INFORMATIK

TECHNISCHE UNIVERSITÄT KAISERSLAUTERN

Modelling Memory Consistency Models for Formal Verification

Maximilian Senftleben

Embedded Systems Group
Department of Computer Science
TU Kaiserslautern

07.06.2019

INFORMATIK

TECHNISCHE UNIVERSITÄT KAISERSLAUTERN

Overview

Introduction

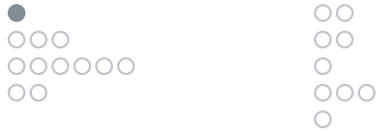
Weak Memory Model

- Sequential memory
  - Intuitive
  - Total order

Introduction

Weak Memory Model

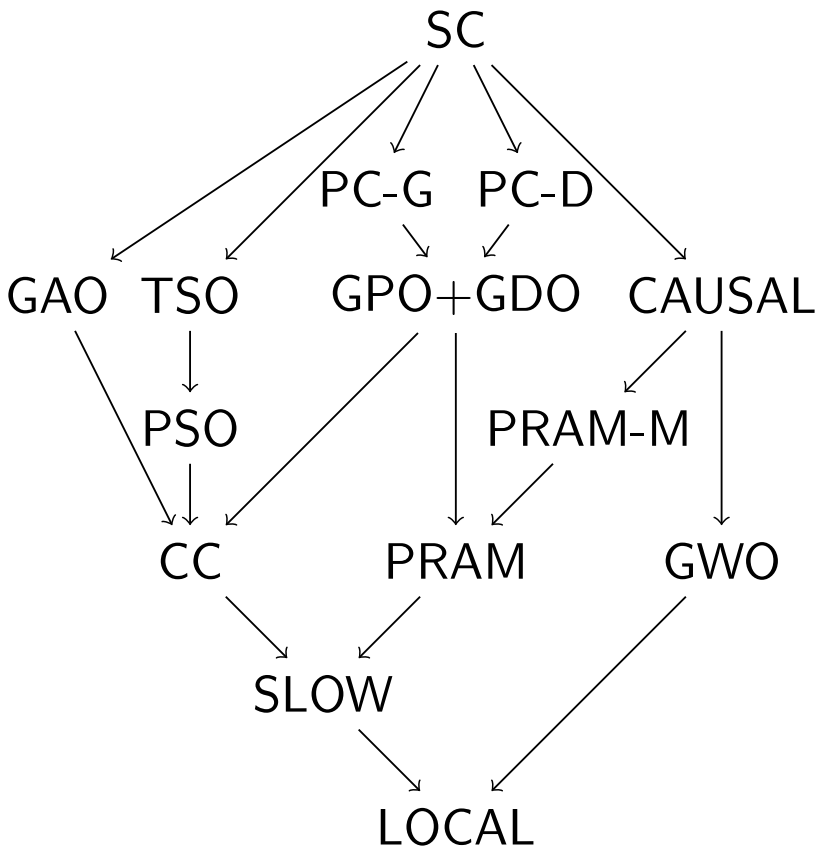
- Sequential memory
  - Intuitive
  - Total order
- Weak memory
  - Optimizations
  - More behavior
  - Potentially unexpected
  - Multitude of models



Introduction

Weak Memory Model

- Sequential memory
  - Intuitive
  - Total order
- Weak memory
  - Optimizations
  - More behavior
  - Potentially unexpected
  - Multitude of models



Example

Dekker's mutex

Procedure p

```
1 x = 1;
2 while y=1 do
3 |   x = 0;
4 |   Sleep(time);
5 |   x = 1;
6 end
7 Critical Section;
8 x = 0;
```

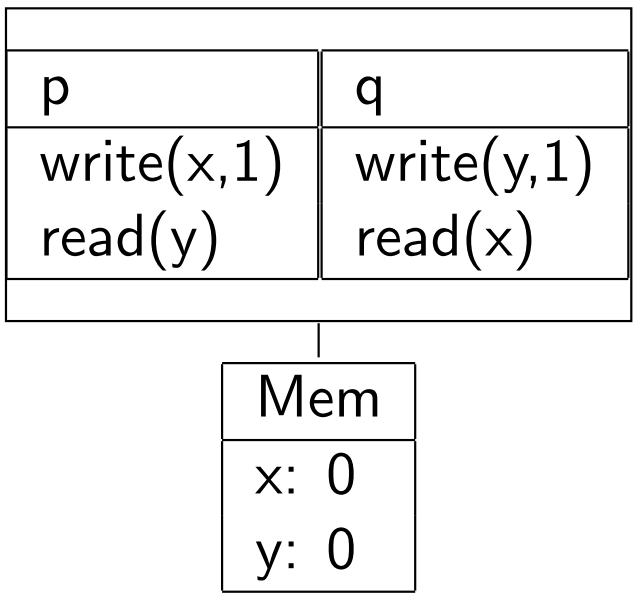
Procedure q

```
1 y = 1;
2 while x=1 do
3 |   y = 0;
4 |   Sleep(time);
5 |   y = 1;
6 end
7 Critical Section;
8 y = 0;
```

- goal: Only one process in critical section at a time.
- violation if both processes read 0.

Example

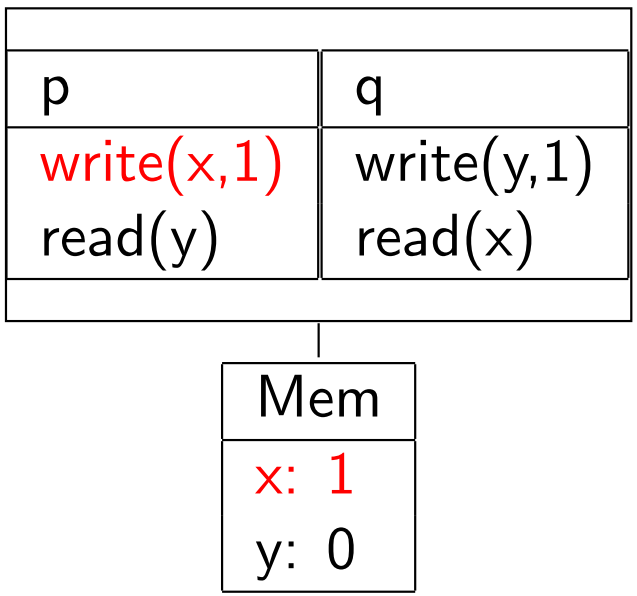
sequential consistency



- sequence:
- result (x,y) = (?,?)

Example

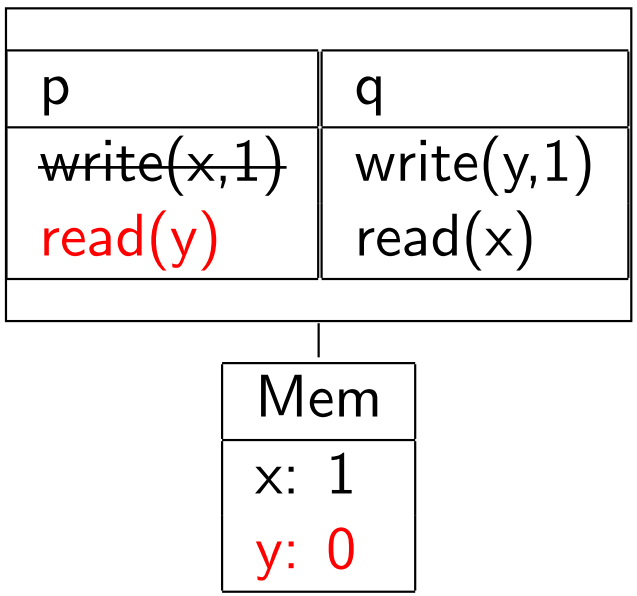
sequential consistency



- sequence: p0
- result (x,y) = (?,?)

