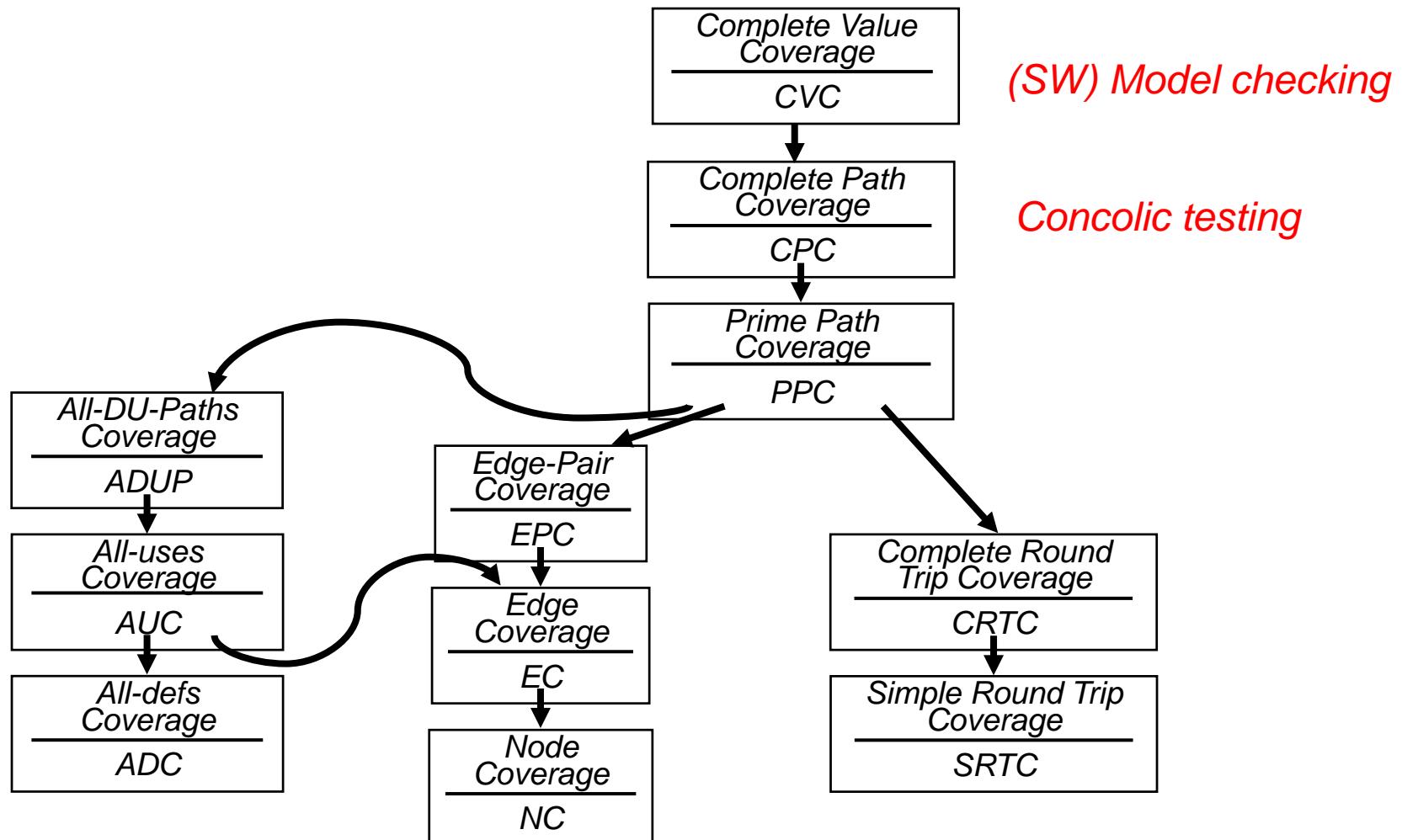


The Spin Model Checker : Part I

Moonzoo Kim

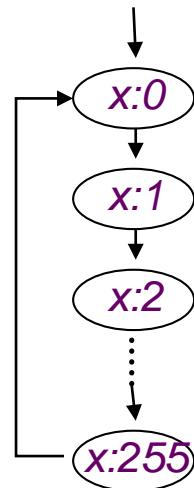
KAIST

Hierarchy of SW Coverage Criteria



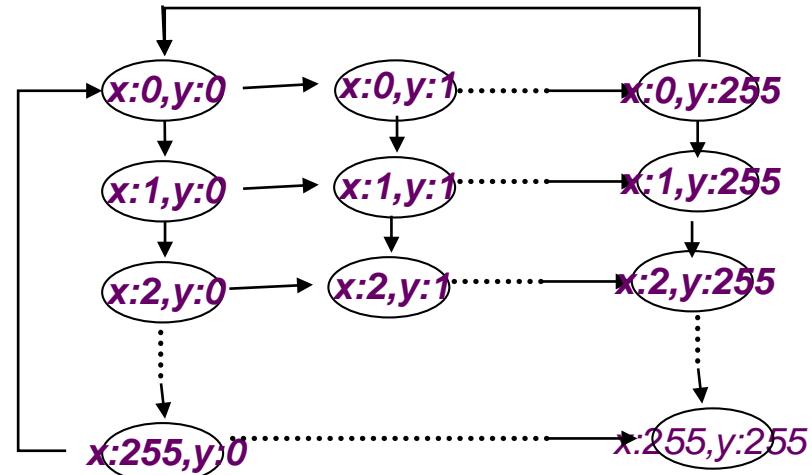
Model Checker Analyzes All Possible Scheduling

```
