Lecture 3:
Verification of Weak Memory Models

Part 1: State Reachability Problem

Ahmed Bouajjani

LIAFA, University Paris Diderot – Paris 7

[Atig, B., Burckhardt, Musuvathi, POPL'10, ESOP'12]

[Atig, B., Parlato, 2011]

VTSA, MPI-Saarbrücken, September 2012

Sequential Consistency (SC) model

- Parallel processes with shared memory
- Interleaving (Sequentially Consistent) semantics:
  - Computations of different processes are shuffled
  - Program order is preserved for each process.

Total Store Ordering (TSO)

- Reads can overtake writes on $\neq$ variables.
- FIFO buffers where writes are stored to be executed later.
- Reads take values from the main memory if no writes in the buffer on the same variable. Otherwise they get the value of the last write in the buffer on the same variable.

Write-to-Read Relaxation

$$\begin{array}{lcl} P_1 & : & \text{write}(x,1) \quad ; \quad \text{read}(y,0) \\ P_2 & : & \text{read}(x,0) \end{array}$$

A scheduling for SC semantics: 3 steps

$$\begin{array}{lcl} P_1 & : & \text{write}(x,1)_{(2)} \quad ; \quad \text{read}(y,0)_{(3)} \\ P_2 & : & \text{read}(x,0)_{(1)} \end{array}$$

Write-to-Read Relaxation

$$\begin{array}{lcl} P_1 & : & \mathsf{write}(x,1) \quad ; \quad \mathsf{read}(y,0) \\ P_2 & : & \mathsf{read}(x,0) \end{array}$$

A scheduling for SC semantics: 3 steps

$$\begin{array}{lcl} P_1 & : & \mathsf{write}(x,1)_{(2)} \quad ; \quad \mathsf{read}(y,0)_{(3)} \\ P_2 & : & \mathsf{read}(x,0)_{(1)} \end{array}$$

Allowing reordering of actions on different variables: 2 steps !

$$\begin{array}{lcl} P_1 & : & \mathsf{read}(y,0)_{(1)} \quad ; \quad \mathsf{write}(x,1)_{(2)} \\ P_2 & : & \mathsf{read}(x,0)_{(1)} \end{array}$$

Relaxed Models

- Read Local Write Early

  write (x,d) ; read (x,d) $\mapsto$ write (x,d)

- (+) W $\rightarrow$ R: Write to Read

  write (x,d) ; read (y,d') $\mapsto$ read (y,d') ; write (x,d)

  $\Rightarrow$ TSO model (Total Store Ordering)

- (+) W $\rightarrow$ W: Write to Write

  write (x,d) ; write (y,d') $\mapsto$ write (y,d') ; write (x,d)

  $\Rightarrow$ PSO model (Partial Store Ordering)

- (+) R $\rightarrow$ R/W: Read to Read/Write

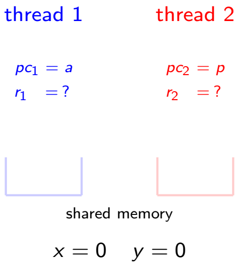
  $\Rightarrow$ $\sim$RMO model (Relaxed Memory Ordering)

Relaxation $\Rightarrow$ Potential Bad Behaviors

$$x = y = 0$$

thread 1	thread 2
a: $y = 1$	p: $x = 1$
b: $r_1 = x$	q: $r_2 = y$
c: if($r_1 == 0$) {	s: if($r_2 == 0$) {
d: ...	t: ...
}	}

1- Initial state



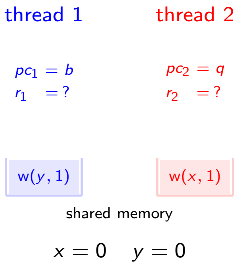
Dekker's mutual exclusion protocol. Fails under Write to Read relaxation.

Relaxation $\Rightarrow$ Potential Bad Behaviors

$x = y = 0$

thread 1	thread 2
a: $y = 1$	p: $x = 1$
b: $r_1 = x$	q: $r_2 = y$
c: if$(r_1 == 0)$ {	s: if$(r_2 == 0)$ {
d: ...	t: ...
}	}

2- Writes are postponed



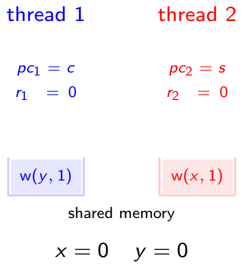
Dekker's mutual exclusion protocol. Fails under Write to Read relaxation.

Relaxation $\Rightarrow$ Potential Bad Behaviors

$$x = y = 0$$

thread 1	thread 2
a: $y = 1$	p: $x = 1$
b: $r_1 = x$	q: $r_2 = y$
c: if($r_1 == 0$) {	s: if($r_2 == 0$) {
d: ...	t: ...
}	}

3- Reading from memory



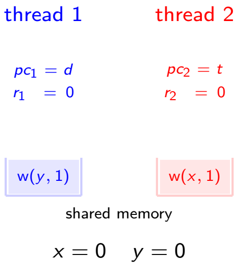
Dekker's mutual exclusion protocol. Fails under Write to Read relaxation.

Relaxation $\Rightarrow$ Potential Bad Behaviors

$x = y = 0$

thread 1	thread 2
a: $y = 1$	p: $x = 1$
b: $r_1 = x$	q: $r_2 = y$
c: if$(r_1 == 0)$ {	s: if$(r_2 == 0)$ {
d: ...	t: ...
}	}

4- Accessing critical sections



Dekker's mutual exclusion protocol. Fails under Write to Read relaxation.

Memory Reordering Fences

- Write-Write Fences (wfence):

  Prevent reordering between writes.

- Read-Read Fences (rfence):

  Prevent reordering between reads.

- Fences (fence):

  Prevent reordering between any two memory operations.

Program Syntax

- Finite number of shared variables $\{x, y, x_1 ...\}$
- Finite data domain $\{d, d_1, d_2, ...\}$
- Finite number of finite-control processes $P_1, \ldots, P_n$ with operations:

$$\mathit{Write}(x, d), \mathit{Wfence}, \mathit{Read}(x, d), \mathit{Rfence}, \mathit{AtomicRW}(x, d_1, d_2)$$

Safety Verification Problem

For a memory model μ, a program P, and a (control + memory) state s

- State Reachability Problem (Safety)

  s is reachable in P ?

Safety Verification Problem

For a memory model μ, a program P, and a (control + memory) state s

- State Reachability Problem (Safety)

  s is reachable in P ?

Decidability / Complexity ?

Each process is finite-state

- For the SC memory model, this problem is PSPACE-complete

Safety Verification Problem

For a memory model μ, a program P, and a (control + memory) state s

- State Reachability Problem (Safety)

  s is reachable in P ?

Decidability / Complexity ?

Each process is finite-state

- For the SC memory model, this problem is PSPACE-complete
- Nontrivial for weak memory models:

  $Paths_\mu(P) = Closure_\mu(Paths_{SC}(P))$ *is nonregular*

Results for TSO [Atig, B., Burckhardt, Musuvathi, 2010]

- The state reachability problem is decidable for TSO.

Results for TSO [Atig, B., Burckhardt, Musuvathi, 2010]

- The state reachability problem is decidable for TSO.
- ... but highly complex: Nonprimitive recursive

Results for TSO [Atig, B., Burckhardt, Musuvathi, 2010]

- The state reachability problem is decidable for TSO.
- ... but highly complex: Nonprimitive recursive
- The repeated state reachability problem is undecidable for TSO

Results for TSO [Atig, B., Burckhardt, Musuvathi, 2010]

- The state reachability problem is decidable for TSO.
- ... but highly complex: Nonprimitive recursive
- The repeated state reachability problem is undecidable for TSO
- $\rightarrow$ *Store buffers can simulate lossy channels, and vice-versa.*

Decidability Frontier [Atig, B., Burckhardt, Musuvathi, 2012]

- The state reachability problem is undecidable for

  $TSO + R2W$

Decidability Frontier [Atig, B., Burckhardt, Musuvathi, 2012]

- The state reachability problem is undecidable for

  $TSO + R2W$

- The state reachability problem is decidable for

  $NSW = TSO + W2W + R2R$

Getting rid of Store Buffers [Atig, B., Parlato, 2011]

- When is it possible to reduce TSO verification to SC verification ?

Getting rid of Store Buffers [Atig, B., Parlato, 2011]

- When is it possible to reduce TSO verification to SC verification ?
- Find restrictions on the explored behaviors such that:

  Given a concurrent program P, it is possible to build a concurrent program P' such that: running P with TSO semantics under these restrictions is equivalent to running P' with SC semantics.

Getting rid of Store Buffers [Atig, B., Parlato, 2011]

- When is it possible to reduce TSO verification to SC verification ?
- Find restrictions on the explored behaviors such that:

  Given a concurrent program P, it is possible to build a concurrent program P' such that: running P with TSO semantics under these restrictions is equivalent to running P' with SC semantics.
- A notion of Context-Bounded Analysis for TSO

Getting rid of Store Buffers [Atig, B., Parlato, 2011]

- When is it possible to reduce TSO verification to SC verification ?
- Find restrictions on the explored behaviors such that:

  Given a concurrent program P, it is possible to build a concurrent program P' such that: running P with TSO semantics under these restrictions is equivalent to running P' with SC semantics.

- A notion of Context-Bounded Analysis for TSO
- Unbounded number of context-switches: Bounding the age of each write in the buffer in terms of context-switches.

Getting rid of Store Buffers [Atig, B., Parlato, 2011]

- When is it possible to reduce TSO verification to SC verification ?
- Find restrictions on the explored behaviors such that:

  Given a concurrent program P, it is possible to build a concurrent program P' such that: running P with TSO semantics under these restrictions is equivalent to running P' with SC semantics.
- A notion of Context-Bounded Analysis for TSO
- Unbounded number of context-switches: Bounding the age of each write in the buffer in terms of context-switches.
- $\Rightarrow$ Transfer decidability/complexity results from SC to TSO.
- $\Rightarrow$ Use existing tools for concurrent programs under SC.

