Parametric Verification of Concurrent Programs under the TSO Weak Memory Model

Ahmed Bouajjani
Paris Diderot University

Based on joint work with

Parosh A. Abdulla        Mohamed Faouzi Atig        T. Phong Ngo
Uppsala University

Sebastian Burckhardt      Madan Musuvathi
Microsoft Research

SynCoP+PV’17, Uppsala, April 22, 2017
Sequential Consistency

- Concurrent processes with Shared Memory
- Operations: Writes and Reads
- Computation of different processes are shuffled
- Program order is preserved for each process
Sequential Consistency

- Concurrent processes with Shared Memory
- Operations: Writes and Reads
- Computation of different processes are shuffled
- Program order is preserved for each process

=> Strong consistency:
   Operations are immediately visible to all processes
Sequential Consistency

• Concurrent processes with Shared Memory
• Operations: Writes and Reads
• Computation of different processes are shuffled
• Program order is preserved for each process

=> Strong consistency:
   Operations are immediately visible to all processes

• Simple and Intuitive model
• Disallows many hardware/compiler optimisations
Weak Memory Models

\[ x = y = 0 \]

```
po  \rightarrow \text{write}(x,1) \leftarrow \text{hb} \rightarrow \text{read}(x,0)
```

```
po  \rightarrow \text{read}(y,0) \leftrightarrow \text{hb} \rightarrow \text{read}(x,0)
```

\[ \text{SC} \]

```
\text{read}(x,0) \hspace{10pt} \text{write}(x,1) \hspace{10pt} \text{read}(y,0)
```

Weak Memory Models

Relax the Program Order Constraints

\( x = y = 0 \)

SC
\[\text{read}(x,0) \quad \text{write}(x,1) \quad \text{read}(y,0)\]

TSO
\[\text{read}(x,0) \quad \text{read}(y,0) \quad \text{write}(x,1)\]

Swap operations
Weak Memory Models

Relax the Program Order Constraints

\[ x = y = 0 \]

\[ \text{write}(x, 1) \quad \text{read}(x, 0) \]
\[ \text{read}(y, 0) \quad \text{read}(y, 0) \]
\[ \text{write}(x, 1) \quad \text{read}(x, 0) \]

SC:
\[ \text{read}(x, 0) \quad \text{write}(x, 1) \quad \text{read}(y, 0) \]

TSO:
\[ \text{read}(x, 0) \quad \text{read}(y, 0) \quad \text{write}(x, 1) \]

Swap operations

Execute in parallel
Total Store Ordering

- writes are sent to **store buffers** (one per process)
- writes are committed to memory at any time
- reads are from
  - own store buffer if a value exists (last write to the variable)
  - otherwise from the memory
- fences executed when own buffer is empty
Non SC Behaviours

$x=y=0$

write(x,1) write(y,1)
read(y,0) read(x,0)

CS1 CS2

CS1 and CS2 ?
Non SC Behaviours

\[ x = y = 0 \]

- Impossible under SC
Non SC Behaviours

\[
x = y = 0
\]

- **Impossible under SC**
- **Possible under TSO!**
  - writes are *delayed*: pending in store buffers
  - reads get old values in the memory (0’s)
Non SC Behaviours

\[ x = y = 0 \]

- **Impossible under SC**
- **Possible under TSO!**
  - writes are *delayed*: pending in store buffers
  - reads get old values in the memory (0’s)
  - \( \rightarrow \) po constraints are *relaxed*
  - \( \rightarrow \) reads can *overtake* writes
TSO: Semantics

\[
\begin{align*}
P1 & \quad > \quad w(x,1) \\
r(y,0) & \\
\end{align*}
\]

\[
\begin{align*}
P2 & \quad > \quad w(y,1) \\
r(x,0) & \\
\end{align*}
\]
TSO: Semantics

P1

w(x,1)

r(y,0)

P2

w(y,1)

r(x,0)

P1

w(x,1)

P2

w(y,1)

x=0

y=0
TSO: Semantics

P1

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

P2

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

Diagram:

- From P1:
  - $w(x,1)$
  - $r(y,0)$

- From P2:
  - $w(y,1)$

- At x=0
- At y=0
Avoiding Reordering: Fences

\[ x = y = 0 \]

\[ \text{hb} \rightarrow \text{write}(x, 1) \rightarrow \text{fence} \rightarrow \text{read}(y, 0) \rightarrow \text{hb} \]

\[ \text{hb} \rightarrow \text{write}(y, 1) \rightarrow \text{fence} \rightarrow \text{read}(x, 0) \rightarrow \text{hb} \]

CS1 and CS2 ?

