Maximal Causality Reduction for TSO and PSO

Shiyou Huang   Jeff Huang
huangsy@tamu.edu
Parasol Lab, Texas A&M University

A Real PSO Bug – $12 million loss of equipment

curPos = new Point(1,2);
class Point { int x, y; }

Thread 1:
newPos = new
Point(**curPos.x+1, curPos.y+1**);

- x=0
- y=0
- x=curPos.x+1
- y=curPos.y+1
- curPos = object

Thread 2:
while (newPos != null)
if (**newPos.x+1 != newPos.y**)
ERROR

http://stackoverflow.com/questions/16159203/

Memory Consistencies

http://preshing.com/20120930/weak-vs-strong-memory-models/

TSO and PSO

Total Store Ordering (TSO)

For a write ***w*** and a read ***r*** by the same thread, the **read r** can be reordered with the write **w** if the two operations access different locations.

Partial Store Ordering (PSO)

For a write ***w1*** and a write ***w2*** by the same thread, the write ***w2*** can be reordered with the write ***w1*** if the two operations access different locations.

New State Generated under TSO/PSO

```
Init: x=y=0

thread 1:            thread 2:
  x = 1  //a1          y = 1  //b1
  a = y  //a2          b = x  //b2

    Assert (a==1 || b==1)
```

b2 – a1 – a2 – b1 **(a=0, b=0)**

Huge Interleaving Space

#interleaving =

$$\prod_{i=1}^{M} \binom{\sum_{j=i}^{M} N_j}{N_i}$$

(Lu et al. FSE'07)

(M : #threads and N_i : #accesses by thread i)

