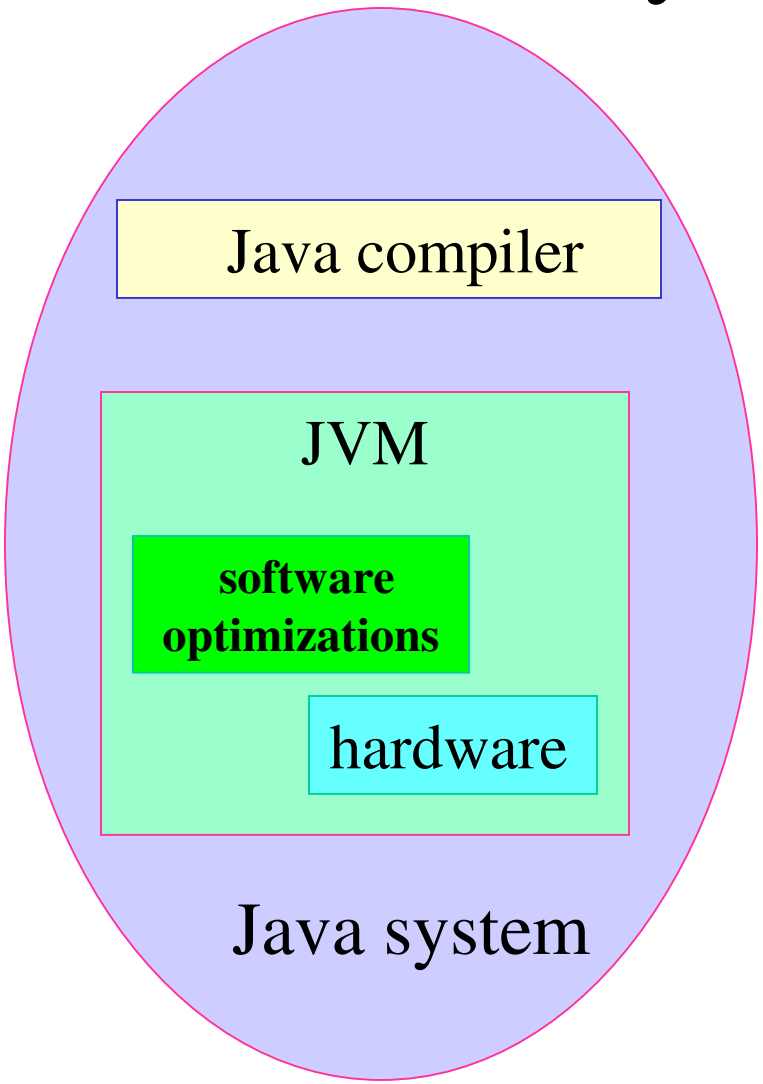
Verification of the Java Causality Requirements

Sergey Polyakov and Assaf Schuster

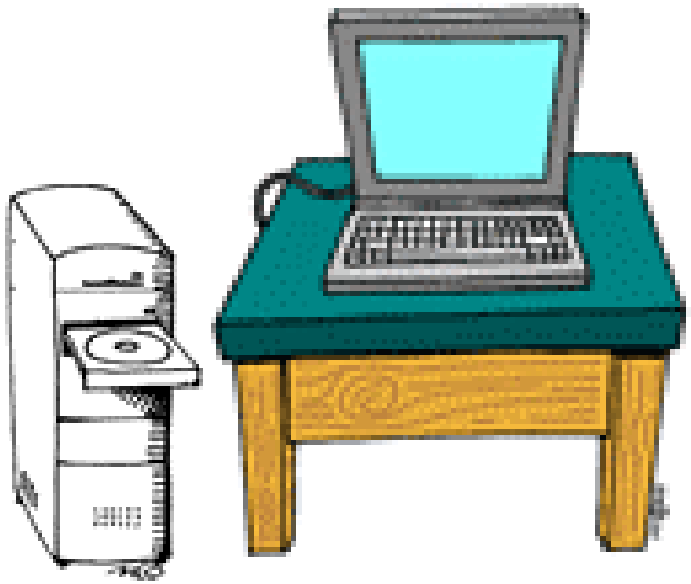
Computer Science Department, Technion

Why verify JMM



- The complexity of the Java system necessitates testing of its implementations
- JMM is a recent development and verification is required to see whether it conforms to its goals:
  - allow all reasonable optimizations
  - safety
  - understandability and consistency