active type A() {  
byte x;  
again:  
    x++;  
    goto again;  
}
```

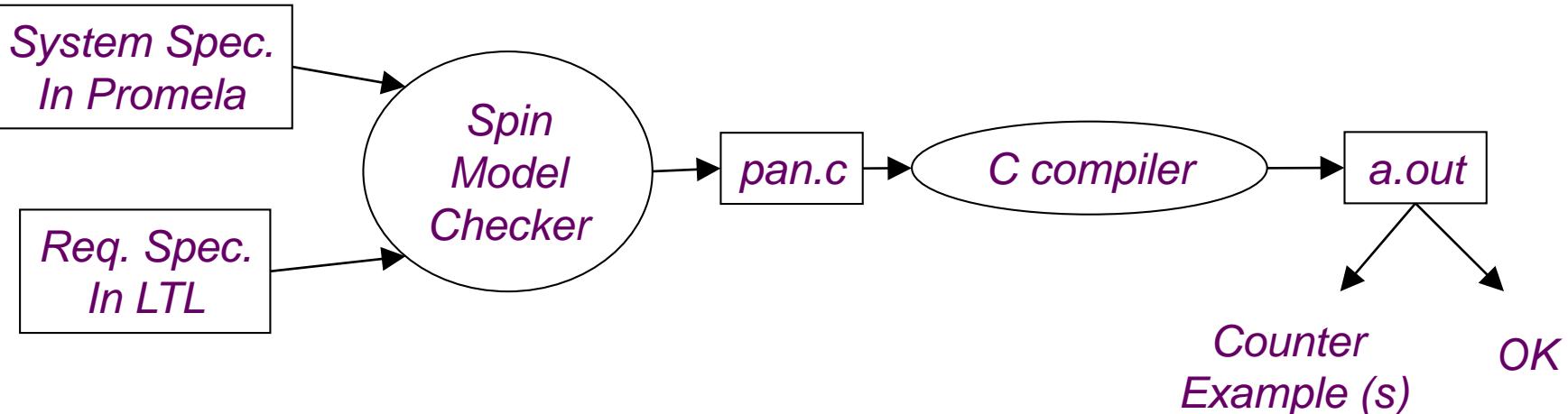


```
active type A() {  
byte x;  
again:  
    x++;  
    goto again;  
}
```

```
active type B() {  
byte y;  
again:  
    y++;  
    goto again;  
}
```