M=4, N1=N2=N3=N4=4, #interleavings > 60 million

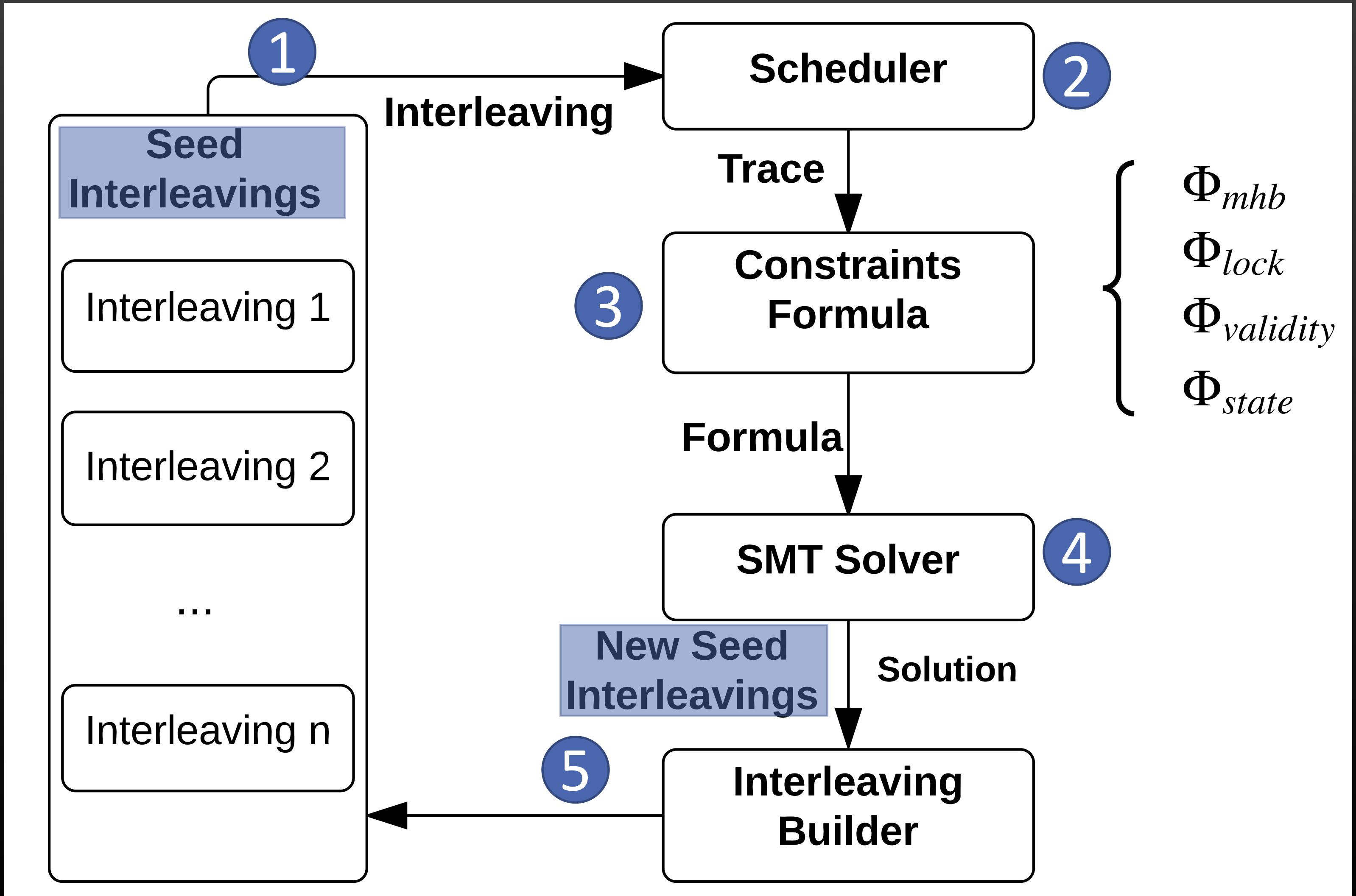
Related Work

- Dynamic Partial Order Reduction (DPOR) [Flanagan et al., POPL’05]
- Maximal Causality Reduction [Huang, PLDI’15]
- rInspect [Zhang et al., PLDI’15]
- SATCheck [Demsky and Lam, OOPSLA’15]

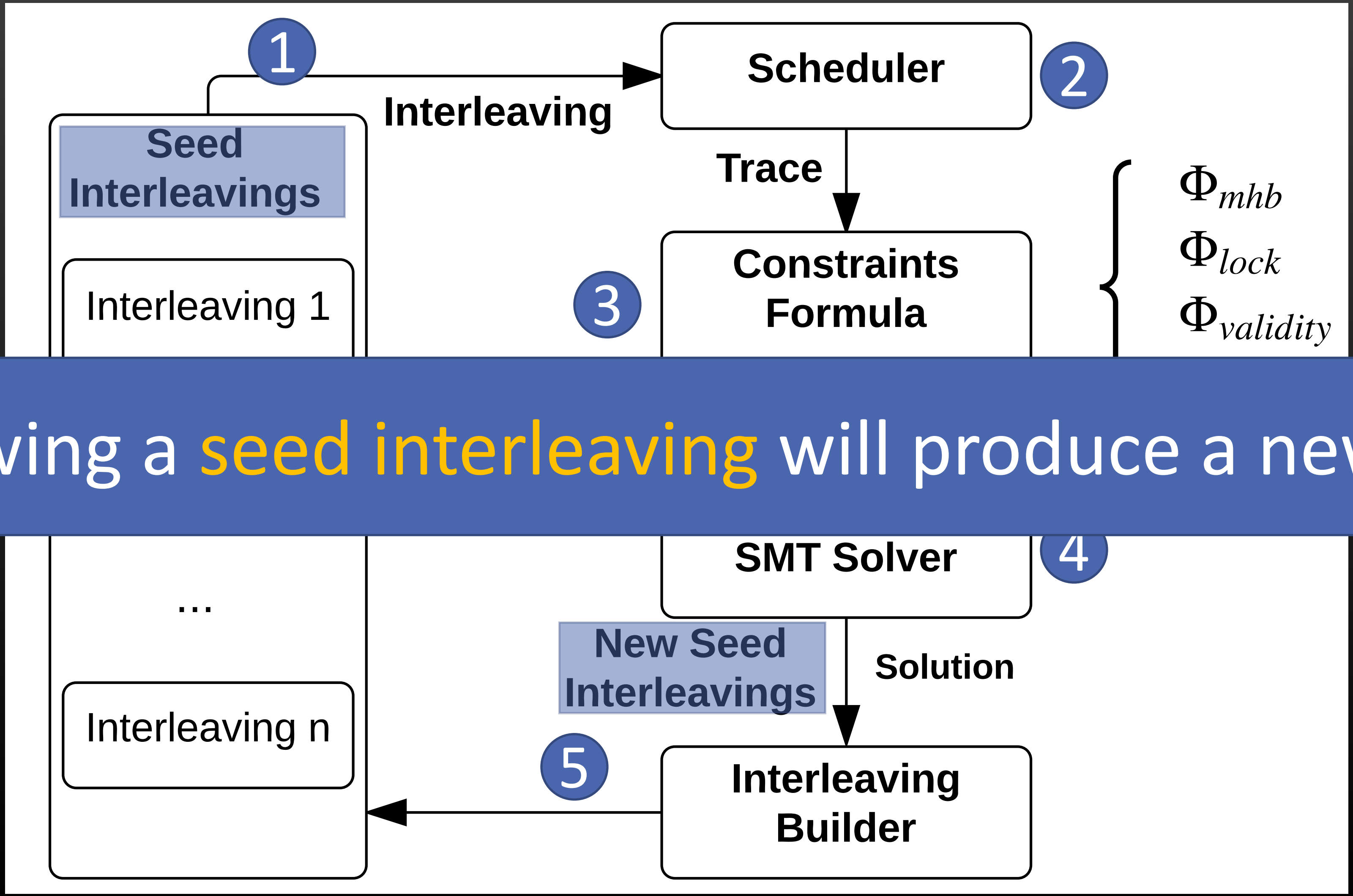
Maximal Causality Reduction (MCR)