Why post-mortem verification



- Java system is difficult for formal verification
  - Java system implementation is complex
  - JMM is formalized in the non-operational form
- Testing complements formal verification

Presentation of the examples

```
class MyThread extends Thread{
    int x;
    int y;
    MyThread(){}
    public void run() {
            x = 2;
            y = 3;
            . . .
    }
    public static void main(String[] args){
            MyThread mt = new MyThread();
            mt.start();
            int r1 = mt.x;
            if (r1 > 0)
               int r2 = mt.y;
             . . .
}
```

Initially, x = y = 0

Thread 1	Thread 2
r1 = x;	x = 2;
if (r1 >0)	y = 3;
r2 = y;	
. . .	. . .

Causality requirements: guarantees for correctly synchronized programs

Initially, x = y = 0

Thread 1	Thread 2
r1 = x;	r2 = y;
if (r1 != 0)	if (r2 != 0)
y = 1;	x = 1;

r1 == r2 == 0 is the only legal behavior

- The program is ***correctly synchronized*** if it is data-race free on a sequentially consistent platform
- Java must guarantee sequentially consistent behavior for correctly synchronized program

Causality requirements : safety guarantees

Initially, x = y = 0

Thread 1	Thread 2
r1 = x; **42**	r2 = y; **42**
y = r1; **42**	x = r2; **42** **justified**

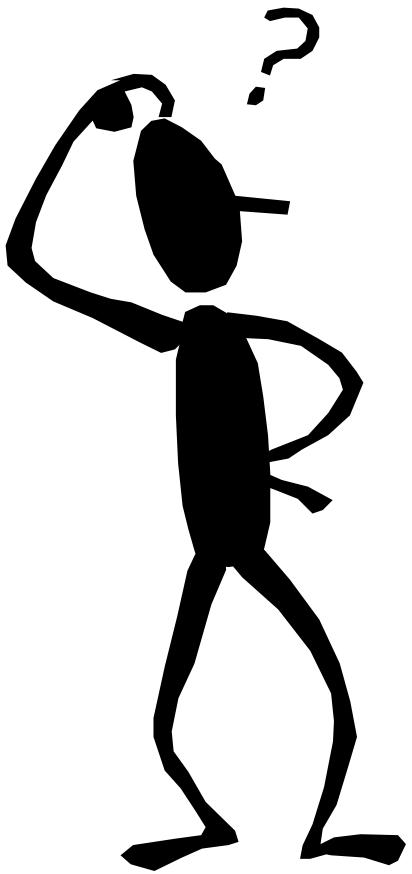
Incorrectly synchronized, but

r1 == r2 == 42

must not be allowed

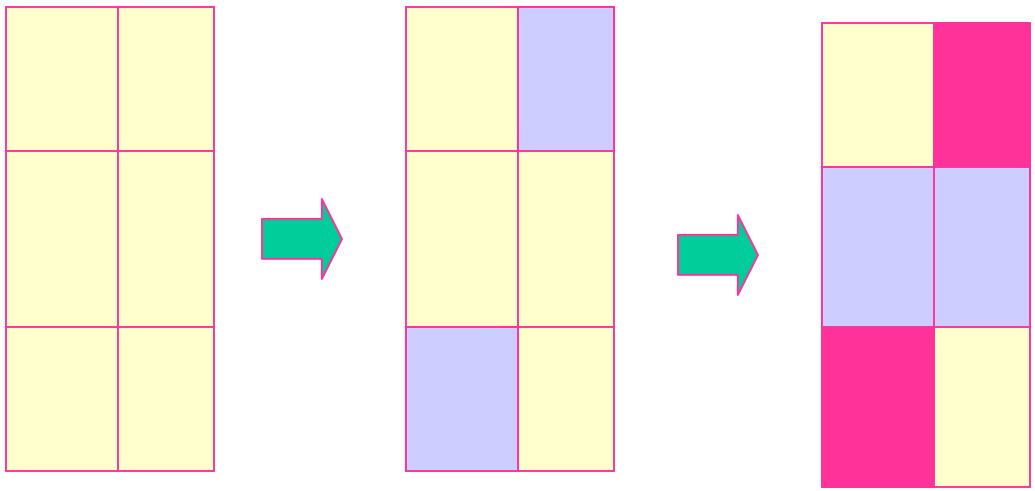
- Java must provide safety guarantees for incorrectly synchronized programs: values must not appear "out of thin air"