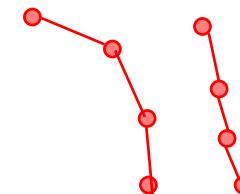


Overview of the Spin Architecture



A few characteristics of Spin

- + Promela allows a finite state model only
- + Asynchronous execution
- + Interleaving semantics for concurrency
- + 2-way process communication
- + Non-determinism
- + Promela provides (comparatively) rich set of constructs such as variables and message passing, dynamic creation of processes, etc



Tcl GUI of SPIN (ispin.tcl): Edit Window

mobile1.pml

Spin Version 6.2.7 -- 2 March 2014 :: iSpin Version 1.1.0 -- 7 June 2012

Edit/View Simulate / Replay Verification Swarm Run <Help> Save Session Restore Session <Quit>

Open... ReOpen Save Save As... Syntax Check Redundancy Check Symbol Table Find: []

```
1    /*
2    * Model of cell-phone handoff strategy in a mobile network.
3    * A translation from the pi-calculus description of this
4    * model presented in:
5    * Fredrik Orava and Joachim Parrow, 'An algebraic verification
6    * of a mobile network,' Formal aspects of computing, 4:497-543 (199
2).
7    * For more information on this model, email: joachim@it.kth.se
8    *
9    * See also the simplified version of this model in mobile2
10
11   * to perform the verification:
12   *   $ spin -a mobile1
13   *   $ cc -o pan pan.c
14   *   $ pan -a
15   */
16
17   mtype = { data, ho_cmd, ho_com, ho_acc, ho_fail, ch_rel, white, red,
blue };
18
19   byte a_id, p_id; /* ids of processes referred to in the property */
20
```

Automata View zoom in zoom out

Select:
p_CC
p_HC
p_MSC
p_BS
p_MS
p_P
p_Q
p_System
p_top
p_bot
claim_ltl_0

```
3 gcc -o pan pan.c
4 ./pan -D | dot > dot.tmp
5 C:/spin/mobile1.pml:1
6 spin -o3 -a mobile1.pml
Itl ltl_0: (! (! ([] (<> (((BS[a_id]@progress)) || ((BS[p_id]@progress))))))) || ([] (!! (<> (inp?[red])))) || (<> (out?[red])))
7 gcc -o pan pan.c
8 ./pan -D | dot > dot.tmp
```

Tcl GUI of SPIN (ispin.tcl): Verification Window

2threads.pml

Spin Version 6.2.7 -- 2 March 2014 :: iSpin Version 1.1.0 -- 7 June 2012

Edit/View Simulate / Replay Verification Swarm Run <Help> Save Session Restore Session <Quit>

Safety	Storage Mode	Search Mode	Remove	Remove
<input checked="" type="radio"/> safety	<input checked="" type="radio"/> exhaustive	<input checked="" type="radio"/> depth-first search	Advanced: Error Trapping	Advanced: Parameters
<input checked="" type="checkbox"/> + invalid endstates (deadlock)	<input type="checkbox"/> + minimized automata (slow)	<input checked="" type="checkbox"/> + partial order reduction	<input checked="" type="radio"/> don't stop at errors	Physical Memory Available (in Mbytes): 1024 explain
<input checked="" type="checkbox"/> + assertion violations	<input type="checkbox"/> + collapse compression	<input type="checkbox"/> + bounded context switching with bound: 0	<input type="radio"/> stop at error nr: 1	Estimated State Space Size (states x 10 ³): 1000 explain
<input type="checkbox"/> + xr/xs assertions	<input type="radio"/> hash-compact <input type="radio"/> bitstate/supertrace	<input type="checkbox"/> + iterative search for short trail	<input checked="" type="checkbox"/> save all error-trails	Maximum Search Depth (steps): 10000 explain
Liveness	Never Claims	<input type="radio"/> breadth-first search	<input type="checkbox"/> add complexity profiling	Nr of hash-functions in Bitstate mode: 3 explain
<input type="radio"/> non-progress cycles	<input checked="" type="radio"/> do not use a never claim or ltl property	<input checked="" type="checkbox"/> + partial order reduction	<input type="checkbox"/> compute variable ranges	Size for Minimized Automaton: 100 explain
<input type="radio"/> acceptance cycles	<input type="radio"/> use claim	<input checked="" type="checkbox"/> report unreachable code	A Full Channel	Extra Verifier Generation Options: explain
<input type="checkbox"/> enforce weak fairness constraint	claim name (opt): pan.out	<input type="checkbox"/> Save Result in: pan.out	<input checked="" type="radio"/> blocks new msgs	Extra Compile-Time Directives: -O2 explain
			<input type="checkbox"/> loses new msgs	Extra Run-Time Options: explain