The rest of the lecture

- Decidability and complexity for TSO:

  Simulations by/of Lossy Channel Systems

The rest of the lecture

- Decidability and complexity for TSO:

  Simulations by/of Lossy Channel Systems

- Decidability and complexity beyond TSO:
  - *Speculative writes lead to undecidability*
  - *Decidability: deal with reordered reads*

The rest of the lecture

- Decidability and complexity for TSO:

  Simulations by/of Lossy Channel Systems

- Decidability and complexity beyond TSO:
  - *Speculative writes lead to undecidability*
  - *Decidability: deal with reordered reads*

- From TSO to SC under bounded analysis
  - *2 notions of bounds*
  - *Store buffers $\rightsquigarrow$ 2K copies of the globals per thread*

An operational model for TSO

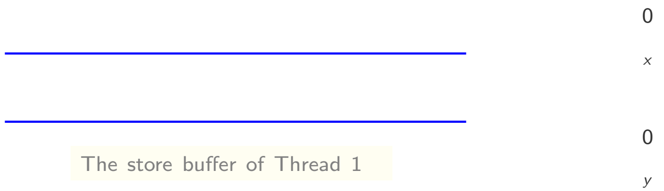
- Each process has a FIFO buffer
- Configuration = control states + memory state + buffers contents
- Write(x,d) is sent to the buffer
- Memory update = execution of a Write taken from some buffer
- Read(x,d) is executed either if
  - The last Write to x in the buffer is Write(x,d) (Read Own Write)
  - The buffer does not contain a Write to x, and $Memory(x) = d$
- $AtomicRW(x, d_1, d_2)$ requires that the buffer is empty ($\sim$ fence)

From W → R systems to Lossy Channel Systems

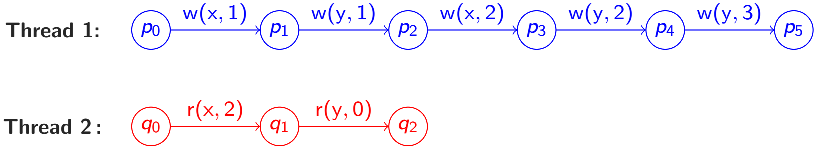
$$\begin{tikzcd}
\textbf{Thread 1:} & p_0 \arrow[r, "w(x,1)"] & p_1 \arrow[r, "w(y,1)"] & p_2 \arrow[r, "w(x,2)"] & p_3 \arrow[r, "w(y,2)"] & p_4 \arrow[r, "w(y,3)"] & p_5
\end{tikzcd}$$

$$\begin{tikzcd}
\textbf{Thread 2:} & q_0 \arrow[r, "r(x,2)"] & q_1 \arrow[r, "r(y,0)"] & q_2
\end{tikzcd}$$

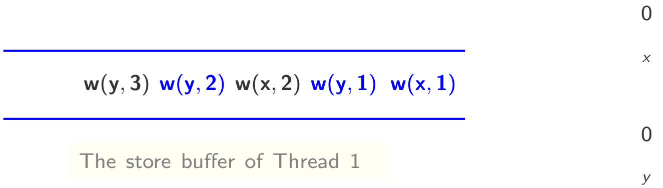
Model: The store buffers are considered as perfect FIFO channels



From W → R systems to Lossy Channel Systems



Model: The store buffers are considered as perfect FIFO channels


w(y, 3) w(y, 2) w(x, 2) w(y, 1) w(x, 1)
The store buffer of Thread 1
0
x
0
y

From W → R systems to Lossy Channel Systems

Thread 1:
$$\begin{tikzcd} p_0 \arrow[r, "w(x,1)"] & p_1 \arrow[r, "w(y,1)"] & p_2 \arrow[r, "w(x,2)"] & p_3 \arrow[r, "w(y,2)"] & p_4 \arrow[r, "w(y,3)"] & p_5 \end{tikzcd}$$

Thread 2:
$$\begin{tikzcd} q_0 \arrow[r, "r(x,2)"] & q_1 \arrow[r, "r(y,0)"] & q_2 \end{tikzcd}$$

Model: The store buffers are considered as perfect FIFO channels

w(y, 3) w(y, 2) w(x, 2) w(y, 1)

The store buffer of Thread 1

1

x

0

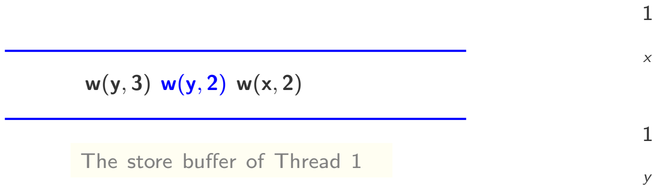
y

From W $\rightarrow$ R systems to Lossy Channel Systems

Thread 1:
$$\begin{tikzcd} p_0 \arrow[r, "w(x,1)"] & p_1 \arrow[r, "w(y,1)"] & p_2 \arrow[r, "w(x,2)"] & p_3 \arrow[r, "w(y,2)"] & p_4 \arrow[r, "w(y,3)"] & p_5 \end{tikzcd}$$

Thread 2:
$$\begin{tikzcd} q_0 \arrow[r, "r(x,2)"] & q_1 \arrow[r, "r(y,0)"] & q_2 \end{tikzcd}$$

Model: The store buffers are considered as perfect FIFO channels

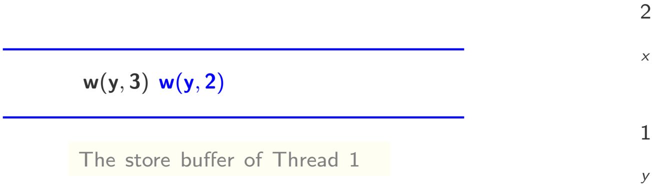

1
x
w(y,3) **w(y,2)** **w(x,2)**
1
The store buffer of Thread 1
y

From W → R systems to Lossy Channel Systems

Thread 1:
$$\begin{tikzcd} p_0 \arrow[r, "w(x,1)"] & p_1 \arrow[r, "w(y,1)"] & p_2 \arrow[r, "w(x,2)"] & p_3 \arrow[r, "w(y,2)"] & p_4 \arrow[r, "w(y,3)"] & p_5 \end{tikzcd}$$

Thread 2:
$$\begin{tikzcd} q_0 \arrow[r, "r(x,2)"] & q_1 \arrow[r, "r(y,0)"] & q_2 \end{tikzcd}$$

Model: The store buffers are considered as perfect FIFO channels

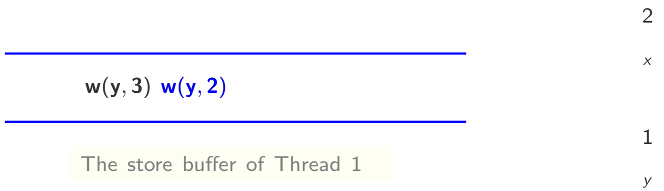


From W $\rightarrow$ R systems to Lossy Channel Systems

Thread 1: $$\begin{tikzcd} p_0 \arrow[r, "w(x,1)"] & p_1 \arrow[r, "w(y,1)"] & p_2 \arrow[r, "w(x,2)"] & p_3 \arrow[r, "w(y,2)"] & p_4 \arrow[r, "w(y,3)"] & p_5 \end{tikzcd}$$

Thread 2: $$\begin{tikzcd} q_0 \arrow[r, "r(x,2)"] & q_1 \arrow[r, "r(y,0)"] & q_2 \end{tikzcd}$$

Model: The store buffers are considered as perfect FIFO channels

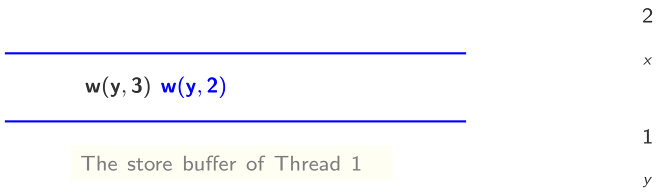


From W → R systems to Lossy Channel Systems

Thread 1: $$\begin{tikzcd} p_0 \arrow[r, "w(x,1)"] & p_1 \arrow[r, "w(y,1)"] & p_2 \arrow[r, "w(x,2)"] & p_3 \arrow[r, "w(y,2)"] & p_4 \arrow[r, "w(y,3)"] & p_5 \end{tikzcd}$$

Thread 2: $$\begin{tikzcd} q_0 \arrow[r, "r(x,2)"] & q_1 \arrow[r, "r(y,0)"] & q_2 \end{tikzcd}$$

Model: The store buffers are considered as perfect FIFO channels



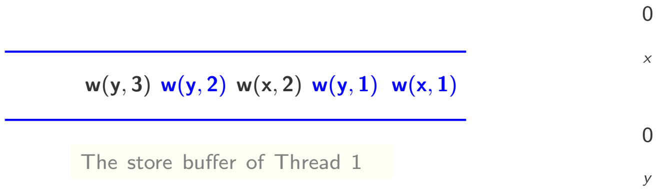
Deadlock

From W → R systems to Lossy Channel Systems

Thread 1:
$$\begin{tikzcd} p_0 \arrow[r, "w(x,1)"] & p_1 \arrow[r, "w(y,1)"] & p_2 \arrow[r, "w(x,2)"] & p_3 \arrow[r, "w(y,2)"] & p_4 \arrow[r, "w(y,3)"] & p_5 \end{tikzcd}$$

Thread 2:
$$\begin{tikzcd} q_0 \arrow[r, "r(x,2)"] & q_1 \arrow[r, "r(y,0)"] & q_2 \end{tikzcd}$$

Assume that the store buffers are lossy FIFO channels



0
x
w(y,3) w(y,2) w(x,2) w(y,1) w(x,1)
0
y
The store buffer of Thread 1

From W $\rightarrow$ R systems to Lossy Channel Systems

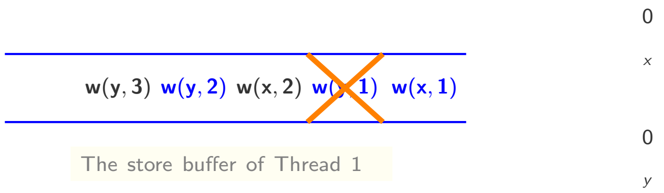
Thread 1:

$$\begin{tikzcd} p_0 \arrow[r, "w(x,1)"] & p_1 \arrow[r, "w(y,1)"] & p_2 \arrow[r, "w(x,2)"] & p_3 \arrow[r, "w(y,2)"] & p_4 \arrow[r, "w(y,3)"] & p_5 \end{tikzcd}$$

Thread 2:

$$\begin{tikzcd} q_0 \arrow[r, "r(x,2)"] & q_1 \arrow[r, "r(y,0)"] & q_2 \end{tikzcd}$$

Assume that the store buffers are lossy FIFO channels



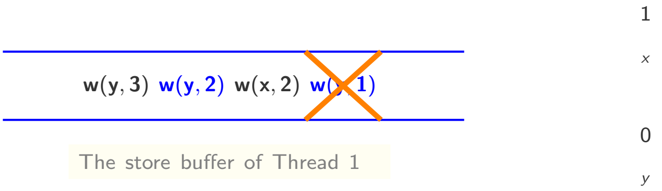
The store buffer of Thread 1

From W → R systems to Lossy Channel Systems

Thread 1:
$$\begin{tikzcd} p_0 \arrow[r, "w(x,1)"] & p_1 \arrow[r, "w(y,1)"] & p_2 \arrow[r, "w(x,2)"] & p_3 \arrow[r, "w(y,2)"] & p_4 \arrow[r, "w(y,3)"] & p_5 \end{tikzcd}$$

