Verification of Concurrent Systems under Weak Consistency Models

Ahmed Bouajjani
Paris Diderot University

MPRI, Paris, December 2020
Lecture 2
PSO (Partial Store Ordering)

- Writes can be delayed, but they are visible to the issuer
- They become visible to all processes simultaneously
- Visible writes are visible in the same order to all processes
- Writes by a same process become visible in their issue order on a same variable
PSO (Partial Store Ordering)

- Writes can be delayed, but they are visible to the issuer
- They become visible to all processes simultaneously
- Visible writes are visible in the same order to all processes
- Writes by a same process become visible in their issue order
  on a same variable

- Weaker model than TSO: writes on ≠ variables can be reordered
PSO (Partial Store Ordering)

- Writes can be delayed, but they are visible to the issuer
- They become visible to all processes simultaneously
- Visible writes are visible in the same order to all processes
- Writes by a same process become visible in their issue order on a same variable

- Weaker model than TSO: writes on ≠ variables can be reordered

  - Operational Model ?
  - Reachability Problem ?
PSO: Operational Model

- Semantics similar to the one of TSO
- One store buffer per variable and per process
PSO behavior

Writes can overtake other writes on $\neq$ variables

\[
\begin{array}{cccc}
\text{x=y=0} \\
(1) \ w(x,1) & (2) \ w(y,1) & (3) \ r(y,1) & (4) \ r(x,0) \\
\hline
\text{x=y=1}
\end{array}
\]
PSO behavior

Writes can overtake other writes on ≠ variables

\[
\begin{array}{c|c}
\text{x=y=0} \\
(1) \ w(x,1) & (3) \ r(y,1) \\
(2) \ w(y,1) & (4) \ r(x,0) \\
\hline
\text{x=y=1} \\
\end{array}
\]

w(y,1)  r(y,1)  r(x,0)  w(x,1)

Permute (1) and (2)
An example of PSO program

P1

> w(x, 1)
> w(y, 1)
> w(x, 2)

x = y = 0

P2

> r(x, 2)
> r(y, 0)

x = 0

y = 0
An example of PSO program

P1
w(x,1)
w(y,1)
w(x,2)>

x=y=0

P2
r(x,2)
r(y,0)>

x=1
y=0
An example of PSO program
An example of PSO program

P1
\[ w(x, 1) \]
\[ w(y, 1) \]
\[ w(x, 2) \]

\[ x = y = 0 \]

P2
\[ r(x, 2) \]
\[ r(y, 0) \]

x = 2
y = 1
An example of PSO program

P1
w(x,1)
w(y,1)
w(x,2)

P2
r(x,2)
r(y,0)

x=y=0

x=2
y=1
An example of PSO program

P1
w(x,1)
w(y,1)
w(x,2)

P2
r(x,2)
\textcolor{red}{x=2}
\textcolor{red}{y=1}

x=y=0

An example of PSO program

P1
w(x,1)
w(y,1)
w(x,2)

x=y=0

P2
r(x,2)
r(y,0)

x=1
y=0
An example of PSO program

P1
w(x,1)
w(y,1)
w(x,2)

x=y=0

P2
r(x,2)
r(y,0)

x=2
y=0
An example of PSO program
An example of PSO program

P1
w(x,1)
w(y,1)
w(x,2)

x=y=0

P2
r(x,2)
r(y,0)

x=2
y=0
From PSO programs to LCS

- 1-channel machine per process + composition
From PSO programs to LCS

- 1-channel machine per process + composition

- Each process:
  - **write**: puts the operation at the tail of the channel
  - **read**: checks the channel, then the memory
  - **memory update**: moves the head of the channel to the memory
From PSO programs to LCS

- 1-channel machine per process + composition

- Each process:
  - write: puts the operation at the tail of the channel
  - read: checks the channel, then the memory
  - memory update: moves the head of the channel to the memory
  - guesses writes by other processes; put them in the channel
From PSO programs to LCS

- 1-channel machine per process + composition

- Each process:
  - **write**: puts the operation at the tail of the channel
  - **read**: checks the channel, then the memory
  - **memory update**: moves the head of the channel to the memory
  - **guesses writes by other processes**: put them in the channel

- **Validation of the guesses by composition**:
  - transitions are labelled by **write operations + process id**
  - machines are **synchronized** on these actions
From LCS to PSO programs
• P1 simulates a LCS with one channel using x and y:
  • send(m) —> write(x, m)
  • receive(m) —> read(y, m)