Given an executed trace, MCR generates new interleavings to explore the program state space. Each new interleaving (called **seed interleaving**) enforces at least one read to read a new value.

Workflow of MCR



Workflow of MCR



Following a seed interleaving will produce a new state

Constraints (ϕ)

- happens-before
- lock-mutual-exclusion

  (l_1, u_1) and (l_2, u_2): $O_{u_1} < O_{l_2} \vee O_{u_2} < O_{l_1}$

- validity

$$\Phi_{value}(r, v) \equiv \bigvee_{w \in W_v^x} (\Phi_{validity}(w) \wedge O_w < O_r \bigwedge_{w \neq w' \in W^x} (O_{w'} < O_w \vee O_r < O_{w'}))$$

- new state

Constraints (ϕ)

- happens-before
- lock-mutual-exclusion

$$(l_1, u_1) \text{ and } (l_2, u_2)\text{: } O_{u_1} < O_{l_2} \vee O_{u_2} < O_{l_1}$$

- validity
-

An event is feasible if every read in the seed interleaving returns the same value as that in the previous trace.

Constraints (ϕ)

- happens-before
- lock-mutual-exclusion

  (l_1, u_1) and (l_2, u_2): $O_{u_1} < O_{l_2} \vee O_{u_2} < O_{l_1}$

- validity

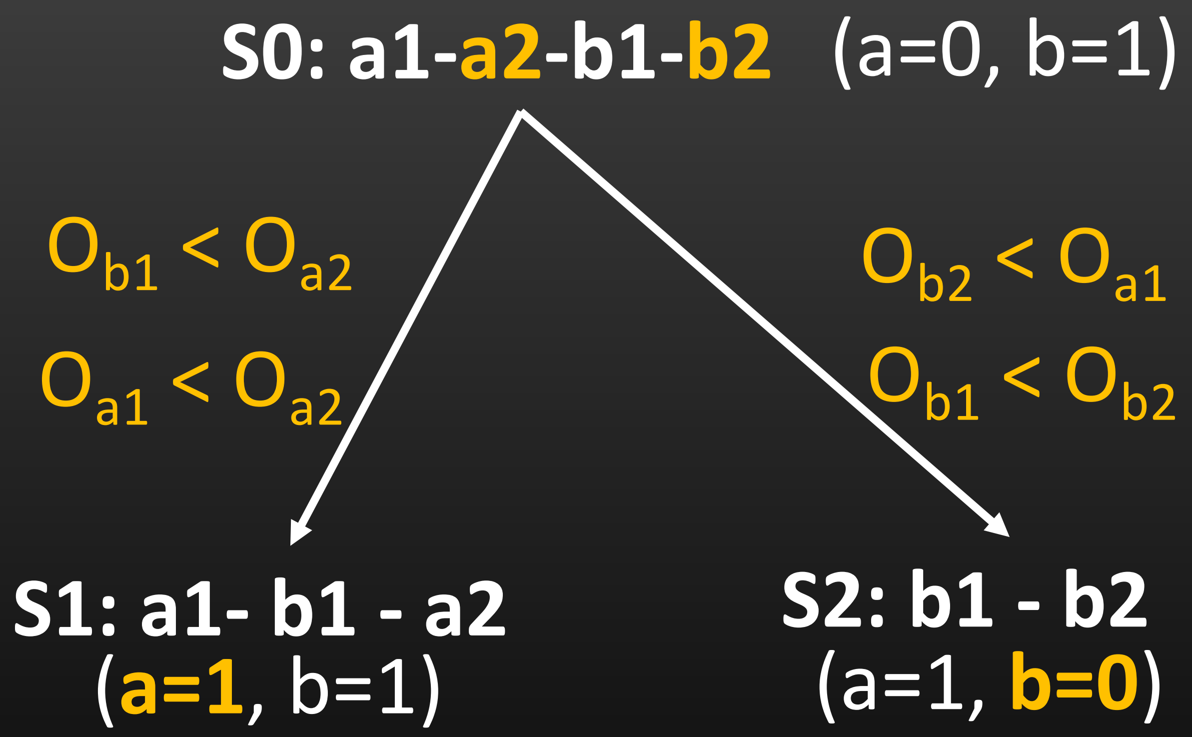
$$\Phi_{value}(r, v) \equiv \bigvee_{w \in W_v^x} (\Phi_{validity}(w) \wedge O_w < O_r \bigwedge_{w \neq w' \in W^x} (O_{w'} < O_w \vee O_r < O_{w'}))$$