Thread 2:
$$\begin{tikzcd} q_0 \arrow[r, "r(x,2)"] & q_1 \arrow[r, "r(y,0)"] & q_2 \end{tikzcd}$$

Assume that the store buffers are lossy FIFO channels

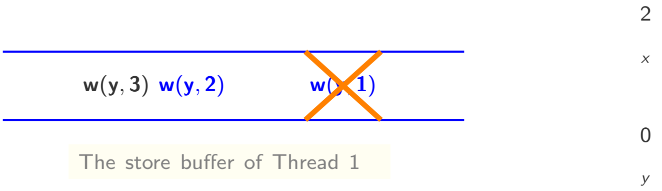


From W → R systems to Lossy Channel Systems

Thread 1: $$\begin{tikzcd} p_0 \arrow[r, "w(x,1)"] & p_1 \arrow[r, "w(y,1)"] & p_2 \arrow[r, "w(x,2)"] & p_3 \arrow[r, "w(y,2)"] & p_4 \arrow[r, "w(y,3)"] & p_5 \end{tikzcd}$$

Thread 2 : $$\begin{tikzcd} q_0 \arrow[r, "r(x,2)"] & q_1 \arrow[r, "r(y,0)"] & q_2 \end{tikzcd}$$

Assume that the store buffers are lossy FIFO channels

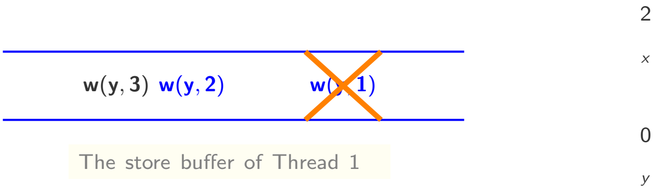


From W $\rightarrow$ R systems to Lossy Channel Systems

Thread 1:
$$\begin{tikzcd} p_0 \arrow[r, "\mathrm{w}(x,1)"] & p_1 \arrow[r, "\mathrm{w}(y,1)"] & p_2 \arrow[r, "\mathrm{w}(x,2)"] & p_3 \arrow[r, "\mathrm{w}(y,2)"] & p_4 \arrow[r, "\mathrm{w}(y,3)"] & p_5 \end{tikzcd}$$

Thread 2:
$$\begin{tikzcd} q_0 \arrow[r, "\mathrm{r}(x,2)"] & q_1 \arrow[r, "\mathrm{r}(y,0)"] & q_2 \end{tikzcd}$$

Assume that the store buffers are lossy FIFO channels



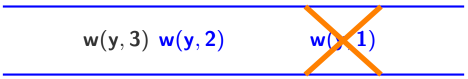
The store buffer of Thread 1

From W → R systems to Lossy Channel Systems

Thread 1: $$\begin{tikzcd} p_0 \arrow[r, "w(x,1)"] & p_1 \arrow[r, "w(y,1)"] & p_2 \arrow[r, "w(x,2)"] & p_3 \arrow[r, "w(y,2)"] & p_4 \arrow[r, "w(y,3)"] & p_5 \end{tikzcd}$$

Thread 2: $$\begin{tikzcd} q_0 \arrow[r, "r(x,2)"] & q_1 \arrow[r, "r(y,0)"] & q_2 \end{tikzcd}$$

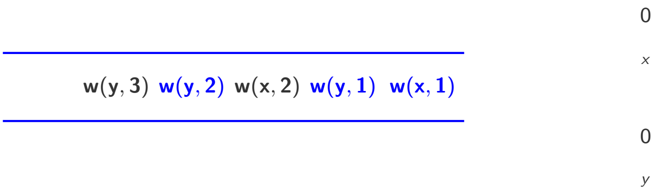
Assume that the store buffers are lossy FIFO channels



The store buffer of Thread 1

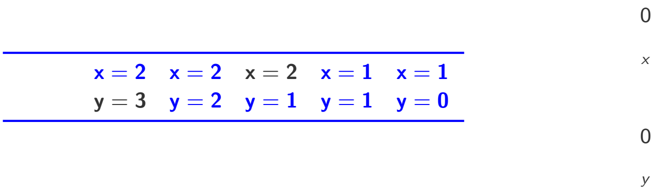
From W $\rightarrow$ R systems to Lossy Channel Systems

Buffer = perfect FIFO channel



$\mathbf{w(y,3)}$ $\mathbf{w(y,2)}$ $\mathbf{w(x,2)}$ $\mathbf{w(y,1)}$ $\mathbf{w(x,1)}$
0
x
0
y

Channel= Sequence of memory states + Lossyness



$\mathbf{x=2}$ $\mathbf{x=2}$ $\mathbf{x=2}$ $\mathbf{x=1}$ $\mathbf{x=1}$
$\mathbf{y=3}$ $\mathbf{y=2}$ $\mathbf{y=1}$ $\mathbf{y=1}$ $\mathbf{y=0}$
0
x
0
y

From W $\rightarrow$ R systems to Lossy Channel Systems

Buffer = perfect FIFO channel

$\mathbf{w(y,3)}$ $\mathbf{w(y,2)}$ $\mathbf{w(x,2)}$ $\mathbf{w(y,1)}$ $\mathbf{w(x,1)}$

0

x

0

y

Channel= Sequence of memory states + Lossyness

$\mathbf{x=2}$	$x=2$	$\mathbf{x=2}$	$x=1$	$x=1$
$\mathbf{y=3}$	$y=2$	$\mathbf{y=1}$	$y=1$	$y=0$

0

x

0

y

Lossyness= Unobservable memory states

From W → R systems to Lossy Channel Systems

Buffer = perfect FIFO channel

0

x

w(y, 3) **w(y, 2)** **w(x, 2)** **w(y, 1)** **w(x, 1)**

0

y

Channel= Sequence of memory states + Lossyness

0

x

x = 2	x = 2	**x = 2**	~~x = 1~~	x = 1
y = 3	y = 2	**y = 1**	~~y = 1~~	y = 0

0

y

Lossyness= Unobservable memory states

From W $\rightarrow$ R systems to Lossy Channel Systems

Buffer = perfect FIFO channel

$$\mathbf{w(y,3)}\;\; \mathbf{w(y,2)}\;\; \mathbf{w(x,2)}\;\; \mathbf{w(y,1)}\;\; \mathbf{w(x,1)}$$

0

x

0

y

Channel= Sequence of memory states + Lossyness

$$\begin{array}{cccc} \mathbf{x=2} & x=2 & \mathbf{x=2} & \cancel{x=1} \\ \mathbf{y=3} & y=2 & \mathbf{y=1} & \cancel{y=1} \end{array}$$

1

x

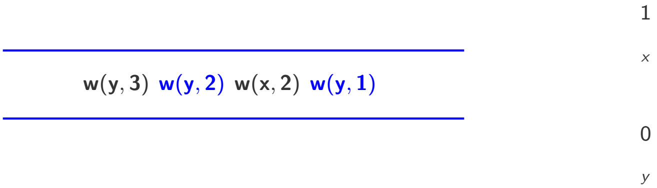
0

y

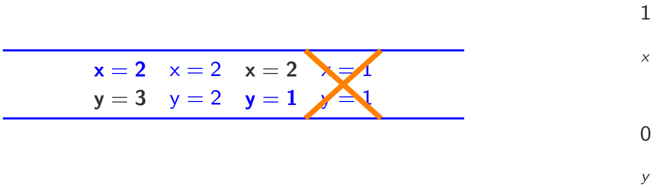
Lossyness= Unobservable memory states

From W → R systems to Lossy Channel Systems

Buffer = perfect FIFO channel



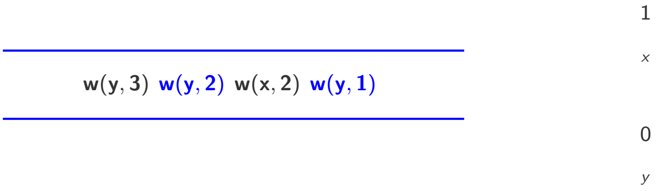
Channel= Sequence of memory states + Lossyness



Lossyness= Unobservable memory states

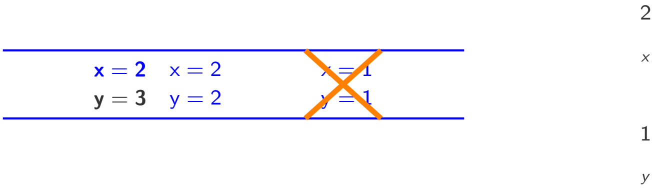
From W $\rightarrow$ R systems to Lossy Channel Systems

Buffer = perfect FIFO channel



1
x
$\mathbf{w(y,3)}$ $\mathbf{w(y,2)}$ $\mathbf{w(x,2)}$ $\mathbf{w(y,1)}$
0
y

Channel= Sequence of memory states + Lossyness



2
x
$\mathbf{x=2}$ $x=2$ $\cancel{x=1}$
$\mathbf{y=3}$ $y=2$ $\cancel{y=1}$
1
y

Lossyness= Unobservable memory states

From W $\rightarrow$ R systems to Lossy Channel Systems

Buffer = perfect FIFO channel

$\mathbf{w(y, 3)}$ $\mathbf{w(y, 2)}$ $\mathbf{w(x, 2)}$

1

x

1

y

Channel= Sequence of memory states + Lossyness

$\mathbf{x = 2}$ $x = 2$ $\cancel{x = 1}$

$\mathbf{y = 3}$ $y = 2$ $\cancel{y = 1}$

2

x

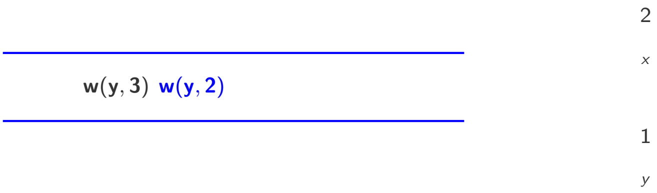
1

y

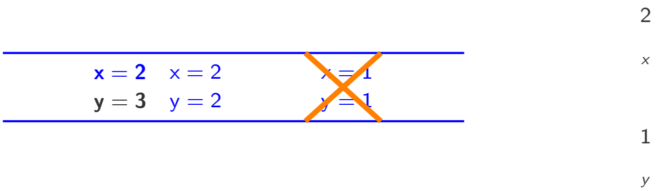
Lossyness= Unobservable memory states

From W → R systems to Lossy Channel Systems

Buffer = perfect FIFO channel

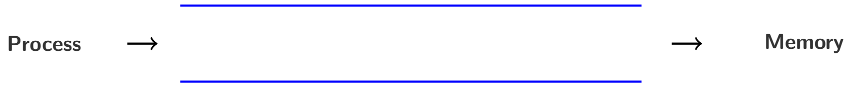


Channel= Sequence of memory states + Lossyness



Lossyness= Unobservable memory states

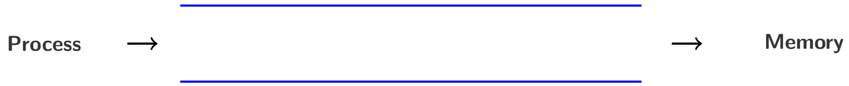
From W $\rightarrow$ R systems to Lossy Channel Systems



- *Write:* *Compute a new memory state; send it to the channel*
- *Read:* *Check the channel/memory*
- *Memory update:* *Receive a state; copy it to the memory*

From W $\rightarrow$ R systems to Lossy Channel Systems

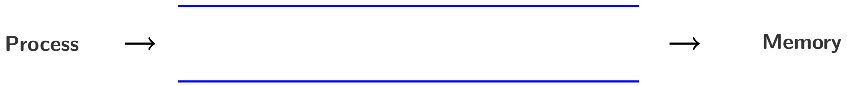
- Problem: Interference between processes ?



