The Spin Model Checker : Part II

Review

- Promela
  - The system specification language of the Spin model checker
  - Syntax is similar to that of C, but simplified
    - No float type, no functions, no pointers etc
  - Paradigm is similar to that of CCS
    - Communication and concurrency
    - Clear operational semantics
    - Interleaved semantics
    - Asynchronous process execution
    - Two-way communication
  - Unique features not found in programming languages
    - Non-determinism (process level and statement level)
    - Executability

6 Types of Basic Statements

- 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

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);
}
```

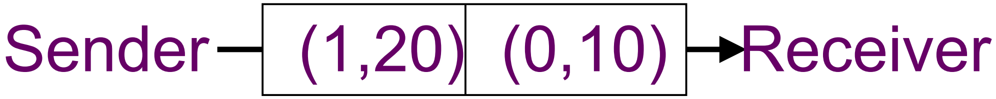
```
[[root@moonzoo deadlocked]# a.out
pan: invalid end state (at depth 3)

(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



Sender (1,20) (0,10) Receiver

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

Operations on Channels

- 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)
    - Sender: ch1!7;ch1!2 vs ch1!!7;ch1!!2
  - `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

Spin's Runtime Options

```
moonzoo@verifier4:~/spin$ spin --help
use: spin [-option] ... [-option] file
        Note: file must always be the last argument
        -A apply slicing algorithm
        -a generate a verifier in pan.c
        -B no final state details in simulations
        -b don't execute printfs in simulation
        -C print channel access info (combine with -g etc.)
        -c columnated -s -r simulation output
        -d produce symbol-table information
        -Dyyy pass -Dyyy to the preprocessor
        -Eyyy pass yyy to the preprocessor
        -e compute synchronous product of multiple never claims
        -f "..formula.."  translate LTL into never claim
        -F file  like -f, but with the LTL formula stored in a 1-line file
        -g print all global variables
        -h at end of run, print value of seed for random nr generator used
        -i interactive (random simulation)
        -I show result of inlining and preprocessing
        -J reverse eval order of nested unlesses
        -jN skip the first N steps in simulation trail
        -k fname use the trailfile stored in file fname, see also -t
        -L when using -e, use strict language intersection
        -l print all local variables
        -M generate msc-flow in tcl/tk format
        -m lose msgs sent to full queues
        -N fname use never claim stored in file fname
        -nN seed for random nr generator
        -O use old scope rules (pre 5.3.0)
-o1 turn off dataflow-optimizations in verifier
-o2 don't hide write-only variables in verifier
-o3 turn off statement merging in verifier
-o4 turn on rendezvous optiomizations in verifier
-o5 turn on case caching (reduces size of pan.m, but affects
reachability reports)
-o6 revert to the old rules for interpreting priority tags
-o7 revert to the old rules for semi-colon usage (pre version 6.3)
-Pxxx use xxx for preprocessing
-p print all statements
-pp pretty-print (reformat) stdin, write stdout
-qN suppress io for queue N in printouts
-r print receive events
-replay  replay an error trail-file found earlier
                if the model contains embedded c-code, the ./pan
executable is used
                otherwise spin itself is used to replay the trailfile
                note that pan recognizes different runtime options than
spin itself
-S1 and -S2 separate pan source for claim and model
        -s print send events
        -T do not indent printf output
        -t[N] follow [Nth] simulation trail, see also -k
        -Uyyy pass -Uyyy to the preprocessor
        -uN stop a simulation run after N steps
        -v verbose, more warnings
        -w very verbose (when combined with -l or -g)
        -[XYZ] reserved for use by xspin interface
        -V print version number and exit
```

-run (or -search) generate a verifier, and compile and run it

options before -search are interpreted by spin to parse the input

options following a -search are used to compile and run the verifier pan

valid options that can follow a -search argument include:

-bfs perform a breadth-first search

-bfspar perform a parallel breadth-first search

-dfspar perform a parallel depth-first search, same as -DNCORE=4

-bcs use the bounded-context-switching algorithm

-bitstate or -bit, use bitstate storage

-biterateN,M use bitstate with iterative search refinement (-w18..-w35)

perform N randomized runs and increment -w every M runs

default value for N is 10, default for M is 1

(use N,N to keep -w fixed for all runs)

(add -w to see which commands will be executed)

(add -W if ./pan exists and need not be recompiled)