• P2 forwards values from x to y

Same reduction as for TSO
TSO + R $\rightarrow$ R/W relaxation

Reads can be delayed

<table>
<thead>
<tr>
<th>$x=y=0$</th>
<th>$x=y=1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1) $w(x,1)$</td>
<td>(2) $r(x,1)$</td>
</tr>
</tbody>
</table>

| $x=y=0$ | $x=y=1$ |
TSO + R —> R/W relaxation

Reads can be delayed

\[
\begin{array}{cccc}
& x=y=0 & & \\
(1) w(x, 1) & (2) r(x, 1) & (4) r(y, 1) & \\
(3) w(y, 1) & & (5) r(x, 0) \\
& x=y=1 & & \\
\end{array}
\]

r(x, 0)  w(x, 1)  r(x, 1)  w(y, 1)  r(y, 1)
**TSO + R —> R/W relaxation**

Reads can be delayed

<table>
<thead>
<tr>
<th></th>
<th>(1) r(x,1)</th>
<th>(2) r(y,0)</th>
<th>(3) r(y,1)</th>
<th>(4) r(x,0)</th>
<th>(5) w(x,1)</th>
<th>(6) w(y,1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>x=y=0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>r(x,1)</td>
<td>r(y,0)</td>
<td>r(x,0)</td>
<td></td>
<td>w(x,1)</td>
<td>w(y,1)</td>
</tr>
<tr>
<td>x=y=1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
TSO + R —> R/W relaxation

Reads can be delayed

<table>
<thead>
<tr>
<th></th>
<th>(1)</th>
<th>(2)</th>
<th>(3)</th>
<th>(4)</th>
<th>(5)</th>
<th>(6)</th>
</tr>
</thead>
<tbody>
<tr>
<td>x=y=0</td>
<td>r(x, 1)</td>
<td>r(y, 0)</td>
<td>r(y, 1)</td>
<td>r(x, 0)</td>
<td>w(x, 1)</td>
<td>w(y, 1)</td>
</tr>
<tr>
<td>x=y=1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

r(x, 0)  w(x, 1)  r(x, 1)  r(y, 0)  w(y, 1)  r(y, 1)

Permute (1) and (2), or (3) and (4)
**TSO + R —> R/W relaxation**

**Reads can be delayed**

<table>
<thead>
<tr>
<th></th>
<th>x=y=0</th>
<th></th>
<th>x=y=1</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1)</td>
<td>r(x,1)</td>
<td>(3)</td>
<td>r(y,1)</td>
</tr>
<tr>
<td>(2)</td>
<td>w(y,1)</td>
<td>(4)</td>
<td>w(x,1)</td>
</tr>
</tbody>
</table>

TSO + R relaxation allows reads to be delayed, which is indicated in the diagram by the sequence of operations:

- Read `r(x,1)` from `x` at step (1), which is followed by write `w(y,1)` to `y` at step (2).
- Read `r(y,1)` from `y` at step (3), which is followed by write `w(x,1)` to `x` at step (4).

This sequence shows that reads can be delayed after writes, which is a feature of the TSO + R relaxation mechanism.
TSO + R —> R/W relaxation

Reads can be delayed

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>(x=y=0)</td>
<td>(1) (r(x,1))</td>
<td>(3) (r(y,1))</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(2) (w(y,1))</td>
<td>(4) (w(x,1))</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>(x=y=1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\(w(y,1)\) \(r(y,1)\) \(w(x,1)\) \(r(x,1)\)

Permute (1) and (2)
TSO + R $\rightarrow$ R/W relaxation

Reads can be delayed

\[
\begin{array}{c|c|c}
\hline
x=y=0 & x=y=1 \\
(1) \ r(x,1) & (3) \ r(y,1) \\
(2) \ w(y,1) & (4) \ w(x,1) \\
\hline
\end{array}
\]

Speculative writes $\Rightarrow$ Causality cycle:
- [2] is executed assuming [1] will be executed
- [1] is executed, but depends causally from [2]
- Writes and reads are sent to the buffers
- Updates: first write on a buffer is taken committed if no preceding read on the same variable in the buffer
- Validation of a read: value in buffer and in memory match
- Writes and reads are sent to the buffers
- Updates: first write on a buffer is taken committed if no preceding read on the same variable in the buffer
- Validation of a read: value in buffer and in memory match
- Writes and reads are sent to the buffers
- Updates: first write on a buffer is taken committed if no preceding read on the same variable in the buffer
- Validation of a read: value in buffer and in memory match
TSO + R —> R/W relaxation: Operational model