- *Write: Compute a new memory state; send it to the channel*
- *Read: Check the channel/memory*
- *Memory update: Receive a state; copy it to the memory*

From W $\rightarrow$ R systems to Lossy Channel Systems

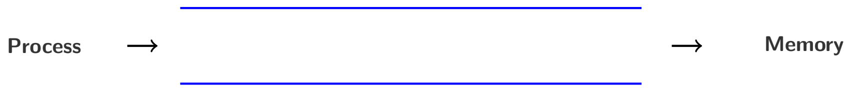
- Problem: Interference between processes ?
- $\Rightarrow$ Each process guesses occurrences of writes by other processes



- *Write: Compute a new memory state; send it to the channel*
- *Read: Check the channel/memory*
- *Memory update: Receive a state; copy it to the memory*
- *Guessed Write: Send the guessed state to the channel*

From W $\rightarrow$ R systems to Lossy Channel Systems

- Problem: Interference between processes ?
- $\Rightarrow$ Each process guesses occurrences of writes by other processes



  - *Write: Compute a new memory state; send it to the channel*
  - *Read: Check the channel/memory*
  - *Memory update: Receive a state; copy it to the memory*
  - *Guessed Write: Send the guessed state to the channel*

- $\Rightarrow$ Check that all process agree on the sequence of states

  Synchronization of the lossy channel machines over send actions

Decidability for the State Reachability Problem

- Thm

  The state reachability problem for TSO programs is reducible to the control-state reachability problem for LCS.

Decidability for the State Reachability Problem

- Thm

  The state reachability problem for TSO programs is reducible to the control-state reachability problem for LCS.

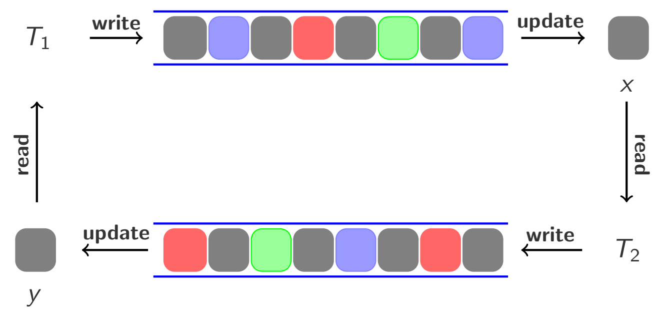
- Thm ([Abdulla, Jonsson, 1993])

  The control-state reachability problem for LCS is decidable

- Corollary

  The state reachability problem for TSO systems is decidable.

From Lossy Channel Systems to W $\rightarrow$ R systems



- T_1 simulates the lossy channel machine:
  - Send operation: Write operation of T_1 to the variable x
  - Read operation: Read operation of T_1 from the variable y
- T_2 transfers the successive values of the variable x to the variable y

Complexity

- **Thm**

  Every LCS can be simulated by a TSO program.

Complexity

- **Thm**

  Every LCS can be simulated by a TSO program.

- **Thm** ([Schnoebelen, 2001])

  The control-state reachability problem for LCS is non-primitive recursive

$\Rightarrow$ Lower bound for the state reachability problem under TSO.

TSO + R2W: Causality cycles

$x = y = 0$	
$\mathcal{P}_1$	$\mathcal{P}_2$
(1) r$(x, 1)$	(3) r$(y, 1)$
(2) w$(y, 1)$	(4) w$(x, 1)$
$x = y = 1$	

TSO + R2W: Causality cycles

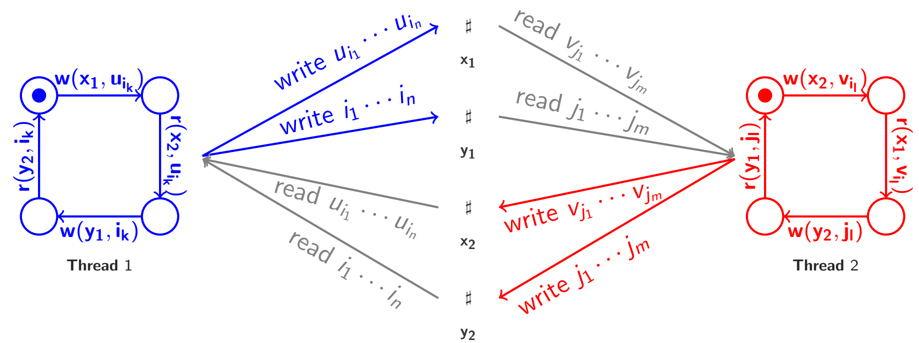
$x = y = 0$	
$\mathcal{P}_1$	$\mathcal{P}_2$
(1) $\mathsf{r}(x, 1)$	(3) $\mathsf{r}(y, 1)$
(2) $\mathsf{w}(y, 1)$	(4) $\mathsf{w}(x, 1)$
$x = y = 1$	

- This behavior is possible since writes can overtake reads:

  $$(2), (3), (4), (1)$$

- Speculative writes $\Rightarrow$ causality cycles
  - (2) is executed assuming that (1) will be executed in the future
  - (1) is indeed executed, but it is based on a write that depends from (2)

TSO + R2W: Undecidabiity

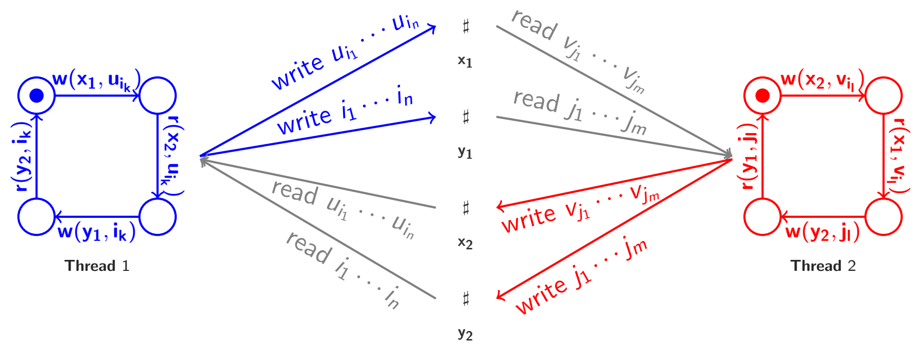


Assume that: $u_{i_1} u_{i_2} \cdots u_{i_n} = v_{j_1} v_{j_2} \cdots v_{j_m}$ and $i_1 i_2 \cdots i_n = j_1 j_2 \cdots j_m$

T_1: $r(y_2, i_n)\ w(y_1, i_n)\ r(x_2, u_{i_n})\ w(x_1, u_{i_n}) \cdots r(y_2, i_1)\ w(y_1, i_1)\ r(x_2, u_{i_1})\ w(x_1, u_{i_1})$

T_2: $r(y_1, j_n)\ w(y_2, j_n)\ r(x_1, v_{j_n})\ w(x_2, v_{j_n}) \cdots r(y_1, j_1)\ w(y_2, j_1)\ r(x_1, v_{j_1})\ w(x_2, v_{j_1})$

TSO + R2W: Undecidabiity

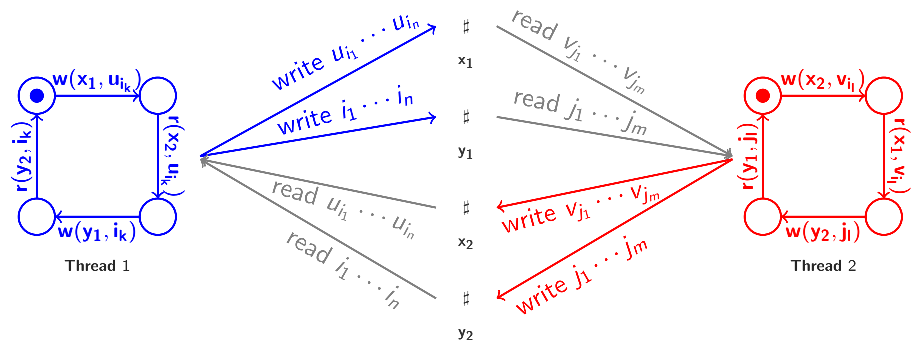


Assume that: $u_{i_1} u_{i_2} \cdots u_{i_n} = v_{j_1} v_{j_2} \cdots v_{j_m}$ and $i_1 i_2 \cdots i_n = j_1 j_2 \cdots j_m$

T_1: $r(y_2, i_n)\ r(x_2, u_{i_n}) \cdots r(y_2, i_1)\ r(x_2, u_{i_1}) \cdots w(y_1, i_n)\ w(x_1, u_{i_n}) \cdots w(y_1, i_1)\ w(x_1, u_{i_1})$

T_2: $w(y_2, j_n)\ w(x_2, v_{j_n}) \cdots w(y_2, j_1)\ w(x_2, v_{j_1}) \cdots r(y_1, j_n)\ r(x_1, v_{j_n}) \cdots r(y_1, j_1)\ r(x_1, v_{j_1})$

TSO + R2W: Undecidabiity



Assume that: $u_{i_1} u_{i_2} \cdots u_{i_n} = v_{j_1} v_{j_2} \cdots v_{j_m}$ and $i_1 i_2 \cdots i_n = j_1 j_2 \cdots j_m$