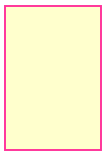
Causality requirements: intuition



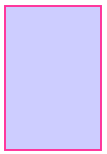
- Gradually building a set of “committed” actions
- Usually,the next action to be committed will reflect the next action that can be performed by a sequentially consistent execution
- Informally, an action is allowed to be committed early only if it can occur without data race

Causality requirements: definition





Not committed

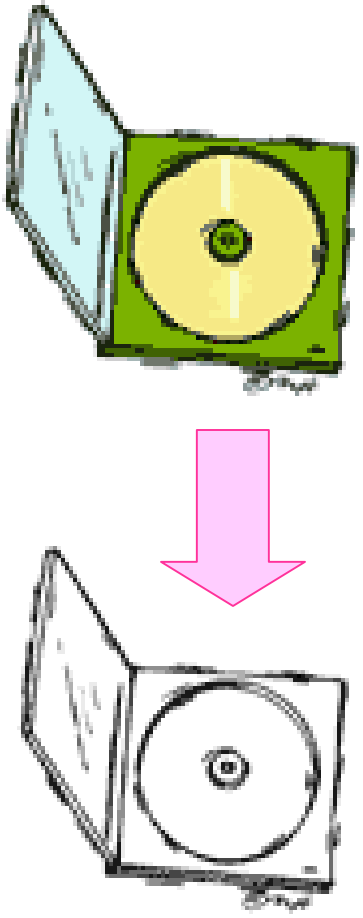


In the first commitment step

In the second commitment step

- **Actions must be committed in the series of executions**
- **Each thread is executed as if it is isolated**
- **Read actions in the second step of commitment may return values written by other threads**

Simplification of test programs



- Final fields are not used
- Synchronization is not used
- As result, simplified memory model

Example

Initially, x = y = 0

Thread 1	Thread 2
r1 = x;	r2 = y;
if (r1 >= 0)	x = r2
y = 1;	

r1 == r2 == 1 is allowed

Step 4

Thread 1	Thread 2
r1 = 1;	r2 = 1;
y = 1 ;	x = 1;