- A fence forces **flushing** the store buffer
- \( \Rightarrow \) CS1 and CS2 becomes **impossible**
Avoiding Reordering: Fences

\[ x = y = 0 \]

--\> A fence forces *flushing* the store buffer
--\> => CS1 and CS2 becomes impossible

**SC can be enforced:** fence after each write
Safety/Reachability Verification Problems

for every \( n \), for every \( m \),

\[
\left[ P_1 \parallel \ldots \parallel P_n \right]_{TSO(m)} \text{ satisfies } \text{Always (Safe)}
\]

there is \( n \), there is \( m \),

\[
\left[ P_1 \parallel \ldots \parallel P_n \right]_{TSO(m)} \text{ satisfies } \text{Reachable (Not Safe)}
\]
First step: Let us fix the number of processes for every $m$,

$[P_1 \parallel \ldots \parallel P_n]_{TSO(m)}$ satisfies Always (Safe)

there is $m$,

$[P_1 \parallel \ldots \parallel P_n]_{TSO(m)}$ satisfies Reachable (Not Safe)
First step: Let us fix the number of processes

Consider Unbounded Store Buffers

there is $m$, 

$[ P_1 \ || \ ... \ || \ P_n ]_{TSO(m)}$ satisfies Reachable (Not Safe) 

$\iff$ 

$[ P_1 \ || \ ... \ || \ P_n ]_{TSO(\infty)}$ satisfies Reachable (Not Safe)
Reachability Problem for a given number of processes: Decidability, Complexity

Assume that processes are finite state

Under \textbf{SC}, the control state reachability problem is

- \textit{PSPACE}-complete, for a fixed number of processes
- \textit{EXPSPACE}-complete, for the parametric case
Assume that processes are finite state

Under SC, the control state reachability problem is

- **PSPACE-complete**, for a fixed number of processes
- **EXPSPACE-complete**, for the parametric case

What about the TSO(∞) reachability?

store buffers are **unbounded perfect FIFO queues**!!
Assume that processes are finite state

Under $\text{SC}$, the control state reachability problem is

- PSPACE-complete, for a fixed number of processes
- EXPSPACE-complete, for the parametric case

What about the TSO($\infty$) reachability?

store buffers are unbounded perfect FIFO queues!!

What about the parametric TSO($\infty$) reachability?
Reachability Problem for TSO programs: Results

- The TSO reachability problem is **decidable**
Reachability Problem for TSO programs: Results

- The TSO reachability problem is **decidable**
- … but it is **highly complex** (non primitive recursive)

Reduction to/from reachability in **lossy channel systems**

[Atig, B., Burckhardt, Musuvathi, POPL’10]
Reachability Problem for TSO programs: Results

- The TSO reachability problem is **decidable**
- … but it is **highly complex** (non primitive recursive)
  Reduction to/from reachability in **lossy channel systems**
  [Atig, B., Burckhardt, Musuvathi, POPL’10]

- The **parametric** TSO reachability problem is **decidable**
Reachability Problem for TSO programs: Results

- The TSO reachability problem is **decidable**
- … but it is **highly complex** (non primitive recursive)
  
  Reduction to/from reachability in **lossy channel systems**
  
  [Atig, B., Burckhardt, Musuvathi, POPL’10]

- The **parametric** TSO reachability problem is **decidable**
  - A **dual semantics for TSO**
  - Monotonic system w.r.t. WQO

- **Simpler and more efficient reduction**
  
  [Abdulla, Atig, B., Ngo, CONCUR’16]
An example of TSO program

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

$P1$ -> TSO store buffer of $P1$ -> $P2$

$x=y=0$

$P1$ -> $x=0$
$y=0$
An example of TSO program

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

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

x\equiv y = 0

TSO store buffer of P1

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

x=0
y=0
An example of TSO program

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

> x=y=0

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

TSO store buffer of P1

x=1
y=0
An example of TSO program

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

\( x = y = 0 \)

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

TSO store buffer of P1

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

\( x = 1 \)
\( y = 1 \)
An example of TSO program

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

x=y=0

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

> w(x,2)

w(y,1)  w(x,1)

TSO store buffer of P1

x=2
y=1
An example of TSO program

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

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

\( x = y = 0 \)

TSO store buffer of P1
- \( w(x, 2) \)
- \( w(y, 1) \)
- \( w(x, 1) \)

\( x = 2 \)
\( y = 1 \)
An example of TSO program

Deadlock under the TSO semantics
TSO Store Buffers —> Lossy Channels ?