T_1: $r(y_2, i_n)$ $r(x_2, u_{i_n})$ $\cdots$ $r(y_2, i_1)$ $r(x_2, u_{i_1})$ $\cdots$ $w(y_1, i_n)$ $w(x_1, u_{i_n})$ $\cdots$ $w(y_1, i_1)$ $w(x_1, u_{i_1})$

T_2: $w(y_2, j_n)$ $w(x_2, v_{j_n})$ $\cdots$ $w(y_2, j_1)$ $w(x_2, v_{j_1})$ $\cdots$ $r(y_1, j_n)$ $r(x_1, v_{j_n})$ $\cdots$ $r(y_1, j_1)$ $r(x_1, v_{j_1})$

$\Rightarrow$ Reachability TSO + R2W

NSW: Non Speculative Writes

- TSO = Read-Local-Write-Early + W2R
- PSO = TSO + W2W
- NSW = PSO + R2R

NSW: Non Speculative Writes

- TSO = Read-Local-Write-Early + W2R
- PSO = TSO + W2W
- NSW = PSO + R2R

- Simulation of TSO under PSO:
  *Add a write-write fence (*wfence*) before each write*

NSW: Non Speculative Writes

- TSO = Read-Local-Write-Early + W2R
- PSO = TSO + W2W
- NSW = PSO + R2R

- Simulation of TSO under PSO:
  *Add a write-write fence (*wfence*) before each write*

- Simulation of PSO under NSW:
  *Add a read-read fence (*rfence*) before each read*

Operational Model: Event Structures

$$\begin{tikzcd}
\textbf{Process 1:} & p_0 \arrow[r, "\mathsf{w}(x,1)"] & p_1 \arrow[r, "\mathsf{w}(y,1)"] & p_2 \arrow[r, "\mathsf{wfence}"] & p_3 \arrow[r, "\mathsf{r}(x,2)"] & p_4 \arrow[r, "\mathsf{w}(y,2)"] & p_5 \\
\textbf{Process 2:} & q_0 \arrow[r, "\mathsf{fence}"] & q_1 \arrow[r, "\mathsf{r}(y,1)"] & q_2 \arrow[r, "\mathsf{w}(x,2)"] & q_3 \arrow[r, "\mathsf{rfence}"] & q_4 \arrow[r, "\mathsf{r}(x,2)"] & q_5
\end{tikzcd}$$

Operational Model: Event Structures

$$\begin{tikzcd}
\textbf{Process 1:} & p_0 \arrow[r, "w(x,1)"] & p_1 \arrow[r, "w(y,1)"] & p_2 \arrow[r, "wfence"] & p_3 \arrow[r, "r(x,2)"] & p_4 \arrow[r, "w(y,2)"] & p_5 \\
\textbf{Process 2:} & q_0 \arrow[r, "fence"] & q_1 \arrow[r, "r(y,1)"] & q_2 \arrow[r, "w(x,2)"] & q_3 \arrow[r, "rfence"] & q_4 \arrow[r, "r(x,2)"] & q_5
\end{tikzcd}$$

Configuration= control states + memory state + event structures

p_0

$x = 0$

$y = 0$

q_0

Operational Model: Event Structures

$$\begin{tikzcd}
\textbf{Process 1:} & p_0 \arrow[r, "w(x,1)"] & p_1 \arrow[r, "w(y,1)"] & p_2 \arrow[r, "wfence"] & p_3 \arrow[r, "r(x,2)"] & p_4 \arrow[r, "w(y,2)"] & p_5 \\
\textbf{Process 2:} & q_0 \arrow[r, "fence"] & q_1 \arrow[r, "r(y,1)"] & q_2 \arrow[r, "w(x,2)"] & q_3 \arrow[r, "rfence"] & q_4 \arrow[r, "r(x,2)"] & q_5
\end{tikzcd}$$

Writes on x are inserted after the last reads, wfences, and writes on x.

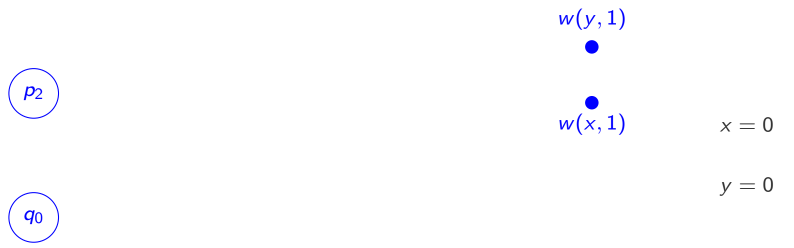
p_1
q_0
$w(x,1)$
$x = 0$
$y = 0$

Operational Model: Event Structures

Process 1: $$\begin{tikzcd} p_0 \arrow[r, "w(x,1)"] & p_1 \arrow[r, "w(y,1)"] & p_2 \arrow[r, "wfence"] & p_3 \arrow[r, "r(x,2)"] & p_4 \arrow[r, "w(y,2)"] & p_5 \end{tikzcd}$$

Process 2: $$\begin{tikzcd} q_0 \arrow[r, "fence"] & q_1 \arrow[r, "r(y,1)"] & q_2 \arrow[r, "w(x,2)"] & q_3 \arrow[r, "rfence"] & q_4 \arrow[r, "r(x,2)"] & q_5 \end{tikzcd}$$

Writes on y are inserted after the last reads, wfences, and writes on y.

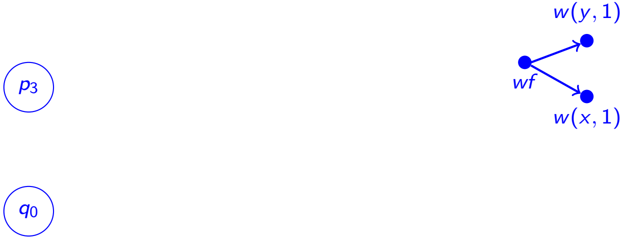


Operational Model: Event Structures

Process 1: $$\begin{tikzcd} p_0 \arrow[r, "w(x,1)"] & p_1 \arrow[r, "w(y,1)"] & p_2 \arrow[r, "wfence"] & p_3 \arrow[r, "r(x,2)"] & p_4 \arrow[r, "w(y,2)"] & p_5 \end{tikzcd}$$

Process 2: $$\begin{tikzcd} q_0 \arrow[r, "fence"] & q_1 \arrow[r, "r(y,1)"] & q_2 \arrow[r, "w(x,2)"] & q_3 \arrow[r, "rfence"] & q_4 \arrow[r, "r(x,2)"] & q_5 \end{tikzcd}$$

Wfences are inserted after the last writes.



$x = 0$

$y = 0$

Operational Model: Event Structures

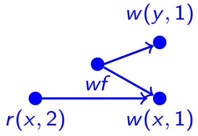
$$\begin{tikzcd}
\textbf{Process 1:} & p_0 \arrow[r, "w(x,1)"] & p_1 \arrow[r, "w(y,1)"] & p_2 \arrow[r, "wfence"] & p_3 \arrow[r, "r(x,2)"] & p_4 \arrow[r, "w(y,2)"] & p_5
\end{tikzcd}$$

$$\begin{tikzcd}
\textbf{Process 2:} & q_0 \arrow[r, "fence"] & q_1 \arrow[r, "r(y,1)"] & q_2 \arrow[r, "w(x,2)"] & q_3 \arrow[r, "rfence"] & q_4 \arrow[r, "r(x,2)"] & q_5
\end{tikzcd}$$

Reads on x are inserted after the last writes/reads on x.

p_4

q_0



$x = 0$

$y = 0$

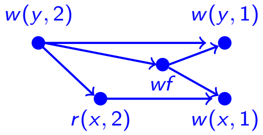
Operational Model: Event Structures

Process 1: $$\begin{tikzcd} p_0 \arrow[r, "\mathrm{w}(x,1)"] & p_1 \arrow[r, "\mathrm{w}(y,1)"] & p_2 \arrow[r, "\mathrm{wfence}"] & p_3 \arrow[r, "\mathrm{r}(x,2)"] & p_4 \arrow[r, "\mathrm{w}(y,2)"] & p_5 \end{tikzcd}$$

Process 2: $$\begin{tikzcd} q_0 \arrow[r, "\mathrm{fence}"] & q_1 \arrow[r, "\mathrm{r}(y,1)"] & q_2 \arrow[r, "\mathrm{w}(x,2)"] & q_3 \arrow[r, "\mathrm{rfence}"] & q_4 \arrow[r, "\mathrm{r}(x,2)"] & q_5 \end{tikzcd}$$

Writes on y are inserted after the last reads, wfences, and writes on y.

p_5



$x = 0$

$y = 0$

q_0

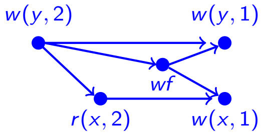
Operational Model: Event Structures

Process 1: $$\begin{tikzcd} p_0 \arrow[r, "w(x,1)"] & p_1 \arrow[r, "w(y,1)"] & p_2 \arrow[r, "wfence"] & p_3 \arrow[r, "r(x,2)"] & p_4 \arrow[r, "w(y,2)"] & p_5 \end{tikzcd}$$

Process 2: $$\begin{tikzcd} q_0 \arrow[r, "fence"] & q_1 \arrow[r, "r(y,1)"] & q_2 \arrow[r, "w(x,2)"] & q_3 \arrow[r, "rfence"] & q_4 \arrow[r, "r(x,2)"] & q_5 \end{tikzcd}$$

Fences are performed by a process only when its event structure is empty.

p_5



$x = 0$

q_1

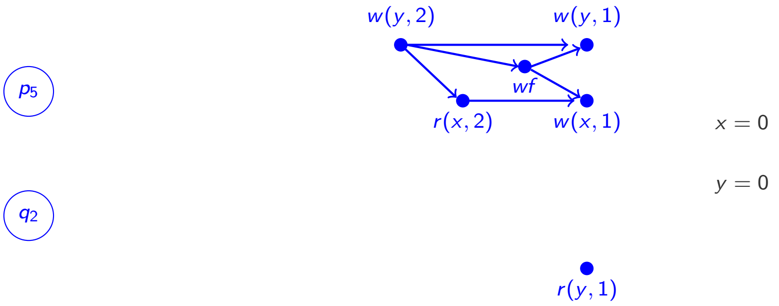
$y = 0$

Operational Model: Event Structures

$$\begin{tikzcd}
\textbf{Process 1:} & p_0 \arrow[r, "w(x,1)"] & p_1 \arrow[r, "w(y,1)"] & p_2 \arrow[r, "wfence"] & p_3 \arrow[r, "r(x,2)"] & p_4 \arrow[r, "w(y,2)"] & p_5
\end{tikzcd}$$

$$\begin{tikzcd}
\textbf{Process 2:} & q_0 \arrow[r, "fence"] & q_1 \arrow[r, "r(y,1)"] & q_2 \arrow[r, "w(x,2)"] & q_3 \arrow[r, "rfence"] & q_4 \arrow[r, "r(x,2)"] & q_5
\end{tikzcd}$$

Reads on y are inserted after the last writes/reads on y.