All reads may see values written by other threads

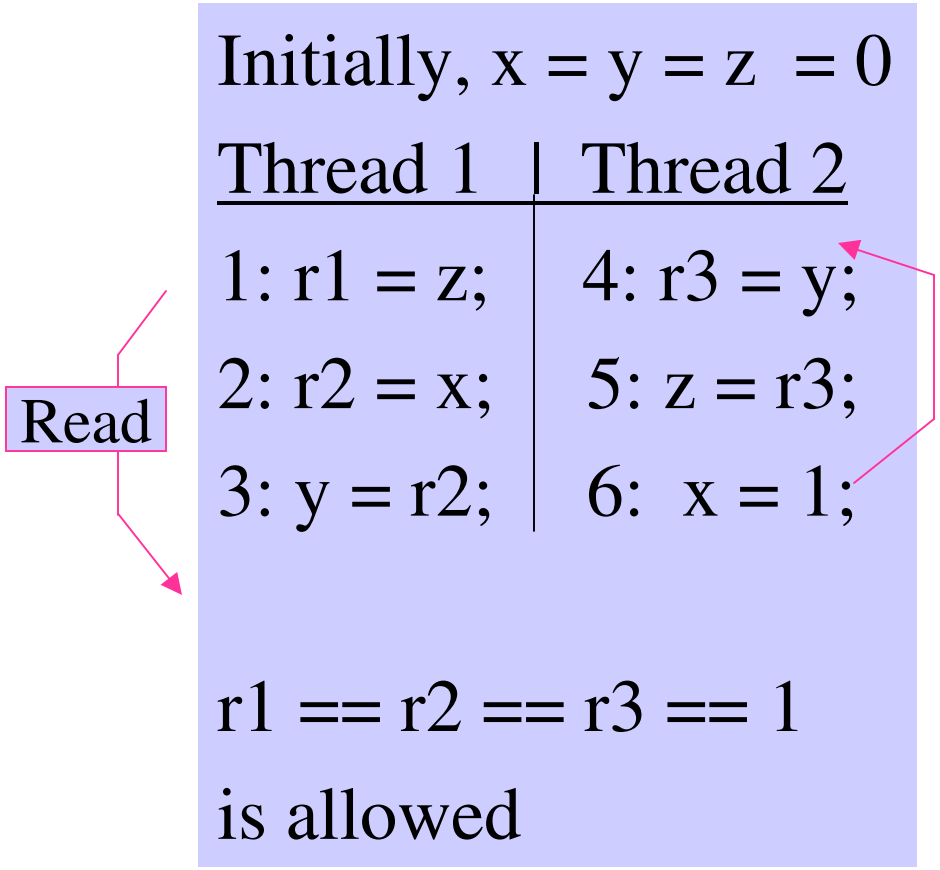
Verification task is non-polynomial

Default: a = 0

Thread 1	Thread 2	C_5	E_5
a = 1;	...		
	$r_{ai} = a;$	*	1
	$r_{bi} = a;$	*	1
	$r_i = r_{ai} - r_{bi};$		0
	...		
	$r_{ai} = a;$	*	1
	$r_{bi} = a;$	*	1
	$r_i = r_{ai} - r_{bi};$		0

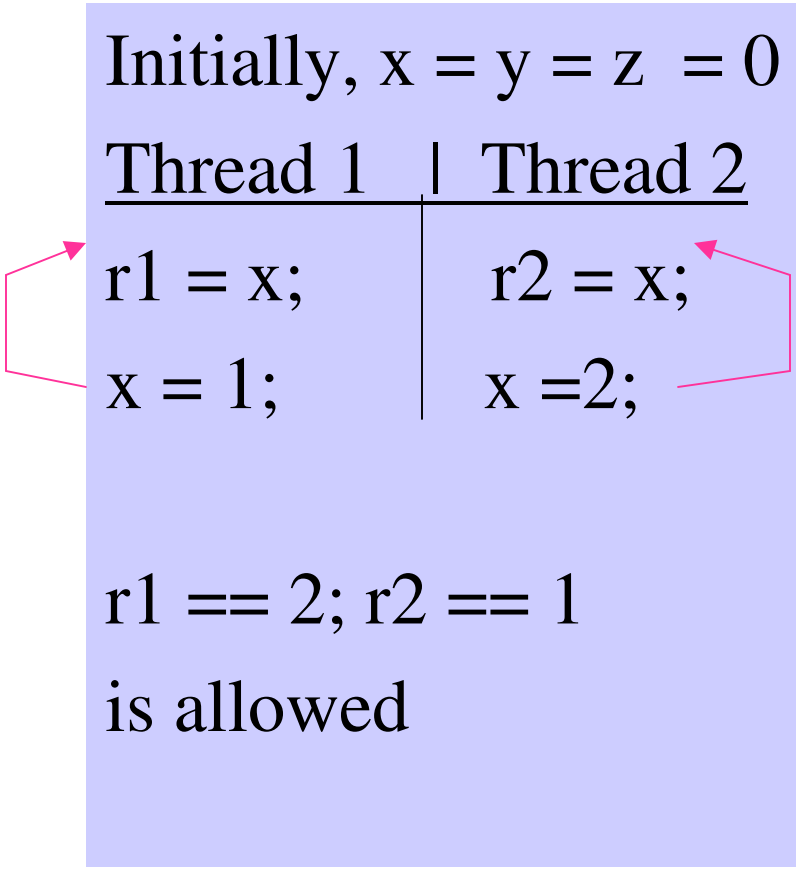
- Reads are independent of one another
- Non-polynomial number of possible combinations of values
- E_3 and E_4 are used for latching the simulation "trigger"

Commitment of Reads in program order



- Objectives: early execution of a Read does not result in a new behavior
- Why rejected: prohibits reasonable optimizations (causality test case 7)