-swarmN,M like -biterate, but running all iterations in parallel

-link file.c link executable pan to file.c

-collapse use collapse state compression

-noreduce do not use partial order reduction

-hc use hash-compact storage

-noclaim ignore all ltl and never claims

-p_permute use process scheduling order random permutation

-p_rotateN use process scheduling order rotation by N

-p_reverse use process scheduling order reversal

-rhash randomly pick one of the -p_... options

-ltl p verify the ltl property named p

-safety compile for safety properties only

-i use the dfs iterative shortening algorithm

-a search for acceptance cycles

-l search for non-progress cycles

similarly, a -D... parameter can be specified to modify the compilation

and any valid runtime pan argument can be specified for the verification

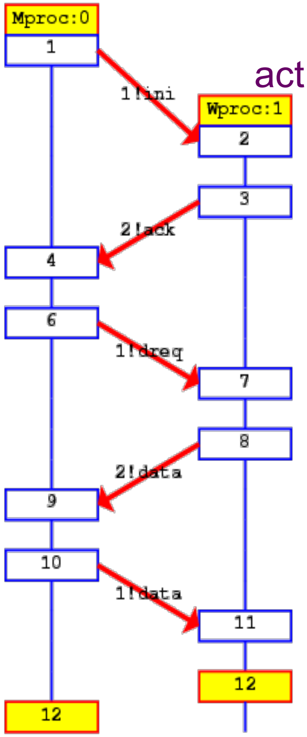
Spin's Simulation Feature

spin -p -n<random seed#> *.pml

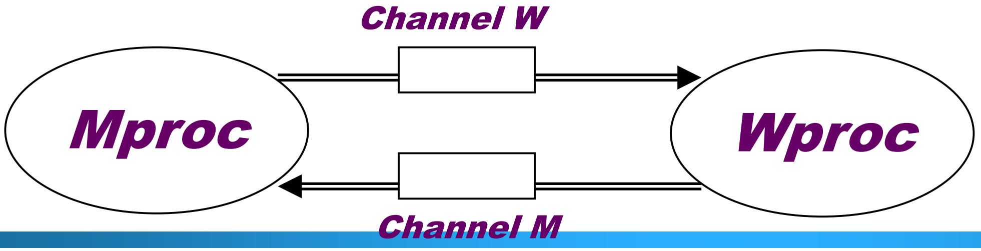
```
moonzoo@verifier4:~/spin$ spin -p -n1 faulty_protocol.pml
  0:    proc  - (:root:) creates proc  0 (Mproc)
  0:    proc  - (:root:) creates proc  1 (Wproc)
  1:    proc  0 (Mproc:1) faulty_protocol.pml:7 (state 1)       [W!ini]
  2:    proc  1 (Wproc:1) faulty_protocol.pml:25 (state 1)      [W?ini]
  3:    proc  1 (Wproc:1) faulty_protocol.pml:26 (state 2)      [M!ack]
  4:    proc  0 (Mproc:1) faulty_protocol.pml:8 (state 2)       [M?ack]
  5:    proc  1 (Wproc:1) faulty_protocol.pml:38 (state 11)     [.(goto)]
      timeout
  6:    proc  0 (Mproc:1) faulty_protocol.pml:10 (state 3)      [(timeout)]
  7:    proc  0 (Mproc:1) faulty_protocol.pml:12 (state 4)      [W!shutup]
  8:    proc  1 (Wproc:1) faulty_protocol.pml:33 (state 7)      [W?shutup]
  9:    proc  0 (Mproc:1) faulty_protocol.pml:19 (state 15)     [.(goto)]
 10:    proc  1 (Wproc:1) faulty_protocol.pml:34 (state 8)      [M!shutup]
 11:    proc  1 (Wproc:1) faulty_protocol.pml:35 (state 9)      [goto :b1]
 12:    proc  0 (Mproc:1) faulty_protocol.pml:19 (state 16)     [M?shutup]
 13:    proc  0 (Mproc:1) faulty_protocol.pml:20 (state 17)     [W!quiet]
 14:    proc  1 (Wproc:1) faulty_protocol.pml:38 (state 13)     [W?quiet]
 15:    proc  1 (Wproc:1) faulty_protocol.pml:39 (state 14)     [M!dead]
 15:    proc  1 (Wproc:1)          terminates
 16:    proc  0 (Mproc:1) faulty_protocol.pml:21 (state 18)     [M?dead]
 16:    proc  0 (Mproc:1)       terminates
2 processes created
moonzoo@verifier4:~/spin$
```