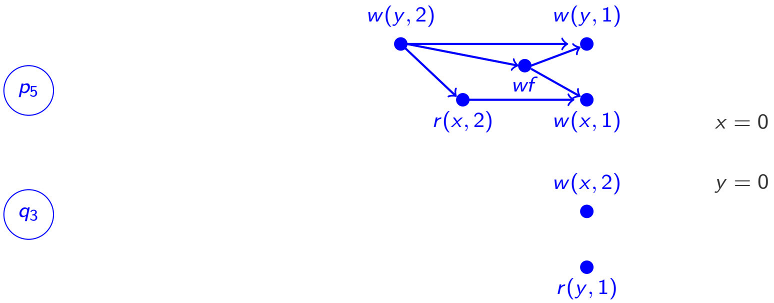


Operational Model: Event Structures

Process 1: $$\begin{tikzcd} p_0 \arrow[r, "\mathrm{w}(x,1)"] & p_1 \arrow[r, "\mathrm{w}(y,1)"] & p_2 \arrow[r, "\mathrm{wfence}"] & p_3 \arrow[r, "\mathrm{r}(x,2)"] & p_4 \arrow[r, "\mathrm{w}(y,2)"] & p_5 \end{tikzcd}$$

Process 2: $$\begin{tikzcd} q_0 \arrow[r, "\mathrm{fence}"] & q_1 \arrow[r, "\mathrm{r}(y,1)"] & q_2 \arrow[r, "\mathrm{w}(x,2)"] & q_3 \arrow[r, "\mathrm{rfence}"] & q_4 \arrow[r, "\mathrm{r}(x,2)"] & q_5 \end{tikzcd}$$

Writes on x are inserted after the last reads, wfences, and writes on x.

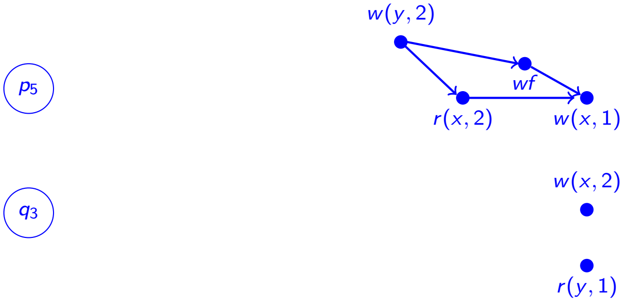


Operational Model: Event Structures

$$\begin{tikzcd}
\textbf{Process 1:} & p_0 \arrow[r, "w(x,1)"] & p_1 \arrow[r, "w(y,1)"] & p_2 \arrow[r, "wfence"] & p_3 \arrow[r, "r(x,2)"] & p_4 \arrow[r, "w(y,2)"] & p_5
\end{tikzcd}$$

$$\begin{tikzcd}
\textbf{Process 2:} & q_0 \arrow[r, "fence"] & q_1 \arrow[r, "r(y,1)"] & q_2 \arrow[r, "w(x,2)"] & q_3 \arrow[r, "rfence"] & q_4 \arrow[r, "r(x,2)"] & q_5
\end{tikzcd}$$

Updates to memory are performed when those writes are minimal.



$x = 0$

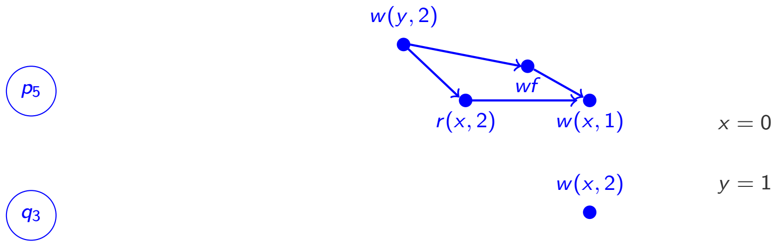
$y = 1$

Operational Model: Event Structures

$$\begin{tikzcd}
\textbf{Process 1:} & p_0 \arrow[r, "w(x,1)"] & p_1 \arrow[r, "w(y,1)"] & p_2 \arrow[r, "wfence"] & p_3 \arrow[r, "r(x,2)"] & p_4 \arrow[r, "w(y,2)"] & p_5
\end{tikzcd}$$

$$\begin{tikzcd}
\textbf{Process 2:} & q_0 \arrow[r, "fence"] & q_1 \arrow[r, "r(y,1)"] & q_2 \arrow[r, "w(x,2)"] & q_3 \arrow[r, "rfence"] & q_4 \arrow[r, "r(x,2)"] & q_5
\end{tikzcd}$$

Reads are validated w.r.t. the memory when they are minimal.

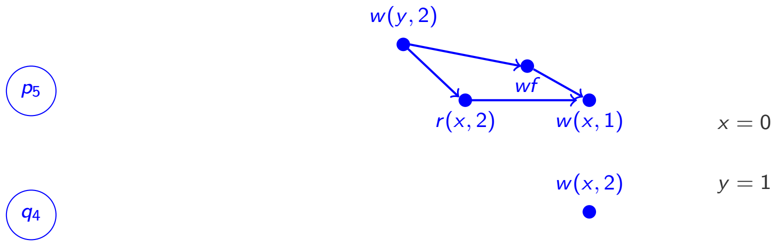


Operational Model: Event Structures

Process 1: $$\begin{tikzcd} p_0 \arrow[r, "\mathrm{w(x,1)}"] & p_1 \arrow[r, "\mathrm{w(y,1)}"] & p_2 \arrow[r, "\mathrm{wfence}"] & p_3 \arrow[r, "\mathrm{r(x,2)}"] & p_4 \arrow[r, "\mathrm{w(y,2)}"] & p_5 \end{tikzcd}$$

Process 2: $$\begin{tikzcd} q_0 \arrow[r, "\mathrm{fence}"] & q_1 \arrow[r, "\mathrm{r(y,1)}"] & q_2 \arrow[r, "\mathrm{w(x,2)}"] & q_3 \arrow[r, "\mathrm{rfence}"] & q_4 \arrow[r, "\mathrm{r(x,2)}"] & q_5 \end{tikzcd}$$

Rfences are performed by a process only if there is no pending reads.

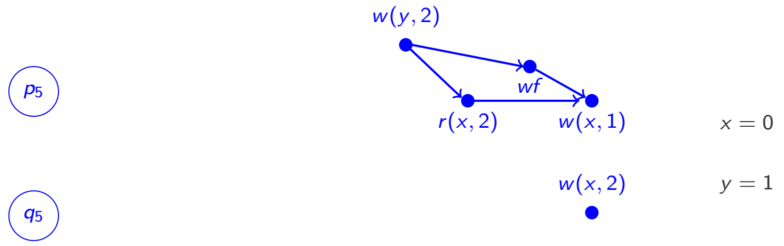


Operational Model: Event Structures

$$\begin{tikzcd}[column sep=large] \textbf{Process 1:} & p_0 \arrow[r, "\mathrm{w}(x,1)"] & p_1 \arrow[r, "\mathrm{w}(y,1)"] & p_2 \arrow[r, "\mathrm{wfence}"] & p_3 \arrow[r, "\mathrm{r}(x,2)"] & p_4 \arrow[r, "\mathrm{w}(y,2)"] & p_5 \end{tikzcd}$$

$$\begin{tikzcd}[column sep=large] \textbf{Process 2:} & q_0 \arrow[r, "\mathrm{fence}"] & q_1 \arrow[r, "\mathrm{r}(y,1)"] & q_2 \arrow[r, "\mathrm{w}(x,2)"] & q_3 \arrow[r, "\mathrm{rfence}"] & q_4 \arrow[r, "\mathrm{r}(x,2)"] & q_5 \end{tikzcd}$$

Reads on x are validated immediately with the last write on x (if possible)

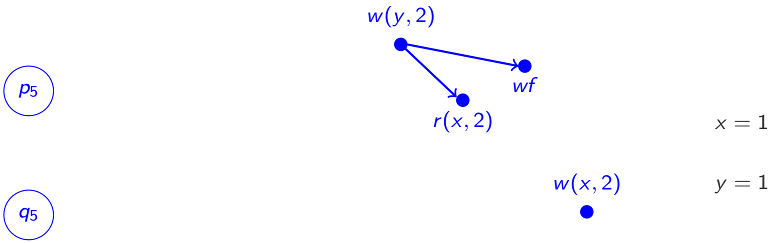


Operational Model: Event Structures

$$\begin{tikzcd}
\textbf{Process 1:} & p_0 \arrow[r, "\mathrm{w}(x,1)"] & p_1 \arrow[r, "\mathrm{w}(y,1)"] & p_2 \arrow[r, "\mathrm{wfence}"] & p_3 \arrow[r, "\mathrm{r}(x,2)"] & p_4 \arrow[r, "\mathrm{w}(y,2)"] & p_5
\end{tikzcd}$$

$$\begin{tikzcd}
\textbf{Process 2:} & q_0 \arrow[r, "\mathrm{fence}"] & q_1 \arrow[r, "\mathrm{r}(y,1)"] & q_2 \arrow[r, "\mathrm{w}(x,2)"] & q_3 \arrow[r, "\mathrm{rfence}"] & q_4 \arrow[r, "\mathrm{r}(x,2)"] & q_5
\end{tikzcd}$$

Updates to memory are performed when those writes are minimal.