Example

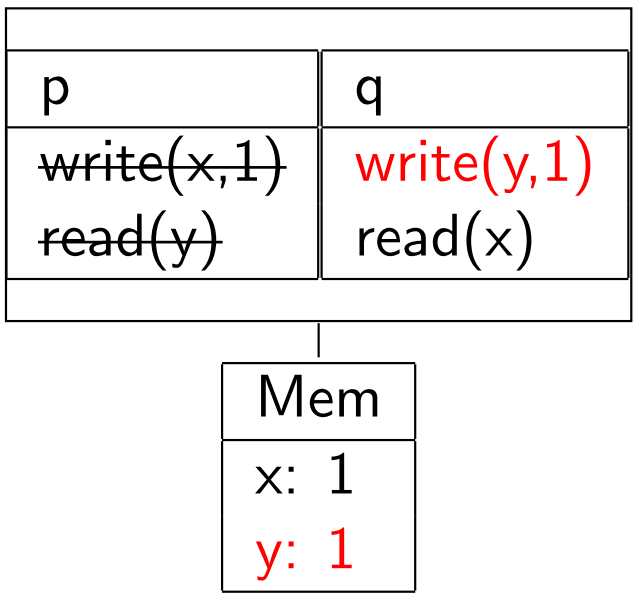
sequential consistency



- sequence: p0 p1
- result (x,y) = (?,0)

Example

sequential consistency



- sequence: p0 p1 q0
- result (x,y) = (?,0)

Example

sequential consistency

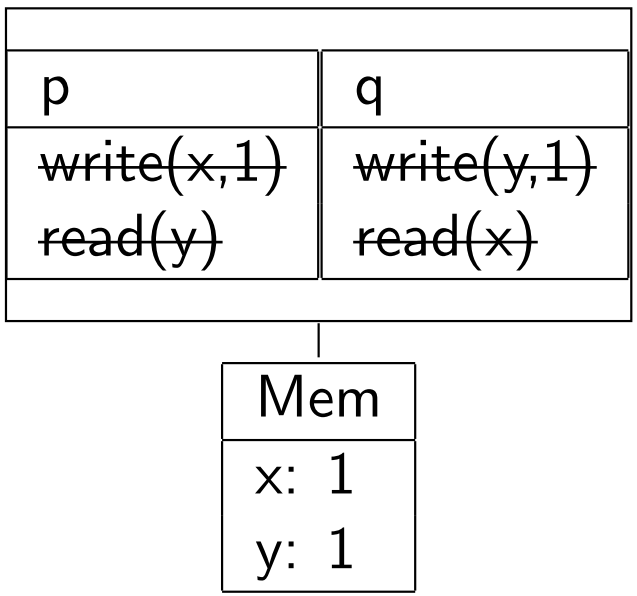
p	q
~~write(x,1)~~	~~write(y,1)~~
~~read(y)~~	read(x)

Mem
x: 1
y: 1

- sequence: p0 p1 q0 q1
- result (x,y) = (1,0)

Example

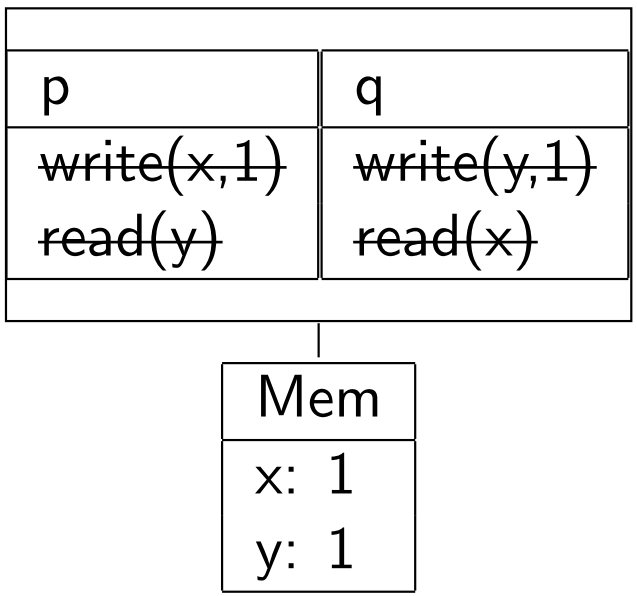
sequential consistency



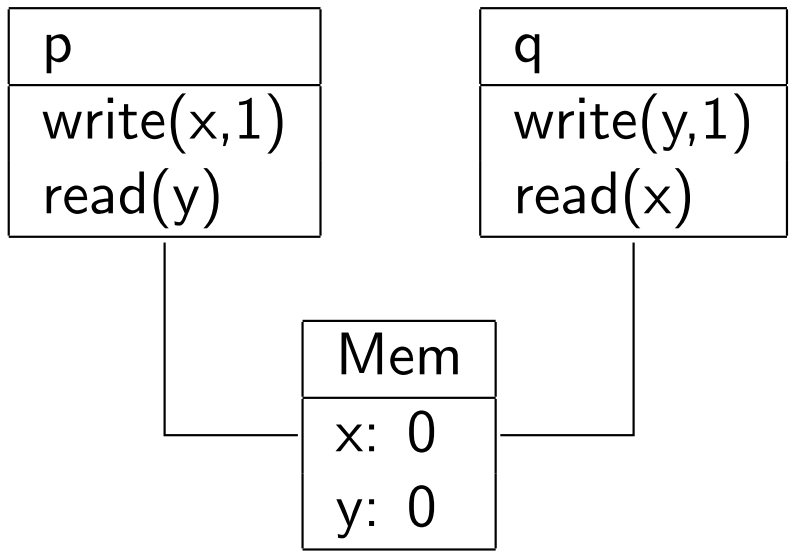
- sequence: p0 p1 q0 q1
- result (x,y) = (1,0)
- (0,1), (1,0) and (1,1) possible.
  (0,0) not possible

Example

sequential consistency



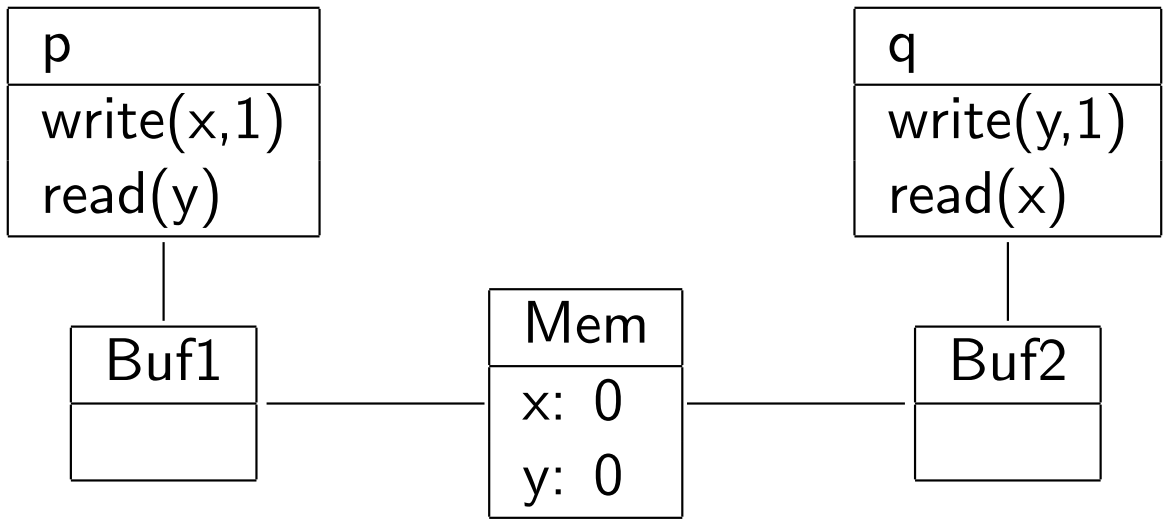
- sequence: p0 p1 q0 q1
- result (x,y) = (1,0)
- (0,1), (1,0) and (1,1) possible.
  (0,0) not possible