Run Stop State Tables Clear Help

```
1 byte x=0, y=0, z =0;
2
3 active proctype p() {
4     x=y+1;
5     y=z+1;
6     z= x+1;
7 }
8
9 active proctype q() {
10    y=z+1;
11    z=x+1;
12    x=y+1;
13 }
14
15 init {
16 timeout-> printf("x:%d,y:%d,z:%d\n",x,y,z);
17 assert(0);
18 }
19
20 /*
21 spin -a 2threads.pml
22 gcc -DMEMLIM=1024 -O2 -DXUSAFE -DSAFETY -DNOCLAIM -w -o pa
n pan.c
```

Full statespace search for:
never claim - (not selected)
assertion violations +
cycle checks - (disabled by -DSAFETY)
invalid end states +

State-vector 24 byte, depth reached 12, errors: 11
115 states, stored
7 states, matched
122 transitions (= stored+matched)
0 atomic steps
hash conflicts: 0 (resolved)

Stats on memory usage (in Megabytes):
0.004 equivalent memory usage for states (stored*(State-vector + overhead))
0.292 actual memory usage for states
64.000 memory used for hash table (-w24)
0.343 memory used for DFS stack (-m10000)
64.539 total actual memory usage

unreached in proctype p
(0 of 4 states)



Tcl GUI of SPIN (ispin.tcl): Simulation Window

2threads.pml

Spin Version 6.2.7 -- 2 March 2014 :: iSpin Version 1.1.0 -- 7 June 2012

Edit/View Simulate / Replay Verification Swarm Run <Help> Save Session Restore Session <Quit>

Mode A Full Channel Output Filtering (reg. exps.) (Re)Run

Random, with seed: 123 blocks new messages process ids:

Interactive (for resolution of all nondeterminism) loses new messages queue ids:

Guided, with trail: C:/Dropbox/classes/Spring browse MSC+stmtnt var names:

initial steps skipped: 0 MSC max text width 20 tracked variable:

maximum number of steps: 10000 MSC update delay 25 track scaling:

Background command executed:
spin -p -s -r -X -v -n123 -l -g -k C
:/Dropbox/classes/Spring14-cs4
92B/2threads/2threads.pml5.trail
-u10000 2threads.pml

Save in: msc.ps

1 int x=0, y=0, z=0;
2
3 active proctype p() {
4 x=y+1;
5 y=z+1;
6 z=x+1;
7 }
8
9 active proctype q() {
10 y=z+1;
11 z=x+1;
12 x=y+1;
13 }

variable values, step 1 [queues, step 1]
y = 1 1: proc 1 (q:1) 2threads.pml:10 (state 1) [y = (z+1)]
x:5,y:4,z:6 2: proc 0 (p:1) 2threads.pml:4 (state 1) [x = (y+1)]
3: proc 1 (q:1) 2threads.pml:11 (state 2) [z = (x+1)]
4: proc 0 (p:1) 2threads.pml:5 (state 2) [y = (z+1)]
5: proc 1 (q:1) 2threads.pml:12 (state 3) [x = (y+1)]
6: proc 0 (p:1) 2threads.pml:6 (state 3) [z = (x+1)]
7: proc 2 (:init::1) 2threads.pml:16 (state 1) [(timeout)]
spin: 2threads.pml:17, Error: assertion violated
spin: text of failed assertion: assert(0)
#processes: 3
9: proc 2 (:init::1) 2threads.pml:17 (state 3)

Overview of the Promela

```
byte x;  
chan ch1= [3] of {byte};
```

*Global variables
(including channels)*

```
active[2] proctype A() {  
    byte z;  
    printf("x=%d\n",x);  
    z=x+1;  
    ch1!z  
}
```