Operational Model: Event Structures

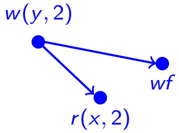
Process 1: $$\begin{tikzcd} p_0 \arrow[r, "\mathrm{w}(x,1)"] & p_1 \arrow[r, "\mathrm{w}(y,1)"] & p_2 \arrow[r, "\mathrm{wfence}"] & p_3 \arrow[r, "\mathrm{r}(x,2)"] & p_4 \arrow[r, "\mathrm{w}(y,2)"] & p_5 \end{tikzcd}$$

Process 2: $$\begin{tikzcd} q_0 \arrow[r, "\mathrm{fence}"] & q_1 \arrow[r, "\mathrm{r}(y,1)"] & q_2 \arrow[r, "\mathrm{w}(x,2)"] & q_3 \arrow[r, "\mathrm{rfence}"] & q_4 \arrow[r, "\mathrm{r}(x,2)"] & q_5 \end{tikzcd}$$

Updates to memory are performed when those writes are minimal.

p_5

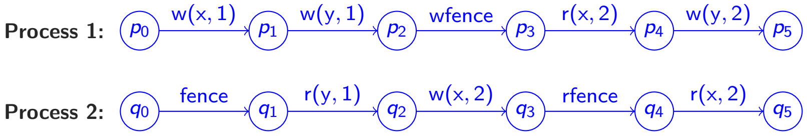
q_5



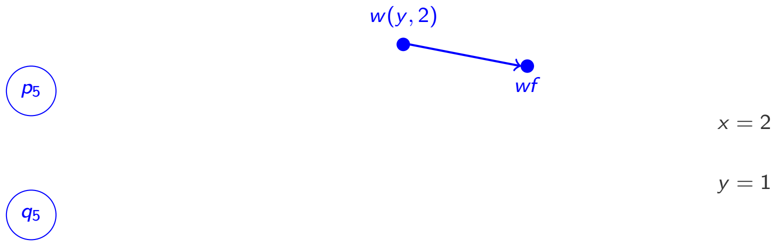
$x = 2$

$y = 1$

Operational Model: Event Structures



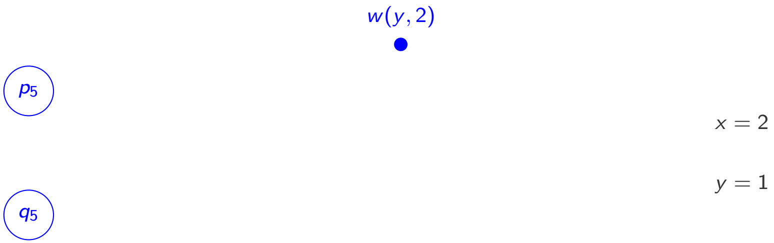
Reads are validated w.r.t. the memory when they are minimal.



Operational Model: Event Structures

$$\begin{tikzcd}
\textbf{Process 1:} & p_0 \arrow[r, "w(x,1)"] & p_1 \arrow[r, "w(y,1)"] & p_2 \arrow[r, "wfence"] & p_3 \arrow[r, "r(x,2)"] & p_4 \arrow[r, "w(y,2)"] & p_5 \\
\textbf{Process 2:} & q_0 \arrow[r, "fence"] & q_1 \arrow[r, "r(y,1)"] & q_2 \arrow[r, "w(x,2)"] & q_3 \arrow[r, "rfence"] & q_4 \arrow[r, "r(x,2)"] & q_5
\end{tikzcd}$$

Wfences are removed if they are minimal.



Operational Model: Event Structures

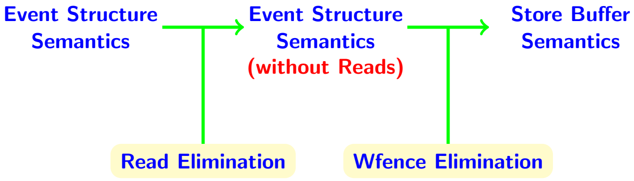
$$\begin{tikzcd}
\textbf{Process 1:} & p_0 \arrow[r, "w(x,1)"] & p_1 \arrow[r, "w(y,1)"] & p_2 \arrow[r, "wfence"] & p_3 \arrow[r, "r(x,2)"] & p_4 \arrow[r, "w(y,2)"] & p_5 \\
\textbf{Process 2:} & q_0 \arrow[r, "fence"] & q_1 \arrow[r, "r(y,1)"] & q_2 \arrow[r, "w(x,2)"] & q_3 \arrow[r, "rfence"] & q_4 \arrow[r, "r(x,2)"] & q_5
\end{tikzcd}$$

Updates to memory are performed when those writes are minimal.

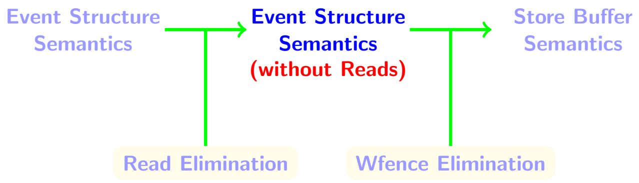
$x = 2$

$y = 2$

From Event Structures to Buffers

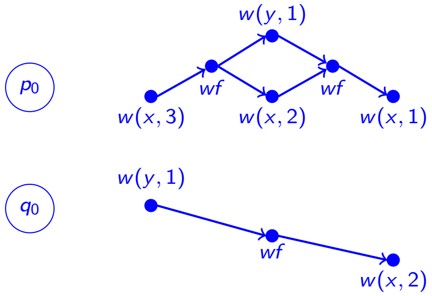

Event Structure Semantics
Event Structure Semantics (without Reads)
Store Buffer Semantics
Read Elimination
Wfence Elimination

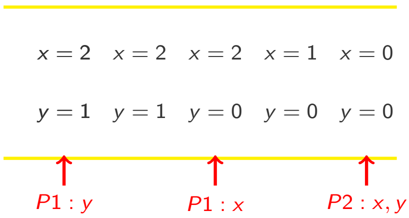
From Event Structures to Buffers



Elimination of Reads

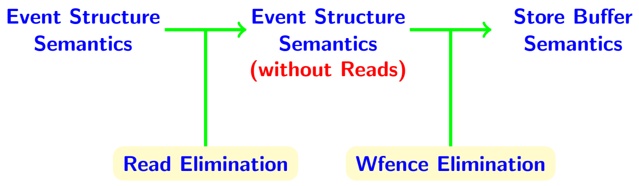
Configuration= control states + event structures+ memory history buffer.



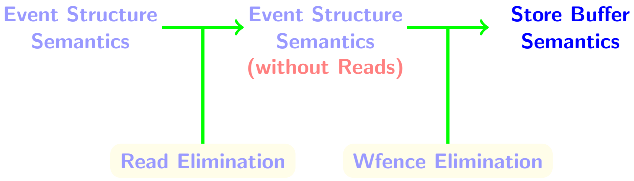


Memory History Buffer

From Event Structures to Buffers

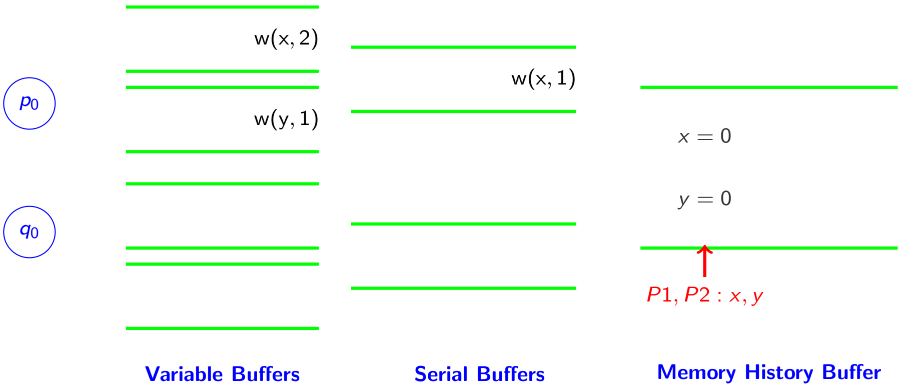

Event Structure Semantics
Event Structure Semantics (without Reads)
Store Buffer Semantics
Read Elimination
Wfence Elimination

From Event Structures to Buffers


Event Structure Semantics
Event Structure Semantics (without Reads)
Store Buffer Semantics
Read Elimination
Wfence Elimination

Elimination of Write Fences

Configurations= Control states + Variable/Serial Buffers + History Buffer



The State Reachability Problem for NSW

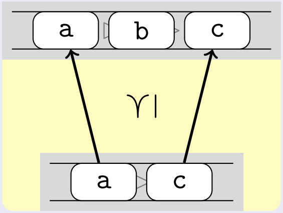
Decidability of State Reachability

Approach: Well Structured Systems [Abdulla et al., Finkel et al.]

- Well-Quasi Ordering $\leq$ on Configurations
  on every sequence $c_0, c_1, c_2, \ldots, \exists i < j.\ c_i \leq c_j$
- Monotonicity:
  $\leq$ is a simulation relation w.r.t. transition relation of the model
- $\Rightarrow$ Backward reachability analysis terminates

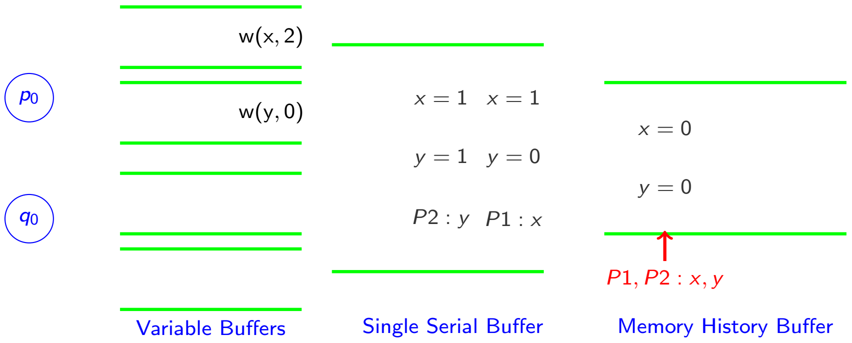
Problem: NSW ?