- multiprocessor
- same behavior as unicore:
  sequential interleaving

Example

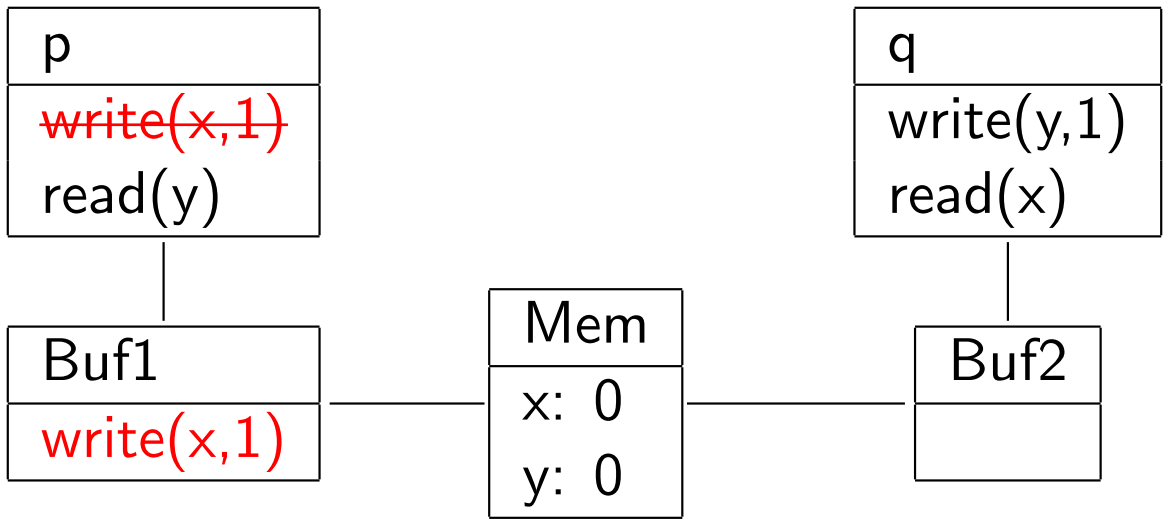
weak consistency



- result (x,y) = (?,?)

Example

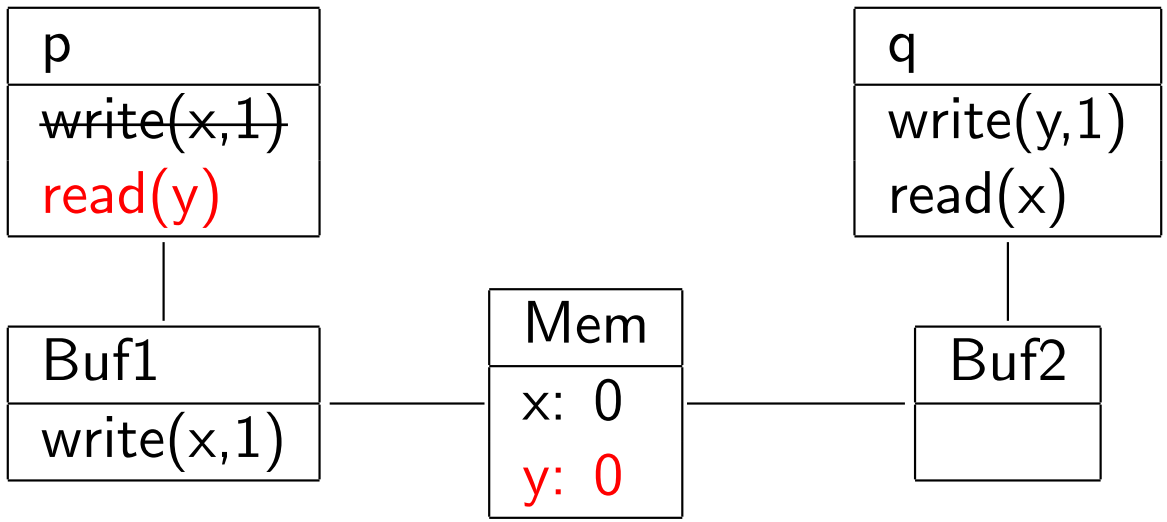
weak consistency



- result (x,y) = (?,?)

Example

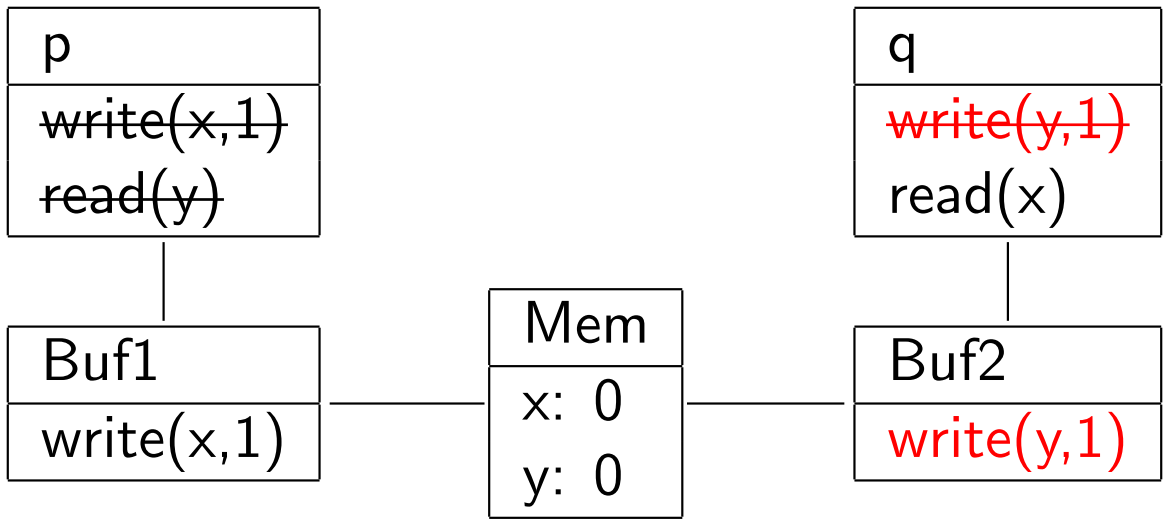
weak consistency



- result (x,y) = (?,0)

Example

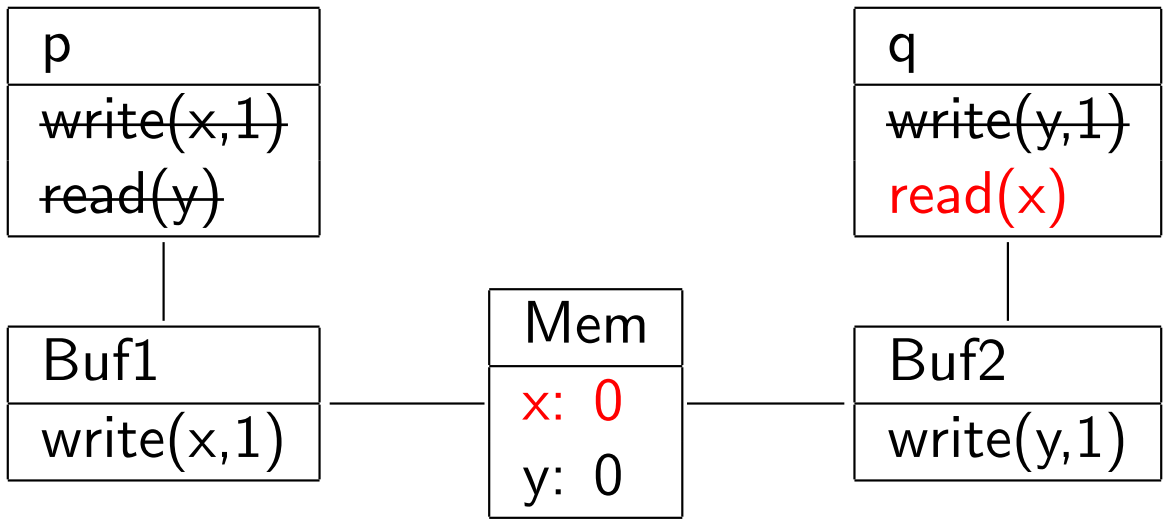
weak consistency



- result (x,y) = (?,0)