- new state

An Example

```
Init: x=y=0
thread 1:
   x = 1   //a1
   a = y   //a2


thread 2:
   y = 1   //b1
   b = x   //b2
```



3 executions

Limitation of MCR

The original MCR only checks the program under sequential consistency.

Limitation of MCR

```
Init: x=y=0

thread 1:            thread 2:
   x = 1  //a1          y = 1  //b1
   a = y  //a2          b = x  //b2

    Assert (a==1 || b==1)
```

Contributions

- Extend MCR for TSO and PSO
- Present a new replay algorithm
- Evaluation on various applications
- Explore 5x – 10x fewer executions than DPOR

Two Challenges

1. Relax the happens-before constraints
2. Replay a schedule out of the program order

Happens-before Relaxation

Relax the happens-before relation of the write-read and write-write events by the same thread:

$$
\Phi_{hb} = \begin{cases} \phi_{rr} & r1 \prec r2, \quad \textit{iff } r1, r2 \in \textit{Reads} \\ \phi_{addr} & e1 \prec e2, \quad \textit{iff } addr(e1) = addr(e2) \\ \phi_{r\text{-}w} & r \prec w, \quad \textit{iff } r \in \textit{Reads} \ \&\& \ w \in \textit{Writes} \\ \phi_{w\text{-}w} & w1 \prec w2, \ \textit{iff } w1, w2 \in \textit{Writes} \end{cases}
$$

Example

```
Init: x=y=0
thread 1:
  x = 1  //a1
  a = y  //a2

thread 2:
  y = 1  //b1
  b = x  //b2
```

Under SC:

$O_{a1} < O_{a2}$

$O_{b1} < O_{b2}$

Under TSO/PSO

$O_{a1}, O_{a2}, O_{b1}, O_{b2}$

Replay

```
thread 1:             thread 2:
   x = 1    //a1         y = 1    //b1
   a = y    //a2         b = x    //b2
```