- Writes and reads are sent to the buffers
- Updates: first write on a buffer is taken committed if no preceding read on the same variable in the buffer
- Validation of a read: value in buffer and in memory match
- Writes and reads are sent to the buffers
- Updates: first write on a buffer is taken committed if no preceding read on the same variable in the buffer
- Validation of a read: value in buffer and in memory match
- Writes and reads are sent to the buffers
- Updates: first write on a buffer is taken committed if no preceding read on the same variable in the buffer
- Validation of a read: value in buffer and in memory match
Allowing R —> R/W relaxation: Undecidability

- Reduction of the Post Correspondance Problem (PCP)

**PCP**: Given two sequences of nonempty words

\[
\{u_1, \ldots, u_n\} \text{ and } \{v_1, \ldots, v_n\}
\]

over some alphabet A. Is there a sequence \(i_1, \ldots, i_k \in \{1, \ldots, n\}\) such that

\[
u_{i_1} \cdots u_{i_k} = v_{i_1} \cdots v_{i_k}\]
Allowing R $\rightarrow$ R/W relaxation: Undecidability
Allowing $R \rightarrow R/W$ relaxation: Undecidability

If all buffers are empty and all reads are validated

1. $v_{j_1} \cdots v_{j_m} \preceq u_{i_1} \cdots u_{i_n}$
2. $u_{i_1} \cdots u_{i_n} \preceq v_{j_1} \cdots v_{j_m}$
3. $j_1 \cdots j_m \preceq u_1 \cdots i_n$
4. $i_1 \cdots i_n \preceq j_1 \cdots j_m$

$\Rightarrow$ solution to PCP
Allowing R → R/W relaxation: Undecidability

Assume $u_{i_1} \cdots u_{i_n} = v_{j_1} \cdots v_{j_m}$ and $i_1 \cdots i_n = j_1 \cdots j_m$

T1: r(y2; in) w(y1; in) r(x2; uin) w(x1; uin) r(y2; i1) w(y1; i1) r(x2; ui1) w(x1; ui1)

T2: r(y1; jn) w(y2; jn) r(x1; vjn) w(x2; vjn) r(y1; j1) w(y2; j1) r(x1;vj1) w(x2; vj1)
Allowing $R \rightarrow R/W$ relaxation: Undecidability

Assume $u_{i_1} \cdots u_{i_n} = v_{j_1} \cdots v_{j_m}$ and $i_1 \cdots i_n = j_1 \cdots j_m$

T1: $r(y_2; \text{in})$  $r(x_2; u_{i_1})$  $r(y_2; i_1)$  $r(x_2; u_{i_1})$  $w(y_1; \text{in})$  $w(x_1; u_{i_1})$  $w(y_1; i_1)$  $w(x_1; u_{i_1})$
T2: $w(y_2; j_n)$  $w(x_2; v_{j_n})$  $w(y_2; j_1)$  $w(x_2; v_{j_1})$  $r(y_1; j_n)$  $r(x_1; v_{j_n})$  $r(y_1; j_1)$  $r(x_1; v_{j_1})$
Bounded R $\rightarrow$ R/W relaxation: Decidability

Theorem:

If the number of reads in the buffer is bounded then the reachability problem becomes decidable

- Use the same reduction as for TSO
- This time keep reads in the buffer
- LCS with a bounded number of rigid symbols are WSS
Reachability over WMM: other results

- NSW (Non speculative writes): PSO + R->W
  Reachability is decidable
  [Atig, Bouajjani, Burkhardt, Musuvathi, 2013]

- Power: reachability is undecidable
  [Atig, Bouajjani, Meyer, … , 2013]
Reachability over WMM: other results

- NSW (Non speculative writes): PSO + R->W
  
  Reachability is decidable
  
  [Atig, Bouajjani, Burkhardt, Musuvathi, 2013]

- Power: reachability is undecidable

  [Atig, Bouajjani, Meyer, … , 2013]

- Hard problem
- Undecidability for complex WMM
- Works on approximate decision procedures
Client

Service

Infrastructure

CLM

Imp1

Imp1/Spec

Imp2

Imp2

ICM

ICM'

TSO

SC

≈

refines?

satisfies?

Guarantee

Assume

Assume

Guarantee

Guarantee

Assume

Assume

weaker
Articles


What’s Decidable about Weak Memory Models, M-F. Atig, A. Bouajjani, S. Burckhardt, M. Musuvathi, ESOP 2012.