Commitment of actions on the same variable in program order

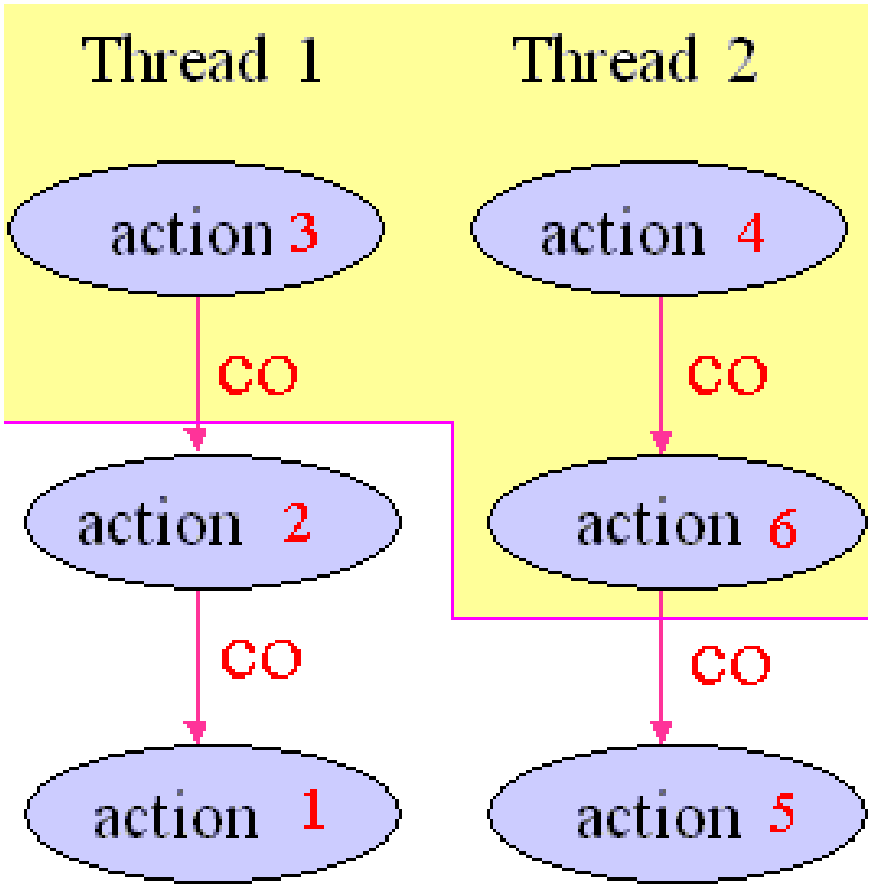


- Objectives: natural for coherent systems
- Why rejected: prohibits reasonable optimizations (causality test case 16)

Commitment of Read actions on the same variable in program order

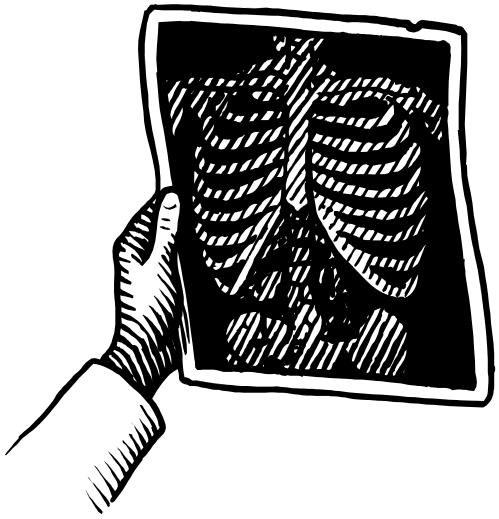
- Objectives: natural if the implementation doesn’t reorder Reads of the same variable
- It is unclear if this restriction is acceptable
- The complexity must be further researched

Tracing and frontier graphs



- In the general case: a non-polynomial number of possible commitment orders for each thread
- If commitment order is fixed by the trace: a polynomial number of sets of committed actions

Tracing Read actions



- If it is known what Reads are committed in each step, then values returned by Reads in each step are also known
- If values returned by Reads in each step are known, then values written by Writes in each step are known
- Writes that provide values for Read actions in other threads must be committed
- There is no need to commit other Writes

Verification algorithm

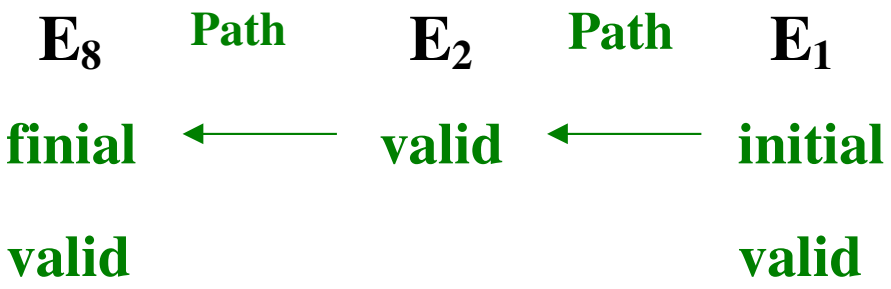
- Consider all possible intermediate executions
  - each one is defined by a set of committed Reads
- Find a path from an initial execution to the final one