Expecting: b2 – a1 – a2 – b1

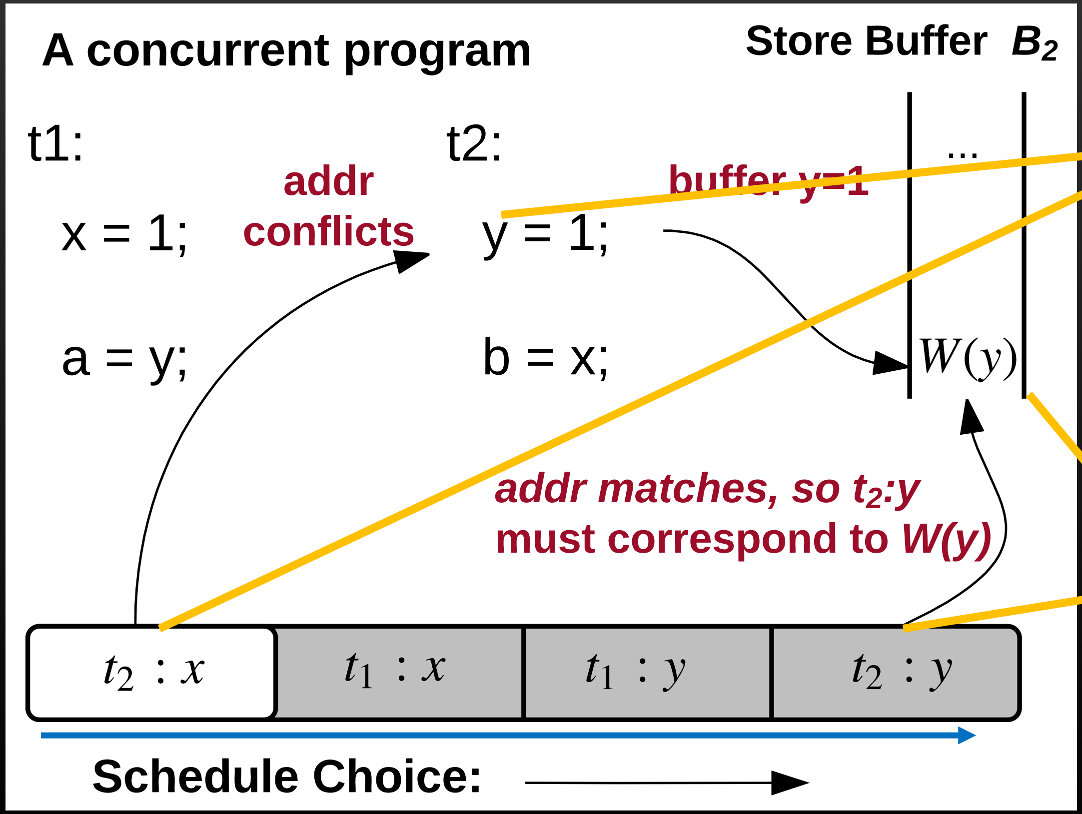
T2 – T1 – T1 – T2

Can't decide whether to buffer

Actual: b1 – a1 – a2 – b2

Replay

***Interleaving** : a sequence of schedule choices, with each schedule choice* **c(tid, addr)**.



Case 1: when addr(e) ≠ addr(c), buffer e

Case 2: when addr(c) = addr(w), w is buffered, update w

Constraints Construction

Initially x=1, y=2, z=0

Thread 1:

thread2.start()

```
1    z = 0
2    x = 0
3    y = 0
4    x = 2
5    y = 3
6    z = 1
```

thread2.join()

Thread 2:

```
7    if (z==1)
8       if (x+1 != y)
9          ERROR
```

Execution: 1-2-3-4-5-6-7-8-8-9

SC/TSO

$O_1 < O_2 < O_3 < O_4 < O_5 < O_6$

$O_7 < O_8^1 < O_8^2$

PSO

$O_1 < O_6$

$O_2 < O_4$

$O_3 < O_5$

$O_7 < O_8^1 < O_8^2$

PSO: O_1=1 , O_2=2, O_3=3, O_4=7, O_5=8, O_6=4, O_7=5, O_8^1=6

A feasible schedule: 1-2-3-6-7-8-4-5 that can trigger the error!

Replay

Replay: 1 - 2 - 3 - 6 - 7 - 8 - 4 - 5

$T_1^z - T_1^x - T_1^y - T_1^z - T_2^z - T_2^x - T_1^x - T_1^y$

thread 1:
1. z=0
2. x=0
3. y=0
4. x=1
5. y=2
6. z=1

thread 2:

7. if (z>0)
8. assert(x+1 == y)

Addr doesn't match

Addr doesn't match

Scheduler

x=1 **y=2**

1:z=0
2:x=0
3:y=0
6:z=1
7:z>0
8:x+1
4:x=1

Evaluation

- Java implementation using ASM and Z3
- Compared with rInspect [Zhang et al., PLDI'15] and SATCheck [Demsky and Lam, OOPSLA'15]
  - States pace exploration effectiveness
  - Efficiency of finding errors
- A collection of benchmarks with known errors

Benchmarks

- 7 popular small benchmarks
- 6 real Java applications including a large one weblech