*Process (thread)
definition and
creation*

```
proctype B(byte y) {  
    byte z;  
    ch1?z;  
}
```

*Another
process
definition*

```
Init {  
    run B(2);  
}
```

*System
initialization*

- Similar to C syntax but simplified
 - No pointer
 - No real datatype such as float or real
 - No functions
- Processes are communicating with each other using
 - Global variables
 - Message channels
- Process can be dynamically created
- Scheduler executes one process at a time using interleaving semantics



```
active[2] proctype A() {  
    byte x;  
    printf("A%d is starting\n");  
}  
  
proctype B() {  
    printf("B is starting\n");  
}  
  
Init {  
    run B();  
}
```

- run() operator creates a process and returns a newly created process ID
- There are 6 possible outcomes due to **non-deterministic** scheduling
 - + A0.A1.B, A0.B.A1
 - + A1.A0.B, A1.B.A0
 - + B.A0.A1, B.A1.A0
- In other words, process creation may **not** immediately start process execution



■ Basic types

- + bit
- + bool
- + Byte (8 bit unsigned integer)
- + short (16 bits signed integer)
- + Int (32 bits signed integer)

■ Arrays

- + bool x[10];

■ Records

- + typedef R { bit x; byte y;}

■ Default initial value of variables is 0

■ Most arithmetic (e.g., +, -), relational (e.g. >, ==) and logical operators of C are supported

- + bitshift operators are supported too.



- Promela spec generates only a finite state model because
 - + Max # of active process ≤ 255
 - + Each process has only finite length of codes
 - + Each variable is of finite datatype
 - + All message channels have bounded capability ≤ 255



- Each Promela statement is either
 - + executable:
 - + Blocked
- There are six types of statement
 - + Assignment: always executable
 - Ex. `x=3+x, x=run A()`
 - + Print: always executable
 - Ex. `printf("Process %d is created.\n", _pid);`
 - + Assertion: always executable
 - Ex. `assert(x + y == z)`
 - + Expression: depends on its value
 - Ex. `x+3>0, 0, 1, 2`
 - Ex. `skip, true`
 - + Send: depends on buffer status
 - Ex. `ch1!m` is executable only if `ch1` is not full
 - + Receive: depends on buffer status
 - Ex. `ch1?m` is executable only if `ch1` is not empty



■ An expression is also a statement

- + It is executable if it evaluates to non-zero
 - + 1 : always executable
 - + $1 < 2$: always executable
 - + $x < 0$: executable only when $x < 0$
 - + $x - 1$: executable only when $x \neq 0$

■ If an expression statement is blocked, it remains blocked until other process changes the condition

- + an expression e is equivalent to `while(!e);` in C



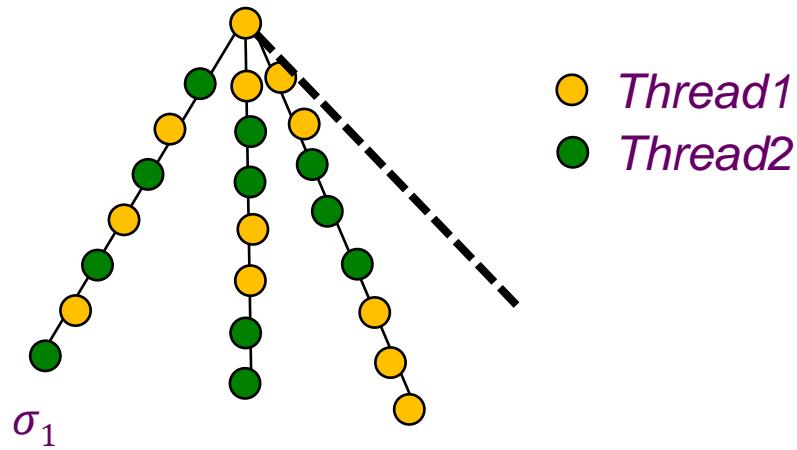
■ assert(expr)

