⇒

**Lock-based**

RPSI reference implementation

in

C11 **release/acquire** fragment

\[
\text{RPSI} = \text{PSI} + \text{mixed accesses}
\]
**RPSI**: PSI + Mixed Accesses

MP

\[
\begin{align*}
x & := 1; \\
y & := 1;
\end{align*}
\]

T:

\[
\begin{align*}
a & := y; // 1 \\
b & := x; // 0
\end{align*}
\]
RPSI: PSI + Mixed Accesses

**MP**

\[
\begin{align*}
\text{x} & := 1; \\
\text{y} & := 1;
\end{align*}
\]

**T:**

\[
\begin{align*}
\text{a} & := \text{y}; \quad // 1 \\
\text{b} & := \text{x}; \quad // 0
\end{align*}
\]

---

```
snapshot(S)
{
    // tentative versions
    retry:
    for (x in S) {
        vx := x_lock;
        while (is-odd(vx))
            skip;
    }
    // read S
    for (x in S) sx := x;
    // validate versions
    for (x in S)
        if (vx != x_lock)
            goto retry;
}
```
RPSI: PSI + Mixed Accesses

snapshot_RPSI(S)

    // tentative versions
    retry:
    for (x in S) {
        vx := x_lock;
        while (is-odd(vx))
            skip;
    }
    // read S
    for (x in S) sx := x;
    // validate versions & values
    for (x in S)
        if (vx != x_lock && sx != x)
            goto retry;

Solution: read every location twice!
**RPSI: PSI + Mixed Accesses**

```
snapshot_RPSI(S) {
    // tentative versions
    retry:
    for (x in S) {
        vx := x_lock;
        while (is-odd(vx))
            skip;
    }
    // read S
    for (x in S) sx := x;
    // validate versions & values
    for (x in S)
        if (vx != x_lock && sx != x)
            goto retry;
}
```

**Caveat:** non-transactional writes with *same value* cannot race with transactions

**Solution:** read every location *twice*!
RPSI Reference Implementation

```
RPSI(T) {
    // lock write set
    for(x ∈ WS) {
        wlock(x);
        if (x ∈ RS) sx := x;
    }
    snapshot_RPSI(RS\WS);
    \[ T \];
    // unlock write set
    for(x ∈ WS) wunlock(x);
}
```

\[ a := x \] = a := sx
\[ x := a \] = x := a; sx := a
\[ S1;S2 \] = \[ S1 \]; \[ S2 \]
... 

⇒ **Sound**: Behaviour(imp) ⊆ Behaviour(RPSI_spec)
⇒ **Complete**: Behaviour(RPSI_spec) ⊆ Behaviour(imp)
Conclusions
Conclusions

✓ **Sound & complete** reference implementation for **PSI**

✓ **Declarative RPSI** semantics with **mixed** accesses

✓ **Sound & complete** reference implementation for **RPSI**
Conclusions

✓ **Sound & complete** reference implementation for *PSI*

✓ **Declarative RPSI** semantics with *mixed* accesses

✓ **Sound & complete** reference implementation for *RPSI*

➡ PSI under *other* weak memory models

➡ *Program Logics* for STMs
Conclusions

✓ Sound & complete reference implementation for PSI

✓ Declarative RPSI semantics with mixed accesses

✓ Sound & complete reference implementation for RPSI

➡ PSI under other weak memory models

➡ Program Logics for STMs

Thank you for listening!