Program	LoC	#Thrd	#Evt	Description
Dekker	119	3	56	Two critical sections with 3 shared variables.
Lamport	162	3	40	Two critical sections with 4 variables.
bakery	119	3	27	n critical sections using 2n shared variables. We take n=2.
Peterson	94	3	72	Two critical sections with 3 variables
StackUnsafe	135	3	34	Unsafe operations on a stack by two threads, which cause the stack underflow.
RVExample	79	3	32	An example from original MCR [21], which contains a very tricky error
Example	73	2	44	The example program from Figure 6 with loop number from 1 to 4.
Account	373	5	51	Concurrent account deposits and withdrawals suffering from atomicity violations.
Airline	136	6	67	A race condition causing the tickets oversold.
Allocation	348	3	125	An atomicity violation causing the same block allocated or freed twice.
PingPong	388	6	44	The player is set to null by one thread and dereferenced by another throwing NPE.
StringBuf	1339	3	70	An atomicity violation in Java StringBuffer causing StringIndexOutOfBoundsException.
Weblech	35K	3	2045	A tool for downloading websites and enumerating standard web-browser behavior.

State Space Exploration

Program	DPOR (rInspect)			MCR (our approach)			#Executions Reduction		
	SC	TSO	PSO	SC	TSO	PSO	SC	TSO	PSO
Dekker	248	252	508	62	98	155	4.0X	2.6X	3.3X
Lamport	128	208	2672	14	91	102	9.1X	2.3X	29.4X
Bakery	350	1164	2040	77	158	165	4.5X	7.1X	12.4X
Peterson	36	95	120	13	18	19	2.8X	5.3X	6.3X
StackUnsafe	252	252	252	29	46	108	8.7X	5.5X	2.3X
RVExample	1959	-	-	57	64	70	34.4X	-	-
Example (N=1 to 4)	4	4	-	2	2	10	2.0X	2.0X	-
	105	105	-	43	43	89	2.4X	2.4X	-
	4282	4282	-	296	296	819	14.5X	14.5X	-
	14840	14840	-	2767	2767	8420	5.4X	5.4X	-
Avg.	**435**	**394**	**1118**	**42**	**79**	**103**	**10.4X**	**5.0X**	**10.9X**

State Space Exploration

Program	DPOR (rInspect)			MCR (our approach)			#Executions Reduction		
	SC	TSO	PSO	SC	TSO	PSO	SC	TSO	PSO
Dekker	248	252	508	62	98	155	4.0X	2.6X	3.3X
Lamport	128	208	2672	14	91	102	9.1X	2.3X	29.4X
Bakery	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	12.4X
Peterson	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	6.3X
StackUnsafe	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	2.3X
RVExample	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	-
Example (N=1 to 4)	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	-
	105	105	-	43	43	89	2.4X	2.4X	-
	4282	4282	-	296	296	819	14.5X	14.5X	-
	14840	14840	-	2767	2767	8420	5.4X	5.4X	-
Avg.	**435**	**394**	**1118**	**42**	**79**	**103**	**10.4X**	**5.0X**	**10.9X**

Our approach explores 5x – 10x fewer executions than DPOR.

Finding Bugs

Program	DPOR			SATCheck		MCR (our approach)		
	SC	TSO	PSO	SC	TSO	SC	TSO	PSO
Dekker	22	28	29	32!	68735!	10	4	5
Lamport	6	8	24	-	-	2	2	3
Bakery	12	15	15	-	-	8	8	15
Peterson	4	5	6	19*	34282!	7	2	3
StackUnsafe	6	6	6	-	-	2	2	2
RVExample	301	-	-	60564!	70365!	53	54	39
Example	14840*	14840*	-	1*	1*	2767*	2767*	3
Avg.	**10**	**12**	**16**	**-**	**-**	**6**	**4**	**6**

!: repeat the same execution

*: finish without finding the bug

Finding Bugs

Program	DPOR			SATCheck		MCR (our approach)		
	SC	TSO	PSO	SC	TSO	SC	TSO	PSO
Dekker	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	5
Lamp	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	3
Bake	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	15
Peters	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	3
StackU	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	2
RVExample	301	-	-	60564!	70365!	53	54	39
Example	14840*	14840*	-	1*	1*	2767*	2767*	3
Avg.	**10**	**12**	**16**	**-**	**-**	**6**	**4**	**6**

Our approach needs 2X- 3X fewer executions than DPOR and SATCheck to find the bugs

!: repeat the same execution

*: finish without finding the bug

Conclusion

1. MCR for TSO and PSO
    - Relax the happens-before constraints
    - Faithfully replay the TSO/PSO interleavings
2. Explore 5X – 10X fewer executions than DPOR
3. Take fewer executions to find the bugs

Acknowledgement

Brian Demsky

University of California, Irvine

Patrick Lam

University of Waterloo

Thank you

&

Questions?