Example

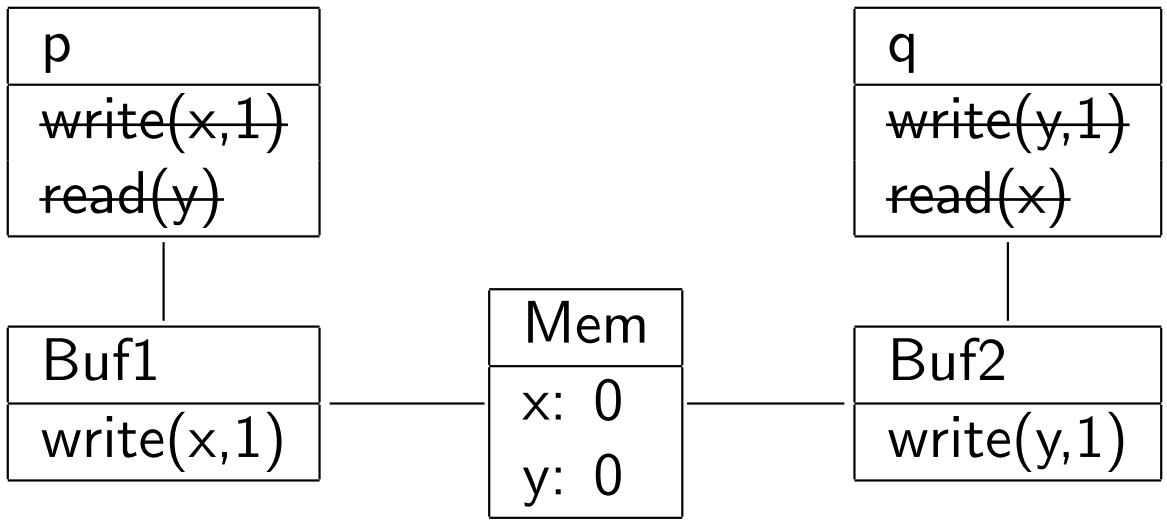
weak consistency



- result (x,y) = (0,0)

Example

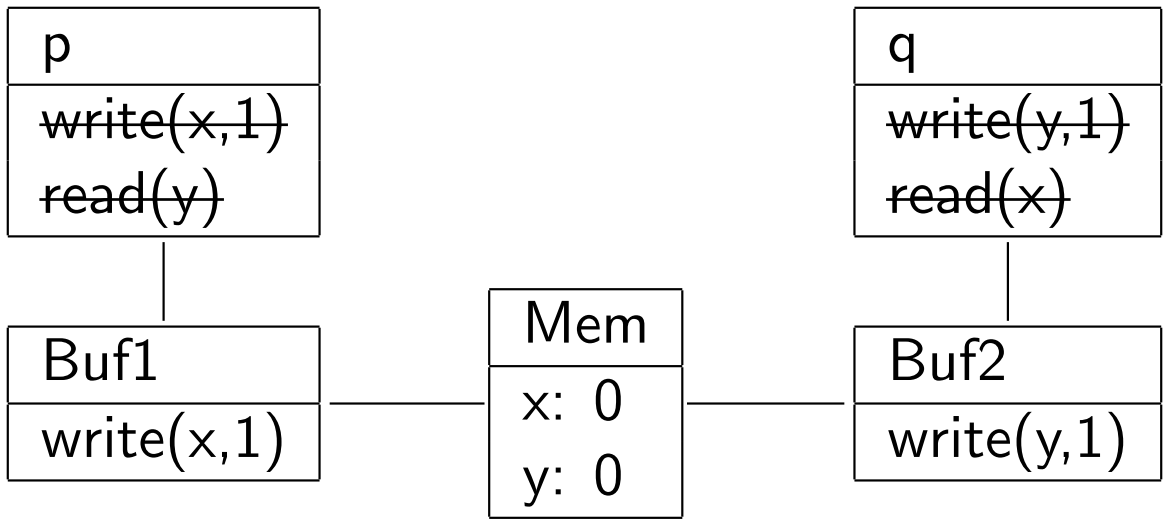
weak consistency



- result (x,y) = (0,0)
- violates mutual exclusion

Example

weak consistency



- result (x,y) = (0,0)
- violates mutual exclusion

- E.g. TSO consistency in SPARC, x86 processors

State of the Art

Modelling approaches

- Textual
- Litmus tests
- Axiomatic
- Operational
- View-based
- ...

State of the Art: Textual

Definition (PRAM)

Consider n processes $P_1, \ldots, P_n$ with a local memory $M_1, \ldots, M_n$ each. Process k executes a read from location i "by **performing a normal read** from location i" of its own memory M_k. Process k executes a write to location i with value v "by **performing a local action** and initializing a global action. Locally, it does a **normal write** to $[M_k]$ at location i with value v. Globally, it sends a message $< i, v >$ to all the other processors."

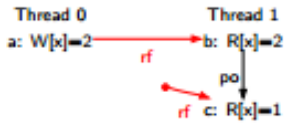
State of the Art: Litmus Tests

POWER and ARM Litmus Tests

http://www.cl.cam.ac.uk/~pes20/ppc-supplemental

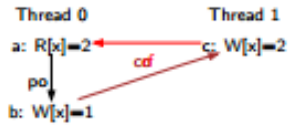
Coherence tests

CoRR1: rf,po,fr forbidden



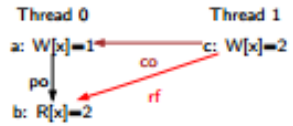
Test CoRR1

CoRW: rf,po,co forbidden



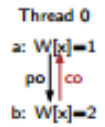
Test CoRW

CoWR: co,po,rf^{-1} forbidden



Test CoWR

CoWW: po,co forbidden

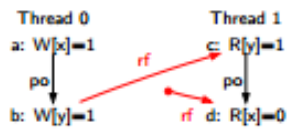


Test CoWW

4-edge 2-thread tests

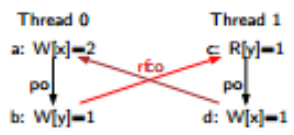
One rf

MP: rf,fr needs lwsync+RRdep



Test MP

S: rf,co needs lwsync+RWdep

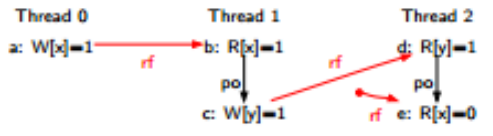


Test S

5-edge extensions along one rf edge

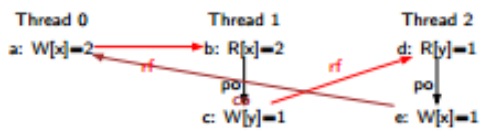
Two rf

WRC: rf,rf,fr needs lwsync+RRdep



Test WRC

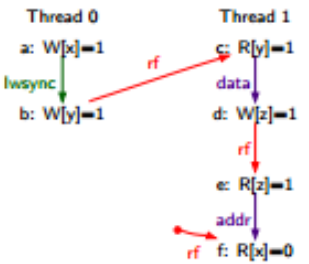
WWC: rf,rf,co needs lwsync+RWdep



Test WWC

Preserved read-read program order

PPO000–019: barrier,rf,intra-thread*,fr



Test PPO000

State of the Art: Axiomatic

Definition

An execution is TSO if there exists a memory order $\leq$ which respects:

- Order: $(w,i,x,v_1) \leq (w,j,y,v_2) \vee (w,j,y,v_2) \leq (w,i,x,v_1)$
- Value:

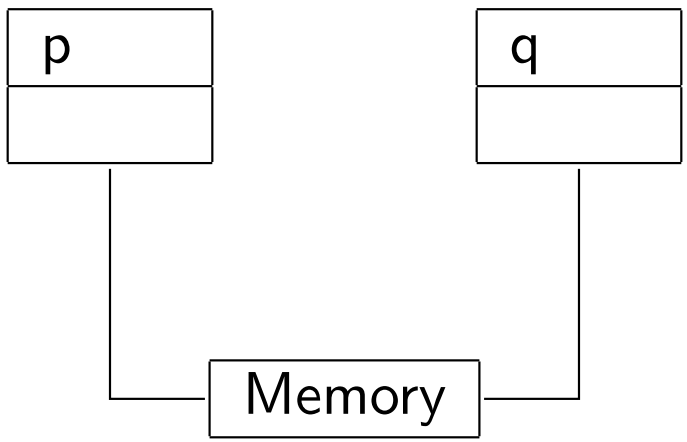
$$val\big((r,i,a,x)\big) = val\Big(Max_{\leq}\big[\{(w,j,a,y) | (w,j,a,y) <_{PO} (r,i,a,x)\} \cup \{(w,j,a,y) | (w,j,a,y) \leq (r,i,a,x)\}\big]\Big)$$

- LoadOp: $r \in (r,i,*,*), o \in (*,i,*,*)\ r <_{PO} o \Rightarrow r \leq o$
- StoreStore: $w_1, w_2 \in (write,i,*,*)\ w_1 <_{PO} w_2 \Rightarrow w_1 \leq w_2$

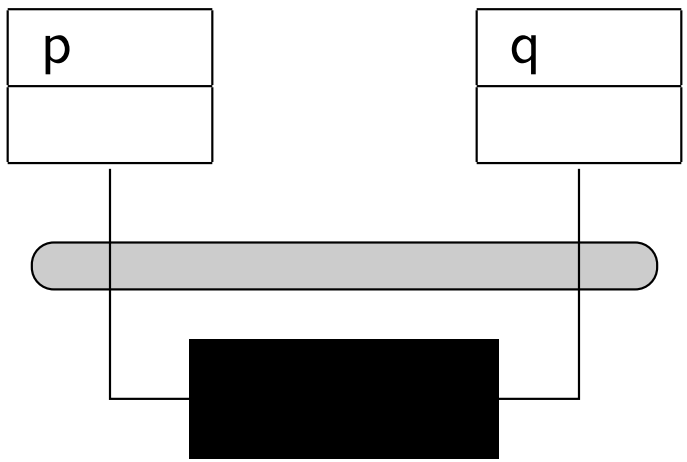
State of the Art: Operational

```
module RefPRAM( [...] ) {
    [...]
    for(i = 0 .. P-1) do || {
        always { // Dist: broadcasts writes to connected FIFOs & sends reads to own FIFO
            if(reqMem[i] & !someFIFOfull[i]) {
                emit(ackMem[i]);
                if(writeMem[i]) { emit(doneMem[i]); }
                for(j = 0 .. P-1) {
                    if((j==i) | writeMem[i]) {
                        FIFOpush[j][i] = true;
                        FIFOinp[j][i] = (writeMem[i], i, adrBus[i], dataBus[i]);
                    }
                }
            }
        }
        || for(j = 0 .. P-1) do || { // FIFO
            always { if(FIFOisfull[j][i]) { emit(someFIFOfull[i]); } }
            || fifo : FIFO(FIFOpop[i][j], FIFOpush[i][j], FIFOisempty[i][j], FIFOisfull[i][j], FIFOinp[i][j], →
                FIFOoutp[i][j]);
        }
        || always { // Arbiter (Simply choose from N components)
            choose(o1 = 0 .. P-1) {
                if(!FIFOisempty[i][o1]) {
                    memIn[i] = (true, FIFOoutp[i][o1]);
                    emit(FIFOpop[i][o1]);
                }
            }
        }
        [...]
```

State of the Art: View-based



State of the Art: View-based



Motivation

- Distribution
  - Consumer CPU, Distributed Computing, Cloud, Embedded
- Modelling approaches
  - Different, incompatible, limited
- Critical software
  - Embedded controllers, Operating system, ...

Motivation

- Distribution
  - Consumer CPU, Distributed Computing, Cloud, Embedded
- Modelling approaches
  - Different, incompatible, limited
- Critical software
  - Embedded controllers, Operating system, ...

$\Rightarrow$ Formal analysis

Contributions

Approaches covering multiple models in a uniform way

Contributions

Approaches covering multiple models in a uniform way

1. Testing as SAT
2. Temporal Logic
3. Operational semantics
4. State transition system

Testing Problem

Complexity

- Testing problem:
  Is memory trace consistent with memory model?

Testing Problem

Complexity

- Testing problem:
  Is memory trace consistent with memory model?

P	**Q**
write(x,1)	write(y,1)
read(y,0)	read(x,0)

Testing Problem

Complexity

- Testing problem:
  Is memory trace consistent with memory model?

P	**Q**
write(x,1)	write(y,1)
read(y,0)	read(x,0)

- Complexity analysis
  - NP complete for many models
  - Testing is in NP (SAT)

INFORMATIK

TECHNISCHE UNIVERSITÄT KAISERSLAUTERN

Testing Problem

SAT encoding

- View-based definitions: Serial view
  - subset of operations

Testing Problem

SAT encoding

- View-based definitions: Serial view
  - subset of operations
- Encoding
  - Variables: execution, serial view
  - $EXE(\mathcal{T})$: existence of execution
    - "writes-to" relation

Testing Problem

SAT encoding

- View-based definitions: Serial view
  - subset of operations
- Encoding
  - Variables: execution, serial view
  - $EXE(\mathcal{T})$: existence of execution
    - "writes-to" relation
  - $SV(\mathcal{T}, O, <)$: serial view property of execution
    - total, transitive, extends $<$

Testing Problem

SAT encoding

- View-based definitions: Serial view
  - subset of operations
- Encoding
  - Variables: execution, serial view
  - $EXE(\mathcal{T})$: existence of execution
    - "writes-to" relation
  - $SV(\mathcal{T}, O, <)$: serial view property of execution
    - total, transitive, extends $<$
- Experiments
  - Z3 SMT solver (SMT-LIBv2 encoding)

Temporal Logic

Linear temporal logic

- Temporal Logic well suited for concurrency

Temporal Logic

Linear temporal logic

- Temporal Logic well suited for concurrency
- Linear temporal logic (LTL):

$$G\varphi,\ F\varphi,\ X\varphi,\ \varphi U\psi$$

Temporal Logic

Linear temporal logic

- Temporal Logic well suited for concurrency
- Linear temporal logic (LTL):

  $$G\varphi,\ F\varphi,\ X\varphi,\ \varphi U\psi$$

- Event variables:
  - Read, Write: Interface
  - Observed: Internal


Temporal Logic

Example

- Local consistency: Observe own writes immediately

$$\mathsf{G}\Big(w \rightarrow \forall_q \mathit{Observed}_q(w) \Big) \tag{1}$$


Temporal Logic

Example

- Local consistency: Observe own writes immediately

$$\mathsf{G}\Big(w \rightarrow \forall_q \mathit{Observed}_q(w) \Big) \tag{1}$$

- Slow consistency: Observe another's writes to the same location in the same order

$$\mathsf{G}\Big(\mathit{Observed}(w) \rightarrow \mathsf{XG}\nexists_{w' <_P w} \mathit{Observed}(w') \Big) \tag{2}$$