- + assert is always executable
- + If expr is 0, SPIN detects this violation
- + assert is most frequently used checking method, especially as a form of invariance
 - ex. active proctype inv() { assert(x== 0);}
 - Note that inv() is equivalent to [] (x==0) in LTL with thanks to interleaving semantics

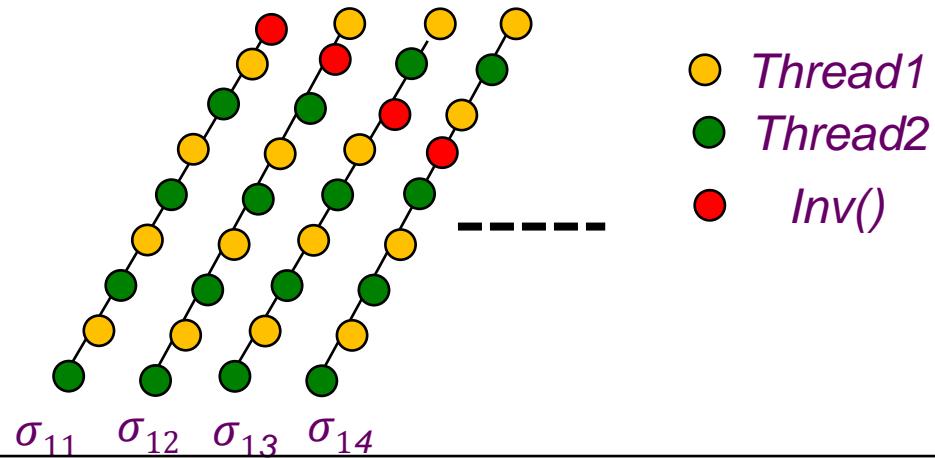


Generation of all possible interleaving scenarios

Original execution tree



After adding Inv() process



Therefore, just a single assert($x==0$) statement in `Inv()` can check if $x==0$ all the time



Program Execution Control

- Promela provides low-level control mechanism, i.e., goto and label as well as if and do
- Note that **non-deterministic** selection is supported
- else is predefined variable which becomes true if all guards are false; false otherwise

```
proctype A() {  
    byte x;  
    starting:  
    x= x+1;  
    goto starting;  
}
```

```
proctype A() {  
    byte x;  
    if  
        ::x<=0 -> x=x+1  
        ::x==0 -> x=1  
    fi  
}
```

```
proctype A() {  
    byte x;  
    do  
        :: x<=0 ->x=x+1;  
        :: x==0 ->x=1;  
        :: else -> break  
    od  
}
```

```
int i;  
for (i : 1 .. 10) {  
    printf("i =%d\n",i)  
}
```



Critical Section Example

```
bool lock;  
byte cnt;
```

```
active[2] proctype P() {  
    !lock -> lock=true;  
    cnt=cnt+1;  
    printf("%d is in the crt sec!\n",_pid);  
    cnt=cnt-1;  
    lock=false;  
}  
  
active proctype Invariant() {  
    assert(cnt <= 1);  
}
```

```
[root@moonzoo spin_test]# ls  
crit.pml  
[root@moonzoo spin_test]# spin -a crit.pml  
[root@moonzoo spin_test]# ls  
crit.pml pan.b pan.c pan.h pan.m pan.t  
[root@moonzoo spin_test]# gcc pan.c  
[root@moonzoo spin_test]# a.out  
pan: assertion violated (cnt<=1) (at depth 8)  
pan: wrote crit.pml.trail
```

Full statespace search for:

never claim - (none specified)
assertion violations +
acceptance cycles - (not selected)
invalid end states +

State-vector 36 byte, depth reached 16, errors: 1

119 states, stored

47 states, matched

166 transitions (= stored+matched)

0 atomic steps

hash conflicts: 0 (resolved)

4.879 memory usage (Mbyte)

```
[root@moonzoo spin_test]# ls
```

```
a.out crit.pml crit.pml.trail pan.b pan.c pan.h  
pan.m pan.t
```



Critical Section Example (cont.)

```
[root@moonzoo spin_test]# spin -t -p crit.pml
```