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

> x=y=0

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

P1 -> w(x,2) w(y,1) w(x,1) -> Lossy Fifo Channel

x=0
y=0
TSO Store Buffers $\rightarrow$ Lossy Channels?

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

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

$x=y=0$

Lossy Fifo Channel

P1 $\rightarrow$

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

Lossy Fifo Channel

x=1

y=0
TSO Store Buffers —> Lossy Channels ?

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

P2
w(x,1)
w(x,2)

Lossy Fifo Channel

x=y=0

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

x=1
y=0
TSO Store Buffers —> Lossy Channels?

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

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

x = y = 0

Lossy Fifo Channel

x = 2
y = 0
TSO Store Buffers —> Lossy Channels?

P1

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

x = y = 0

P2

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

x = 2
y = 0

Lossy Fifo Channel
TSO Store Buffers $\rightarrow$ Lossy Channels?

Unsound simulation of TSO!
Store Memory Snapshots

Future Snapshots of the Memory

P1

x = y = 0

P2

x = 0
y = 0

w(x, 1)
w(y, 1)
w(x, 2)
r(x, 2)
r(y, 0)
Store Memory Snapshots

Future Snapshots of the Memory

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

x=y=0

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

Future Snapshots of the Memory

x=1
y=0

x=0
y=0
Store Memory Snapshots

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

x = y = 0

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

Future Snapshots of the Memory

x = 0
y = 0
Store Memory Snapshots

Future Snapshots of the Memory
Store Memory Snapshots

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

x=y=0

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

Future Snapshots of the Memory

x=1
y=0
Store Memory Snapshots with Losses

Future Snapshots of the Memory

+ Lossyness
Store Memory Snapshots with Losses

Future Snapshots of the Memory

+ Lossyness
Store Memory Snapshots with Losses

Future Snapshots of the Memory + Lossyness
Store Memory Snapshots with Losses

Future Snapshots of the Memory + Lossyness

Valid Simulation of TSO
From TSO to Lossy Channel Systems

- 1-channel machine per process + composition
From TSO to Lossy Channel Systems

- 1-channel machine per process + composition

- Each process:
  - write: puts a new memory state 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 TSO to Lossy Channel Systems

- 1-channel machine per process + composition

- Each process:
  - write: puts a new memory state at the tail of the channel
  - read: checks the channel, then the memory
  - memory update: moves the head of the channel to the memory

Problem: Interferences between processes? Processes must agree on the same order of memory updates
From TSO to Lossy Channel Systems

- 1-channel machine per process + composition

- Each process:
  • write: puts a new memory state at the tail of the channel
  • read: checks the channel, then the memory
  • memory update: moves the head of the channel to the memory

   **Problem:** Interferences between processes? Processes must agree on the same order of memory updates
  • 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 Lossy Channel Systems to TSO 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**
From Lossy Channel Systems to TSO 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

P2 can miss some values
Thm: The control state reachability problem under TSO is reducible to the reachability problem in lossy channel systems, and vice-versa.

[Atig, B., Burckhardt, Musuvathi, 2010]
Reachability for TSO programs

[Atig, B., Burckhardt, Musuvathi, 2010]

**Thm:** The control state reachability problem under TSO is reducible to the reachability problem in lossy channel systems, and vice-versa.

**Coro:** The control state reachability problem under TSO is decidable, and it is non primitive recursive.

Well …

The complexity is high!
The complexity is high!

... but this is not the main problem
The complexity is high!