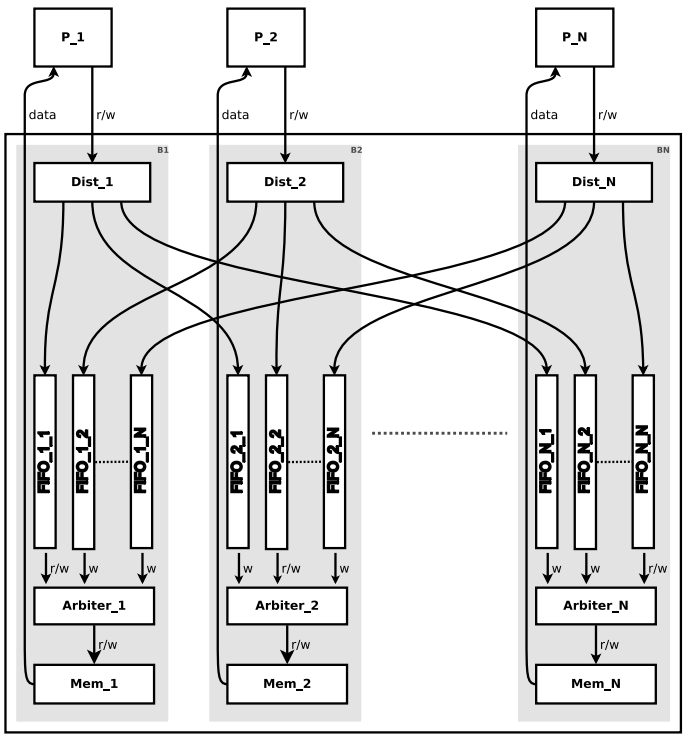
- Experiments: NuSMV, NuXMV model checker

INFORMATIK

TECHNISCHE UNIVERSITÄT KAISERSLAUTERN

Operational Semantics: PRAM

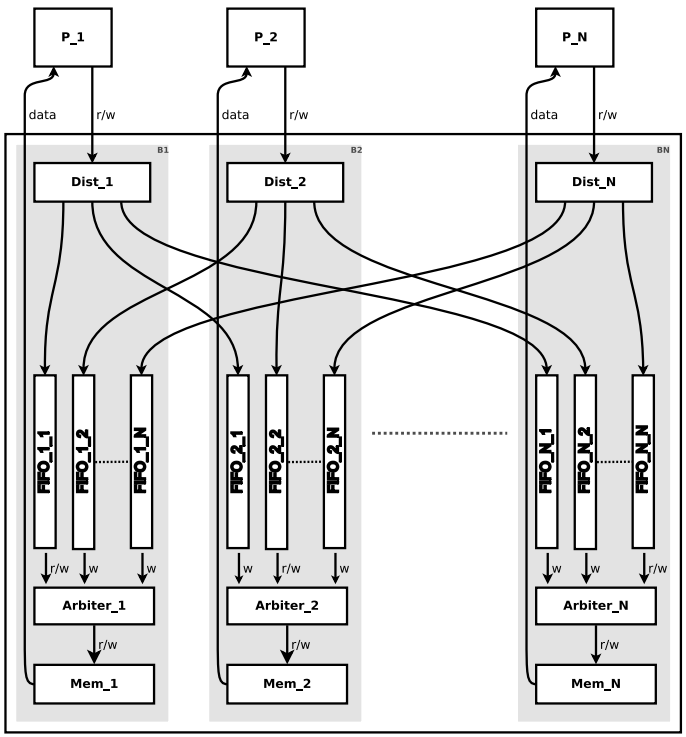
$$\forall_p \exists <_{sv} = \mathit{SerialView}\left(<_P \middle| (*,*,*,p,*) \cup (w,*,*,*,*)\right)$$



- Reference Machine
  - Correctness
  - Completeness

Operational Semantics: PRAM

$$\forall_p \exists <_{sv} = \mathit{SerialView}\left(<_P \middle| (*,*,*,p,*) \cup (w,*,*,*,*)\right)$$



- Reference Machine
  - Correctness
  - Completeness
- Applications
  - Simulation
  - Synthesis

State Transition System

- Semantics of programming language & memory model

State Transition System

- Semantics of programming language & memory model
- State $\Sigma = \langle \mathcal{E} \,|\, \mathcal{S}_1, \ldots, \mathcal{S}_i, \ldots, \mathcal{S}_n \rangle$

State Transition System

- Semantics of programming language & memory model
- State $\Sigma = \langle \mathcal{E} \,|\, \mathcal{S}_1, \ldots, \mathcal{S}_i, \ldots, \mathcal{S}_n \rangle$
- Structural operational semantics

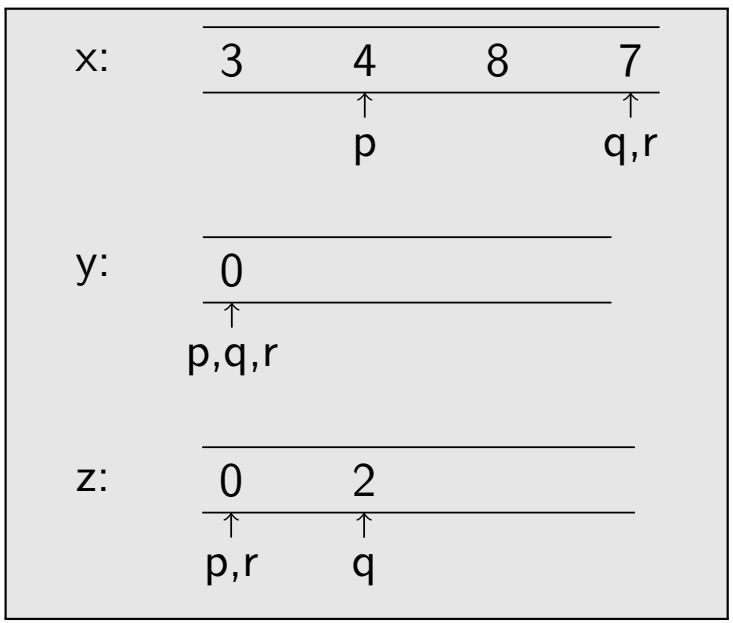
$$\frac{\alpha}{\langle \mathcal{E} \,|\, \mathcal{S}_1, \ldots, \mathcal{S}_p, \ldots, \mathcal{S}_n \rangle \mapsto \langle \mathcal{E}' \,|\, \mathcal{S}_1, \ldots, \mathcal{S}'_p, \ldots, \mathcal{S}_n \rangle}$$

INFORMATIK

TECHNISCHE UNIVERSITÄT KAISERSLAUTERN

State Transition System

- $\mathcal{E}$: environment suited to express different memory models



State Transition System

- Reading

$$\frac{\text{NextVar}(e) = v \neq \bot \quad \wedge \quad \mathcal{E}(v, p) = d \quad \wedge \quad e' = e\big|_d^v}{\langle \mathcal{E} \,|\, \mathcal{S}_1, \ldots, \mathcal{S}_p : \texttt{x = e}, \ldots, \mathcal{S}_n \rangle \mapsto \langle \mathcal{E} \,|\, \mathcal{S}_1, \ldots, \mathcal{S}'_p : \texttt{x = e'}, \ldots, \mathcal{S}_n \rangle}$$