Starting P with pid 0

Starting P with pid 1

Starting Invariant with pid 2

```
1: proc 1 (P) line 5 "crit.pml" (state 1)      [(!lock)]
2: proc 0 (P) line 5 "crit.pml" (state 1)      [(!lock)]
3: proc 1 (P) line 5 "crit.pml" (state 2)      [lock = 1]
4: proc 1 (P) line 6 "crit.pml" (state 3)      [cnt = (cnt+1)]
    1 is in the crt sec!
5: proc 1 (P) line 7 "crit.pml" (state 4)      [printf('%d is in the crt sec!\n',_pid)]
6: proc 0 (P) line 5 "crit.pml" (state 2)      [lock = 1]
7: proc 0 (P) line 6 "crit.pml" (state 3)      [cnt = (cnt+1)]
    0 is in the crt sec!
8: proc 0 (P) line 7 "crit.pml" (state 4)      [printf('%d is in the crt sec!\n',_pid)]
```

spin: line 13 "crit.pml", Error: assertion violated

spin: text of failed assertion: assert((cnt<=1))

```
9: proc 2 (Invariant) line 13 "crit.pml" (state 1)      [assert((cnt<=1))]
```

spin: trail ends after 9 steps

#processes: 3

lock = 1

cnt = 2

```
9: proc 2 (Invariant) line 14 "crit.pml" (state 2) <valid end state>
```

```
9: proc 1 (P) line 8 "crit.pml" (state 5)
```

```
9: proc 0 (P) line 8 "crit.pml" (state 5)
```

3 processes created

Revised Critical Section Example

```
bool lock;
byte cnt;

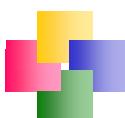
active[2] proctype P() {
    atomic{ !lock -> lock=true;
    cnt=cnt+1;
    printf("%d is in the crt sec!\n",_pid);
    cnt=cnt-1;
    lock=false;
}

active proctype Invariant() {
    assert(cnt <= 1);
}
```

[root@moonzoo revised]# a.out
Full statespace search for:

never claim	- (none specified)
assertion violations	+
acceptance cycles	- (not selected)
invalid end states	+

State-vector 36 byte, depth reached 14, errors: 0
62 states, stored
17 states, matched
79 transitions (= stored+matched)
0 atomic steps
hash conflicts: 0 (resolved)
4.879 memory usage (Mbyte)



Deadlocked Critical Section Example

```
bool lock;
byte cnt;

active[2] proctype P() {
    atomic{ !lock -> lock==true;}
    cnt=cnt+1;
    printf("%d is in the crt sec!\n",_pid);
    cnt=cnt-1;
    lock=false;
}
```

```
active proctype Invariant() {
    assert(cnt <= 1);
}
```

[[root@moonzoo deadlocked]# a.out
pan: invalid end state (at depth 3)

(Spin Version 4.2.7 -- 23 June 2006)
Warning: Search not completed
+ Partial Order Reduction

Full statespace search for:

never claim	-	(none specified)
assertion violations	+	
acceptance cycles	-	(not selected)
invalid end states	+	

State-vector 36 byte, depth reached 4, errors: 1
5 states, stored
0 states, matched
5 transitions (= stored+matched)
2 atomic steps
hash conflicts: 0 (resolved)

4.879 memory usage (Mbyte)



Deadlocked Critical Section Example (cont.)

```
[root@moonzoo deadlocked]# spin -t -p deadlocked_crit.pml
Starting P with pid 0
Starting P with pid 1
Starting Invariant with pid 2
 1: proc 2 (Invariant) line 13 "deadlocked_crit.pml" (state 1)
[assert((cnt<=1))]
 2: proc 2 terminates
 3: proc 1 (P) line 5 "deadlocked_crit.pml" (state 1) [(!(lock))]
 4: proc 0 (P) line 5 "deadlocked_crit.pml" (state 1) [(!(lock))]
spin: trail ends after 4 steps
#processes: 2
          lock = 0
          cnt = 0
 4: proc 1 (P) line 5 "deadlocked_crit.pml" (state 2)
 4: proc 0 (P) line 5 "deadlocked_crit.pml" (state 2)
3 processes created
```