… but this is not the main problem

The proposed encoding of TSO programs as LCS’s

- Is not practical:
  it requires handling memory snapshots

- Can not be extended to the parametric case
  it manipulates process id’s
Well …

The complexity is high!
    … but this is not the main problem

The **proposed encoding** of TSO programs as LCS’s

- Is not practical:
  it requires handling **memory snapshots**

- Can not be extended to the parametric case
  it manipulates **process id’s**

=> We need to change our angle of view…
Dual TSO

- **Store Buffers** $\rightarrow$ **Load Buffers**
- **Writes** immediately update the Memory
- **Reads** are sent by the memory to processes
- **Reads** can be skipped by processes (Load Buffers are **lossy**)

![Diagram of Dual TSO](image)
Dual TSO

- **Store Buffers —> Load Buffers**
- Writes **immediately update** the Memory
- Reads are **sent by the memory** to processes
- Reads **can be skipped** by processes (Load Buffers are **lossy**)
- => One sequence of memory updates (order of writes)
- => Buffers contain **expected reads** by processes
- => Buffers represent a “**(sub)history**” of the memory updates

\[ r(x,0) \quad r(y,1) \quad r(x,1) \quad r(y,3) \]
\[ r(y,1) \quad r(y,3) \quad r(x,2) \]
Dual TSO: Semantics

P1

> w(x,1)

r(y,0)

P2

> w(y,1)

r(x,0)
Dual TSO: Semantics

P1

> w(x,1)

r(y,0)

P2

> w(y,1)

r(x,0)

Diagram showing the flow of processes P1 and P2 with events w(x,1), w(y,1), r(y,0), and r(x,0) with conditions x=0 and y=0.
Dual TSO: Semantics

P1

\[ \frac{w(x, 1)}{r(y, 0)} \]

P2

\[ \frac{w(y, 1)}{r(x, 0)} \]
Dual TSO: Semantics

P1
w(x,1)
r(y,0)

P2
w(y,1)
r(x,0)
Thm: The Dual TSO semantics is equivalent to the TSO semantics with respect to the reachability problem.
Given \( n \geq 1 \), a configuration of size \( n \) is:
- \( q_1, \ldots, q_n \), control states of \( P_1, \ldots, P_n \)
- \( B_1, \ldots, B_n \), contents of the load buffers of \( P_1, \ldots, P_n \)
- \( \text{Mem} \), the memory state

WQO \( \leq \) between configurations of sizes \( n \) and \( m \):
- same memory state
- exists an injective function \( h: [n] \rightarrow [m] \) s.t.
  - same control state, for each \( P_i \) and \( P'_h(i) \)
  - sub-word relation on load buffers, for each \( P_i \) and \( P'_h(i) \)

**Thm**: Parameterized Dual TSO systems are monotonic w.r.t. \( \leq \)
Comparing the two encodings

Dual TSO:
- No memory snapshot
- No reference to Process Id’s
- Applicable to Parametric Verification
- More efficient verification algorithm
## Experimental Results: Dual-TSO vs Memorax