- Writing (SC)

$$\frac{\begin{array}{c} d \in \mathbb{Z} \quad \wedge \quad v \in \mathcal{V}_g \quad \wedge \quad \mathit{Len}(\mathcal{E}, v) = i \\ \wedge \quad \mathcal{E}' = \mathit{Ins}(\mathit{MovAll}(\mathcal{E}, v, i+1), v, d, i+1) \end{array}}{\langle \mathcal{E} \,|\, \mathcal{S}_1, \ldots, \texttt{v=d;}, \ldots, \mathcal{S}_n \rangle \mapsto \langle \mathcal{E}' \,|\, \mathcal{S}_1, \ldots, \texttt{nothing;}, \ldots, \mathcal{S}_n \rangle}$$

Comparison

	coverage	simulation	state	state space
SAT	trace	no	-	-
LTL	program	no	min	full
OpS	program	yes	full[1]	optimized[2]
STS	program	yes	small	optimized

[1] including processor state

[2] only consistent states

Conclusion

Modeling approaches for multitude of models

- Testing Problem SAT encoding
- Temporal Logic
- Operational Semantics
- State Transition System

Thank you for your attention.

SAT encoding I

$$(w,x,1).(r,x,2) \parallel (w,x,2).(r,x,1)$$

$$EXE(\mathcal{T}_\alpha) \wedge SV(\mathcal{T}_\alpha, <_{PO})$$

```
(declare-fun ex (Int Int) Bool) ; Varaible ex_i,j
(declare-fun sv (Int Int) Bool) ; Variable sv_i,j
; ### Execution ##########################################
; Exists write for read
(assert (ex 0 3)) ; wx1 -> rx1
(assert (ex 2 1)) ; wx2 -> rx2
; Only one write for read (same variable & data)
; > none, as only one write matches read
; ### SerialViews ########################################
; ## SV: forall (*,*,*,*,*), respecting <_PO
; Total Order & AntiSymmetry
(assert (xor (not (sv 0 1) ) (not (sv 1 0))))
(assert (xor (not (sv 0 2) ) (not (sv 2 0))))
(assert (xor (not (sv 0 3) ) (not (sv 3 0))))
(assert (xor (not (sv 1 2) ) (not (sv 2 1))))
(assert (xor (not (sv 1 3) ) (not (sv 3 1))))
(assert (xor (not (sv 2 3) ) (not (sv 3 2))))
; Transitivity
(assert (=> (and (sv 0 1) (sv 1 2)) (sv 0 2) ))
(assert (=> (and (sv 0 1) (sv 1 3)) (sv 0 3) ))
(assert (=> (and (sv 0 2) (sv 2 1)) (sv 0 1) ))
(assert (=> (and (sv 0 2) (sv 2 3)) (sv 0 3) ))
(assert (=> (and (sv 0 3) (sv 3 1)) (sv 0 1) ))
(assert (=> (and (sv 0 3) (sv 3 2)) (sv 0 2) ))
(assert (=> (and (sv 1 0) (sv 0 2)) (sv 1 2) ))
(assert (=> (and (sv 1 0) (sv 0 3)) (sv 1 3) ))
```

SAT encoding II

```
(assert (=> (and (sv 1 2) (sv 2 0)) (sv 1 0) ))
(assert (=> (and (sv 1 2) (sv 2 3)) (sv 1 3) ))
(assert (=> (and (sv 1 3) (sv 3 0)) (sv 1 0) ))
(assert (=> (and (sv 1 3) (sv 3 2)) (sv 1 2) ))
(assert (=> (and (sv 2 0) (sv 0 1)) (sv 2 1) ))
(assert (=> (and (sv 2 0) (sv 0 3)) (sv 2 3) ))
(assert (=> (and (sv 2 1) (sv 1 0)) (sv 2 0) ))
(assert (=> (and (sv 2 1) (sv 1 3)) (sv 2 3) ))
(assert (=> (and (sv 2 3) (sv 3 0)) (sv 2 0) ))
(assert (=> (and (sv 2 3) (sv 3 1)) (sv 2 1) ))
(assert (=> (and (sv 3 0) (sv 0 1)) (sv 3 1) ))
(assert (=> (and (sv 3 0) (sv 0 2)) (sv 3 2) ))
(assert (=> (and (sv 3 1) (sv 1 0)) (sv 3 0) ))
(assert (=> (and (sv 3 1) (sv 1 2)) (sv 3 2) ))
(assert (=> (and (sv 3 2) (sv 2 0)) (sv 3 0) ))
(assert (=> (and (sv 3 2) (sv 2 1)) (sv 3 1) ))
; SV refines <_PO
(assert (sv 0 1)) ; wx1 <_PO rx2
(assert (sv 2 3)) ; wx2 <_PO rx1
; Writes-To implies SV & no intermediate write
; wx1 -> rx1
(assert (or (not (ex 0 3)) (sv 0 3)))
(assert (or (not (ex 0 3)) (not (sv 0 0)) (not (sv 0 3))))
(assert (or (not (ex 0 3)) (not (sv 0 2)) (not (sv 2 3))))
; wx2 -> rx2
(assert (or (not (ex 2 1)) (sv 2 1)))
(assert (or (not (ex 2 1)) (not (sv 2 0)) (not (sv 0 1))))
(assert (or (not (ex 2 1)) (not (sv 2 2)) (not (sv 2 1))))
(check-sat)
```

LTL (SMV encoding): invariants I

```
-----------------------------------------------------------------------
-- INVARIANTS --

-- read and write only occur if processor is active
[[repeatline %p=MIN_P..MAX_P]]
    INVAR (!step[%p]) -> (!read[%p] & !write[%p]);
[[/repeatline]]

-- read and write never occur at the same time
[[repeatline %p=MIN_P..MAX_P]]
    INVAR !(write[%p] & read[%p]);
[[/repeatline]]

-- readValue is undefined while no read occurs
[[repeatline %i=MIN_P..MAX_P]]
    INVAR (!read[%i]) -> (readValue[%i]=UNDEFINED);
[[/repeatline]]

-----------------------------------------------------------------------
-- Fairness: step[p] has to hold infinitely often (for all processors p)
[[repeatline %p=MIN_P..MAX_P]]
    JUSTICE step[%p];
[[/repeatline]]
-----------------------------------------------------------------------
```

LTL (SMV encoding): base I

```
-----------------------------------------------------------------------
-- Processing Causality
(
[[forall %p=MIN_P..MAX_P]][[forall %q=MIN_P..MAX_P]]
[[forall %i=MIN_ID..MAX_ID]][[forall %l=MIN_LOC..MAX_LOC]]
        ( (!(proc[%p] & (procProcess[%p] = %q)
        & (procId[%p] = %i) & (procLocation[%p] = %l)))
        U (write[%q] & (writeId[%q] = %i) & (writeLocation[%q] = %l)) )
        | G(!(proc[%p] & (procProcess[%p] = %q)
        & (procId[%p] = %i) & (procLocation[%p] = %l)))
[[/forall]][[/forall]]
[[/forall]][[/forall]]
)
-----------------------------------------------------------------------
-- Processing Uniqueness
& (G
    [[forall %p=MIN_P..MAX_P]][[forall %q=MIN_P..MAX_P]]
    [[forall %i=MIN_ID..MAX_ID]]
        (proc[%p] & (procProcess[%p] = %q) & (procId[%p] = %i))
        -> X G !(proc[%p] & (procProcess[%p] = %q) & (procId[%p] = %i))
    [[/forall]]
    [[/forall]][[/forall]])

-- Read Initial
& ([[forall %p=MIN_P..MAX_P]][[forall %l=MIN_LOC..MAX_LOC]]
        ( (read[%p] & (readLocation[%p]=%l) -> (readValue[%p]=UNDEFINED))
        U (proc[%p] & procLocation[%p]=%l) )
    | G(read[%p] & (readLocation[%p]=%l) -> (readValue[%p]=UNDEFINED))
[[/forall]][[/forall]])
-----------------------------------------------------------------------
-- Read Causality
& (G
    [[forall %q=MIN_P..MAX_P]][[forall %i=MIN_ID..MAX_ID]]
    [[forall %l=MIN_LOC..MAX_LOC]][[forall %v=MIN_VAL..MAX_VAL]]
```