Communication Using Message Channels

- Spin provides communications through various types of message channels
 - + Buffered or non-buffered (rendezvous comm.)
 - + Various message types
 - + Various message handling operators

■ Syntax

- + `chan ch1 = [2] of { bit, byte};`
 - `ch1!0,10;ch1!1,20`
 - `ch1?b,bt;ch1?1,bt`
 - + `chan ch2= [0] of {bit, byte}`
- Sender* →

(1,20)	(0,10)
--------	--------

 → *Receiver*



■ Basic channel inquiry

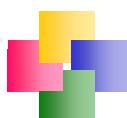
- `len(ch)`
- `empty(ch)`
- `full(ch)`
- `nempty(ch)`
- `nfull(ch)`

■ Additional message passing operators

- `ch?[x,y]`: polling only
- `ch?<x,y>`: copy a message without removing it
- `ch!>x,y`: sorted sending (increasing order)
- `ch??5,y`: random receiving
- `ch?x(y) == ch?x,y` (for user's understandability)

■ Be careful to use these operators inside of expressions

- They have side-effects, which spin may not allow



Faulty Data Transfer Protocol

(pg 27, data switch model proposed at 1981 at Bell labs)

mtype={*ini,ack,dreq,data,shutup,quiet,dead*}

chan *M*=[1] of {mtype};

chan *W*=[1] of {mtype};

active proctype *Mproc()*

{

W!ini; /* connection */
 M?ack; /* handshake */

timeout -> /* wait */
if /* two options: */
:: *W!shutup*; /* start shutdown */
:: *W!dreq*; /* or request data */

 do

 :: *M?data* -> *W!data*
 :: *M?data*-> *W!shutup*;

 break

od

fi;

M?shutup;
 W!quiet;
 M?dead;

}



active proctype *Wproc()* {

W?ini; /* wait for ini */
 M!ack; /* acknowledge */

do /* 3 options: */
:: *W?dreq*-> /* data requested */
 M!data /* send data */

:: *W?data*-> /* receive data */
 skip /* no response */

:: *W?shutup*->
 M!shutup; /* start shutdown */
 break

od;

W?quiet;
 M!dead;

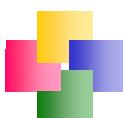
The Sieve of Eratosthenes (pg 326)

```
/*
The Sieve of Eratosthenes (c. 276-196 BC)
Prints all prime numbers up to MAX
*/
#define MAX    25
mtype = { number, eof };
chan root = [0] of { mtype, int };

init
{    int n = 2;

    run sieve(root, n);
    do
        :: (n < MAX) -> n++; root!number(n)
        :: (n >= MAX) -> root!eof(0); break
    od
}
```

```
proctype sieve(chan c; int prime)
{    chan child = [0] of { mtype, int };
    bool haschild; int n;
    printf("MSC: %d is prime\n", prime);
end: do
    :: c?number(n) ->
        if
            :: (n%prime) == 0 -> printf("MSC: %d
= %d*%d\n", n, prime, n/prime)
            :: else ->
                if
                    :: !haschild -> /* new prime */
                        haschild = true;
                        run sieve(child, n);
                    :: else ->
                        child!number(n)
                fi;
            fi
        :: c?eof(0) -> break
    od;
    if
        :: haschild -> child!eof(0)
        :: else
    fi
```



```
[moonzoo@verifier spin]$ spin sieve-of-eratosthenes.pml
```

```
2| MSC: 2 is prime
3| MSC: 3 is prime
MSC: 4 = 2*2
5| MSC: 5 is prime
MSC: 6 = 2*3
MSC: 8 = 2*4
7| MSC: 7 is prime
MSC: 9 = 3*3
MSC: 10 = 2*5
MSC: 12 = 2*6
MSC: 14 = 2*7
11| MSC: 11 is prime
MSC: 15 = 3*5
13| MSC: 13 is prime
MSC: 16 = 2*8
MSC: 18 = 2*9
MSC: 20 = 2*10
```