- Sub-word ordering on buffers?
  - NSW are Not Monotonic!
- Hard to apply WSS framework to NSW



NSW$^+$ systems

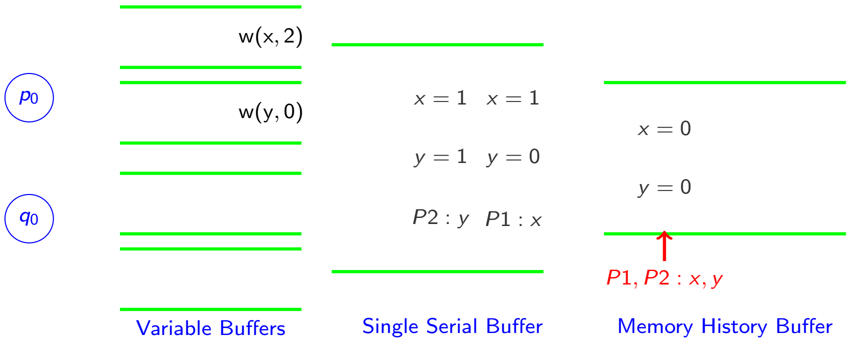
- NSW $\equiv$ NSW$^+$
- NSW$^+$: WSS wrt $\preceq$



NSW$^+$ systems

- NSW $\equiv$ NSW$^+$
- NSW$^+$: WSS wrt $\preceq$

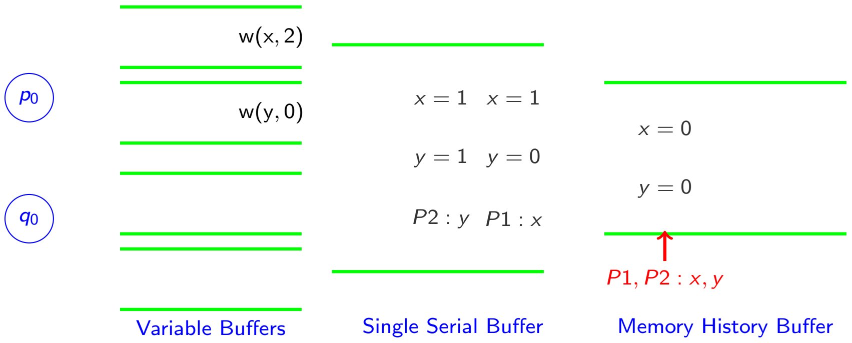
Single Serial Buffer



NSW$^+$ systems

- NSW $\equiv$ NSW$^+$
- NSW$^+$: WSS wrt $\preceq$

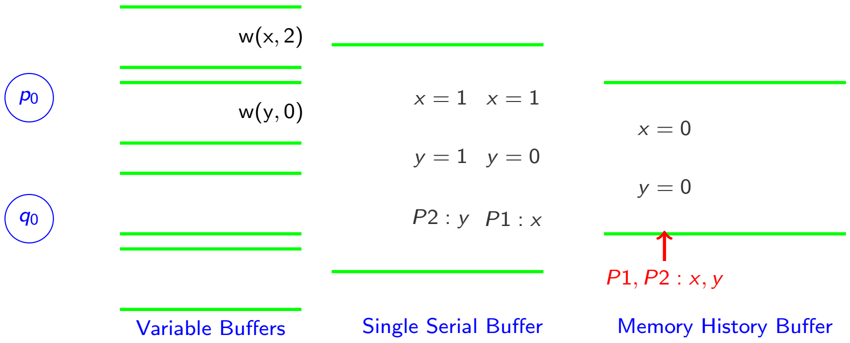
Each message in the serial buffer contains a snapshot of memory



NSW$^+$ systems

- NSW $\equiv$ NSW$^+$
- NSW$^+$: WSS wrt $\preceq$

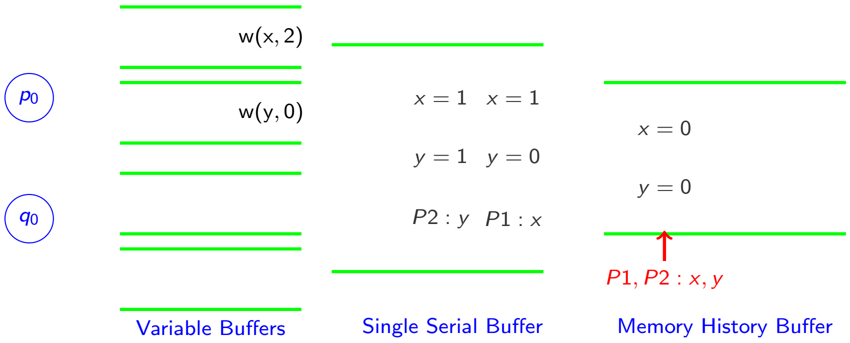
Unbounded buffers but lossy



NSW^+ systems

- $\text{NSW} \equiv \text{NSW}^+$
- NSW^+: WSS wrt $\preceq$

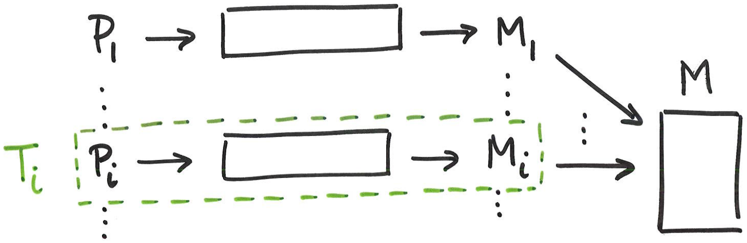
Processes have different views of memory (the use of pointers)



State Reachability: Under approximate analysis

- What is a suitable bounding notion ?
- Should allow a compositional reduction to SC
- Should avoid representing the contents of store buffers

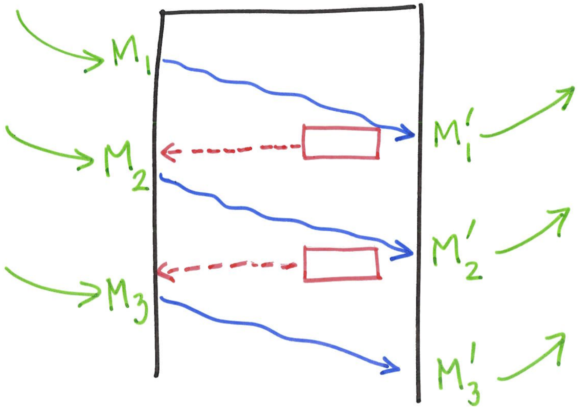
K-round Reachability



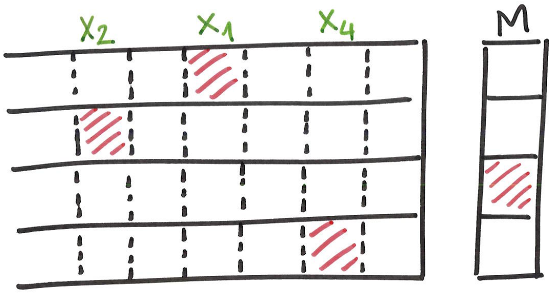
$$\text{Run} = \underbrace{P_{i_1} M_{i_1}}_{\text{round}} \underbrace{P_{i_2} M_{i_2}} \underbrace{P_{i_3} M_{i_3}} \cdots$$

K_round bounded : $\forall i.\ T_i$ has $\leq K$ rounds

Compositional Reasoning



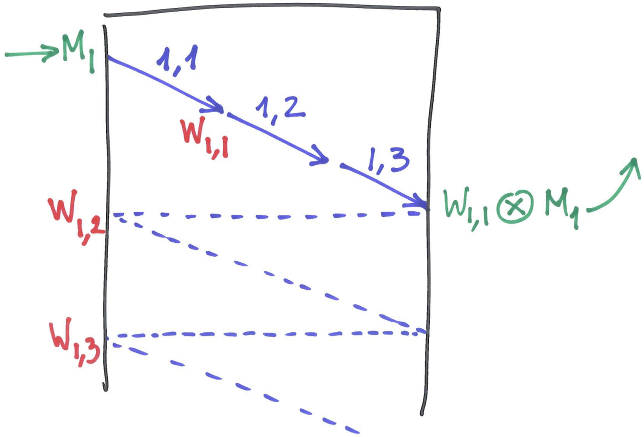
Encoding Store Buffers: The View of a Process



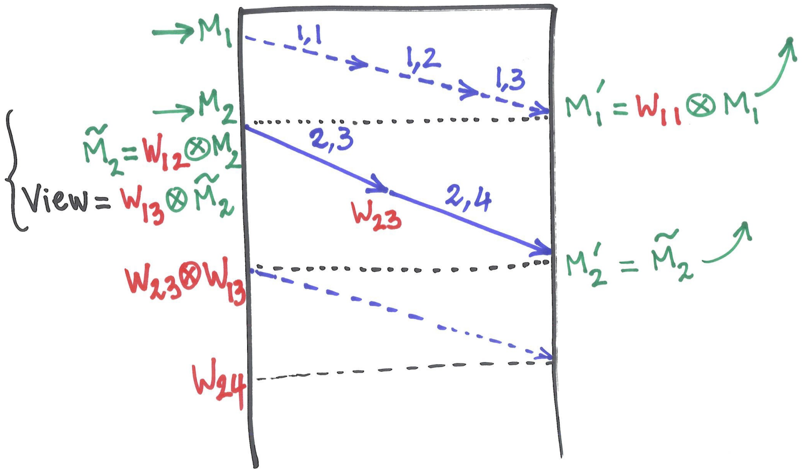
$\text{Mask} : \text{Var} \rightarrow \{0,1\}$

$\text{Queue} : \text{Var} \rightarrow \mathbb{D} \cup \{\perp\}$

Simulating Round 1

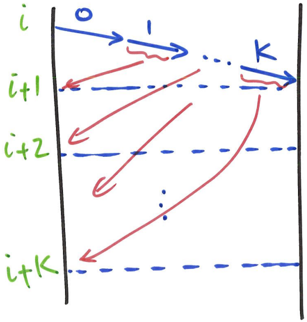


Simulating Round 2

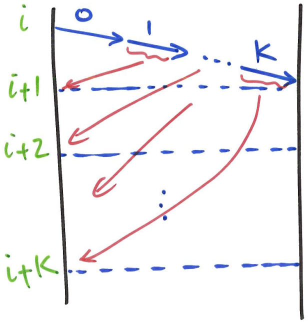


M_1
1,1
1,2
1,3
$M'_1 = W_{11} \otimes M_1$
M_2
$\tilde{M}_2 = W_{12} \otimes M_2$
View $= W_{13} \otimes \tilde{M}_2$
2,3
W_{23}
2,4
$M'_2 = \tilde{M}_2$
$W_{23} \otimes W_{13}$
W_{24}

Bounding Store Ages



Bounding Store Ages



Translation: $Mask_j$ and $Queue_j$ are used circularly (modulo $K + 1$).

Consequences

- K-round reachability is decidable for boolean concurrent programs with recursive procedure calls.
- K-store-age reachability is decidable for boolean concurrent programs with finite-state threads (without recursion).
- These results hold also for programs with parametric/dynamic number of threads. (Reduction to coverability in Petri nets, using [Atig, B., Qadeer, 2009] for programs with recursion)
- It is possible to use existing tools for the analysis/verification/testing of concurrent programs under SC.

State Reachability: Conclusion

- State Reachability: Decidable for TSO and beyond. Undecidability when speculative writes are allowed.
- But it is a hard problem (nonprimitive recursive when decidable) !
- However, it is possible to have efficient analysis techniques
- Reduction to SC is a promising idea, can be generalized beyond TSO
- Abstraction-based techniques:

  e.g., [Kuperstein, Vechev, Yahav, PLDI'11]

- Symbolic techniques:

  [Abdulla, Atig, Chen, Leonardson, Rezine, TACAS'12]
  [Linden, Wolper, SPIN'10-11]

- Other important models: PowerPC, ARM (hardware), C++