Example 1

Initially, x = y = 0

Thread 1	Thread 2
1: r3 = x;	5: r2 = y;
if (r3 == 0)	6: x = r2;
2: x = 42;	
3: r1 = x;	
4: y = r1;	

r1 == r2 == r3 == 42 is a legal behavior



ALLOWED

Example 2

Initially, x = y = 0

Thread 1	Thread 2
1: r3 = x;	3: r2 = y;
2: y = r1;	4: x = r2;

r1 == r2 == 1 is not a legal behavior

PROHIBITED

Example 3

Initially, x = y = z 0

Thread 1	Thread 2	Thread 3	Thread 4
1: r3 = x;	3: r2 = y;	5: z = 42;	6: r0 = z;
2: y = r1;	4: x = r2;		7: x = r0;

r0 == 0, r1 == r2 == 42 is not a legal behavior

PROHIBITED

A problem: causality test case 6

Initially, A = B = 0

Thread 1	Thread 2
1: r1 = A;	3: r2 = B;
if (r1 == 1)	if (r2 == 1)
2: B = 1;	4: A = 1;
	if (r2 == 0)
	5: A = 1;

r1 == r2 == 1

is a legal behavior

PROHIBITED

Causality test case 6: proposition

Initially, A = B = 0

Thread 1	Thread 2
1: r1 = A;	3: r2 = B;
if (r1 == 1)	if (r2 == 1)
2: B = 1;	4: A = 1;
	if (r2 == 0)
	5: A = 1;

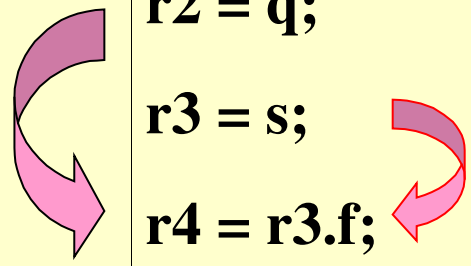
r1 == r2 == 1

is a legal behavior

ALLOWED

Open problems: final fields

Thread 1	Thread 2	Thread 3
r1.f = 1;		**r6 = p;**
p = r1;		**r6.g =1;**
freeze r1.f;		**freeze r5.g ;**
q = r1;	**r2 = q;**	**s = r6;**
dc1	**r3 = s;**	
	r4 = r3.f;	**dc2**
	r5 = r4.g;	



Either r4 or r5 cannot contain the default value

- ***dc*** may be choused in different ways

- there may be a non-polynomial number of possible results

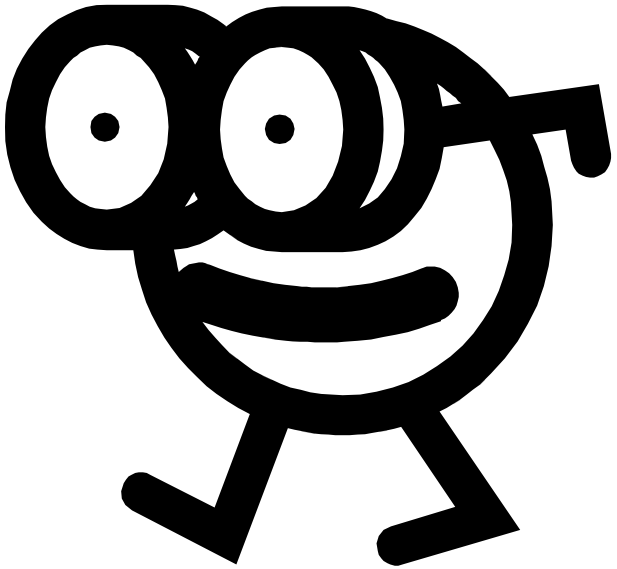
Open problems: synchronization

Thread 1	Thread 2	Thread 3
1: lock l;	4: lock m;	
2: x = 1;	5: x = 2;	
3: unlock l;	6: unlock m;	7: lock l;
		8: lock m;
		9: r1 = x;
		…

r1 may be either 1 or 2

- Assume **(3:)** $\xrightarrow{sq}$ **(7:), (6:)** $\xrightarrow{sq}$ **(8:).** Must be **(2:)** $\xrightarrow{hb}$ **(9:), (5:)** $\xrightarrow{hb}$ **(9:).**
- r1 has a **set** of hb-reads that it is allowed to see
- there may be a non-polynomial number of possible executions on each commitment step

Conclusions



- Verification of the Java causality requirements is NP-complete in the general case; in certain private case it is polynomial
- JMM needs clarifications
- Verification of the full-scale JMM (with final fields and synchronization) is an open problem