LTL (SMV encoding): base II

```
      (write[%q] & (writeId[%q]=%i)
    & (writeLocation[%q]=%l) & (writeValue[%q]=%v))
      -> G [[forall %p=MIN_P..MAX_P]]
                        (proc[%p] & (procProcess[%p] = %q) & (procId[%p]=%i))
                        -> ( ( ( ( read[%p] & (readLocation[%p]=%l))
                     -> readValue[%p]=%v )
                U (proc[%p] & (procLocation[%p]=%l)
                     & ( (procProcess[%p] != %q) | (procId[%p] != %i) )) )
                | ( G ( ( read[%p] &
                        (readLocation[%p]=%l)) -> readValue[%p]=%v ) ) )
                [[/forall]]
    [[/forall]][[/forall]]
    [[/forall]][[/forall]])
-------------------------------------------------------------------------
```

LTL (SMV encoding): local consistency I

```
-------------------------------------------------------------------------
-- Include Base Spec
(
#include(spec-base.ltl)
)
-- Local Causality
& ([[forall %p=MIN_P..MAX_P]]
        G ((write[%p]) ->
        (proc[%p] & (procProcess[%p]=%p) & (procId[%p]=writeId[%p]))))
[[/forall]])
-------------------------------------------------------------------------
```

LTL (SMV encoding): Peterson Mutex example I

```
-------------------------------------------------------------------------
-- Petersons Mutual Exclusion Algorithm ---------------------------------
-------------------------------------------------------------------------

-- LOCATIONS: flag0, flag1, turn, data

-- 00:  write (0,1)                                  flag[self] <- T
-- 01:  write (2,1)                                  turn <- other
-- 02:  reg = read (1)                         if (!flag[other]) goto 6
-- 03:  if(reg=0) goto 6                       "
-- 04:  reg = read (2)                         if (turn=other) goto 2
-- 05:  if(reg=1) goto 2                       "
-- 06:  reg = read (3)                         data++;
-- 07:  write (3,reg+1)                  "
-- 08:  write (0,0)                                  flag[self] <- F
-- 09:  goto 9

-------------------------------------------------------------------------

#define MIN_P                   0
#define MAX_P                   1
#define MIN_LOC         0
#define MAX_LOC         3
#define MIN_VAL         0
#define MAX_VAL         2
#define UNDEFINED       0
#define MIN_ID                  0
#define MAX_ID                  10

-------------------------------------------------------------------------

-- Processor --
MODULE Processor(id, step, write, writeLocation, writeValue, read,
                 readLocation, readValue)
```

LTL (SMV encoding): Peterson Mutex example II

```
VAR
        reg : MIN_VAL..MAX_VAL;
        pc : 0..9;
        other: MIN_P..MAX_P;
ASSIGN
        other := ((id=0)?1:0);
        init(pc) := 0;
        next(pc) :=
                case
                        !step : pc;
                        (pc = 3) & (reg = 0) : 6;
                        (pc = 5) & (reg != id) : 2;
                        pc >= 9 : 9;
                        TRUE : (pc+1);
                esac;
        init(reg) := UNDEFINED;
        next(reg) :=
                case
                        !step : reg;
                        pc = 2 : readValue;
                        pc = 4 : readValue;
                        pc = 6 : readValue;
                        TRUE : reg;
                esac;
        write :=
                case
                        !step: FALSE;
                        TRUE: (pc = 0 | pc = 1 | pc = 7 | pc = 8);
                esac;
        writeLocation :=
                case
                        !step: 0;
                        pc = 0 : id;
                        pc = 1 : 2;
```

LTL (SMV encoding): Peterson Mutex example III

```
                        pc = 7 : 3;
                        pc = 8 : id;
                        TRUE : 0;
                esac;
        writeValue :=
                case
                        !step: 0;
                        pc = 0 : 1;
                        pc = 1 : other;
                        pc = 7 : (reg>=MAX_VAL)?reg:(reg+1);
                        pc = 8 : 0;
                        TRUE : 0;
                esac;
        read :=
                case
                        !step: FALSE;
                        TRUE: (pc = 2 | pc = 4 | pc = 6);
                esac;
        readLocation :=
                case
                        !step: 0;
                        pc = 2 : other;
                        pc = 4 : 2;
                        pc = 6 : 3;
                        TRUE : 0;
                esac;
-----------------------------------------------------------------------
MODULE main
VAR
        step: array MIN_P..MAX_P of boolean;
        write: array MIN_P..MAX_P of boolean;
        writeId: array MIN_P..MAX_P of MIN_ID..MAX_ID;
        writeLocation: array MIN_P..MAX_P of MIN_LOC..MAX_LOC;
        writeValue: array MIN_P..MAX_P of MIN_VAL..MAX_VAL;
```

LTL (SMV encoding): Peterson Mutex example IV

```
        read: array MIN_P..MAX_P of boolean;
        readValue: array MIN_P..MAX_P of MIN_VAL..MAX_VAL;
        readLocation: array MIN_P..MAX_P of MIN_LOC..MAX_LOC;
        proc: array MIN_P..MAX_P of boolean;
        procProcess: array MIN_P..MAX_P of MIN_P..MAX_P;
        procId: array MIN_P..MAX_P of MIN_ID..MAX_ID;
        procLocation: array MIN_P..MAX_P of MIN_LOC..MAX_LOC;
VAR
[[repeatline %i=MIN_P..MAX_P]]
    p%i : Processor(%i, step[%i], write[%i], writeLocation[%i],
            writeValue[%i], read[%i], readLocation[%i], readValue[%i]);
[[/repeatline]]
ASSIGN
[[repeatline %i=MIN_P..MAX_P]]  init(writeId[%i]) := 0;[[/repeatline]]
[[repeatline %i=MIN_P..MAX_P]]
    next(writeId[%i]) :=
      (write[%i] & (writeId[%i]<MAX_ID)) ? (writeId[%i]+1) : writeId[%i];
[[/repeatline]]

-----------------------------------------------------------------------

#include(invariants)

-----------------------------------------------------------------------
-- First Test: CC
LTLSPEC
(
#include(spec-cc.ltl)
)
-- Property to check
->
( G  !((p0.pc=6) & (p1.pc=6)) )
;
--EXPECTED: FALSE (counterexample <=> possible behavior)
```

LTL (SMV encoding): Peterson Mutex example V

```
--PROVEN (BMC CounterExample)
------------------------------------------------------------------------
-- Second Test: PRAM
LTLSPEC
(
#include(spec-pram.ltl)
)
-- Property to check
->
( G  !((p0.pc=6) & (p1.pc=6)) )
;
--EXPECTED: FALSE (counterexample <=> possible behavior)
--PROVEN (BMC CounterExample)
------------------------------------------------------------------------
-- Third Test: SC
LTLSPEC
(
#include(spec-sc.ltl)
)
-- Property to check
->
( G  !((p0.pc=6) & (p1.pc=6)) )
;
--EXPECTED: TRUE (no counterexample)
--PROVEN (BMC Depth 27)
------------------------------------------------------------------------
```