<table>
<thead>
<tr>
<th>Program</th>
<th>#P</th>
<th>Dual-TSO</th>
<th></th>
<th>Memorax</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>#T</td>
<td>#C</td>
<td>#T</td>
<td>#C</td>
</tr>
<tr>
<td>SB</td>
<td>5</td>
<td>0.3</td>
<td>10641</td>
<td>559.7</td>
<td>10515914</td>
</tr>
<tr>
<td>LB</td>
<td>3</td>
<td>0.0</td>
<td>2048</td>
<td>71.4</td>
<td>1499475</td>
</tr>
<tr>
<td>WRC</td>
<td>4</td>
<td>0.0</td>
<td>1507</td>
<td>63.3</td>
<td>1398393</td>
</tr>
<tr>
<td>ISA2</td>
<td>3</td>
<td>0.0</td>
<td>509</td>
<td>21.1</td>
<td>226519</td>
</tr>
<tr>
<td>RWC</td>
<td>5</td>
<td>0.1</td>
<td>4277</td>
<td>61.5</td>
<td>1196988</td>
</tr>
<tr>
<td>W+RWC</td>
<td>4</td>
<td>0.0</td>
<td>1713</td>
<td>83.6</td>
<td>1389009</td>
</tr>
<tr>
<td>IRIW</td>
<td>4</td>
<td>0.0</td>
<td>520</td>
<td>34.4</td>
<td>358057</td>
</tr>
<tr>
<td>Nbw_w_wr</td>
<td>2</td>
<td>0.0</td>
<td>222</td>
<td>10.7</td>
<td>200844</td>
</tr>
<tr>
<td>Sense_rev_bar</td>
<td>2</td>
<td>0.1</td>
<td>1704</td>
<td>0.8</td>
<td>20577</td>
</tr>
<tr>
<td>Dekker</td>
<td>2</td>
<td>0.1</td>
<td>5053</td>
<td>1.1</td>
<td>19788</td>
</tr>
<tr>
<td>Dekker_simple</td>
<td>2</td>
<td>0.0</td>
<td>98</td>
<td>0.0</td>
<td>595</td>
</tr>
<tr>
<td>Peterson</td>
<td>2</td>
<td>0.1</td>
<td>5442</td>
<td>5.2</td>
<td>90301</td>
</tr>
<tr>
<td>Peterson_loop</td>
<td>2</td>
<td>0.2</td>
<td>7632</td>
<td>5.6</td>
<td>100082</td>
</tr>
<tr>
<td>Szymanski</td>
<td>2</td>
<td>0.6</td>
<td>29018</td>
<td>1.0</td>
<td>26003</td>
</tr>
<tr>
<td>MP</td>
<td>4</td>
<td>0.0</td>
<td>883</td>
<td>TO</td>
<td>•</td>
</tr>
<tr>
<td>Ticket_spin_lock</td>
<td>3</td>
<td>0.9</td>
<td>18963</td>
<td>TO</td>
<td>•</td>
</tr>
<tr>
<td>Bakery</td>
<td>2</td>
<td>2.6</td>
<td>82050</td>
<td>TO</td>
<td>•</td>
</tr>
<tr>
<td>Dijkstra</td>
<td>2</td>
<td>0.2</td>
<td>8324</td>
<td>TO</td>
<td>•</td>
</tr>
<tr>
<td>Lamport_fast</td>
<td>3</td>
<td>17.7</td>
<td>292543</td>
<td>TO</td>
<td>•</td>
</tr>
<tr>
<td>Burns</td>
<td>4</td>
<td>124.3</td>
<td>2762578</td>
<td>TO</td>
<td>•</td>
</tr>
</tbody>
</table>
## Experimental Results: Parameterised Case

<table>
<thead>
<tr>
<th>Program</th>
<th>Dual-TSO</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>#T</td>
</tr>
<tr>
<td>SB</td>
<td>0.0</td>
</tr>
<tr>
<td>LB</td>
<td>0.6</td>
</tr>
<tr>
<td>MP</td>
<td>0.0</td>
</tr>
<tr>
<td>WRC</td>
<td>0.8</td>
</tr>
<tr>
<td>ISA2</td>
<td>4.3</td>
</tr>
<tr>
<td>RWC</td>
<td>0.2</td>
</tr>
<tr>
<td>W+RWC</td>
<td>1.5</td>
</tr>
<tr>
<td>IRIW</td>
<td>4.6</td>
</tr>
</tbody>
</table>
Scalability: D-TSO vs Memorex
Conclusion

- Verification under WMM’s is **hard**
- **Decidability for** (relatively strong) models such as **TSO**
- High complexity, but **practical approaches are possible**
- **Duality** —> **simple, general, and efficient** decision procedure
Conclusion

- Verification under WMM’s is **hard**
- **Decidability for** (relatively strong) models such as **TSO**
- High complexity, but **practical approaches are possible**
- **Duality** —> **simple, general, and efficient** decision procedure
- Extension to **other models**?
- Hardware/Programming Languages models?
Conclusion

- Verification under WMM’s is **hard**
- **Decidability for** (relatively strong) models such as **TSO**
- High complexity, but **practical approaches are possible**
- **Duality** —> **simple, general, and efficient** decision procedure
- Extension to other models ?
- Hardware/Programming Languages models ?
- Related to **Consistency Criteria** in concurrent/distributed syst.
Conclusion

- Verification under WMM’s is **hard**
- **Decidability** for (relatively strong) models such as TSO
- High complexity, but **practical approaches are possible**
- **Duality** —> **simple, general, and efficient** decision procedure

- Extension to other models ?
- Hardware/Programming Languages models ?
- Related to **Consistency Criteria** in concurrent/distributed syst.
- Undecidability for more complex models (RMO, Power)
- Under/upper-approximate analyses are needed
  E.g., context-bounded analysis for TSO
  [Atig, B., Parlato, CAV 2011]
  context-bounded analysis for Power
  [Abdulla, Atig, B., Ngo, TACAS 2017]