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;
}
```



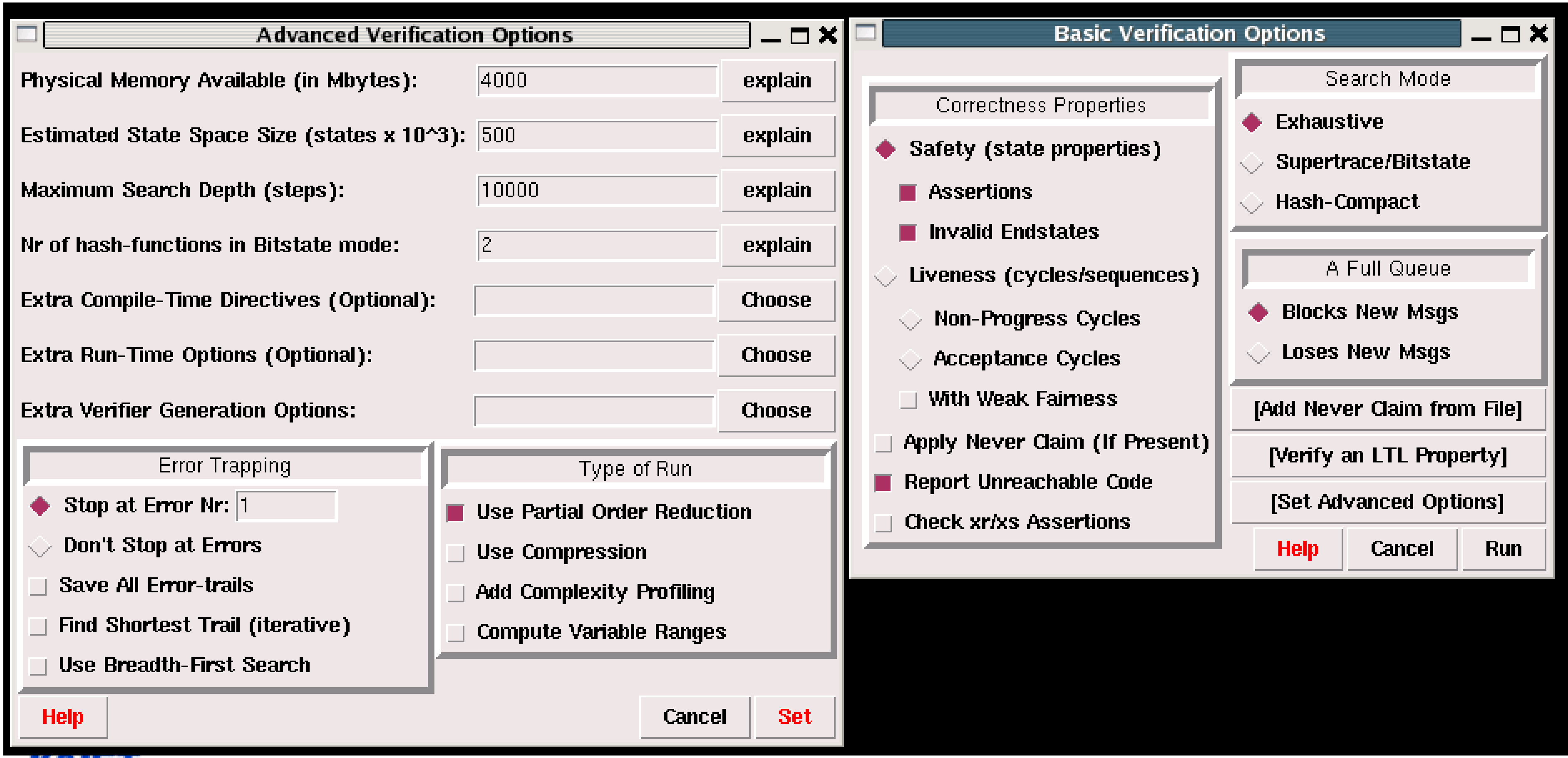
The Sieve of Eratosthenes (pg 326)

```
/*
   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 = %
                :: 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
}
```

Options in XSPIN

- Now you have learned all necessary techniques to verify common problems in the SW development



Advanced Verification Options

Physical Memory Available (in Mbytes): 4000

Estimated State Space Size (states x 10^3): 500

Maximum Search Depth (steps): 10000

Nr of hash-functions in Bitstate mode: 2

Extra Compile-Time Directives (Optional):

Extra Run-Time Options (Optional):

Extra Verifier Generation Options:

Error Trapping

- ◆ Stop at Error Nr: 1
- ◇ Don't Stop at Errors
- □ Save All Error-trails
- □ Find Shortest Trail (iterative)
- □ Use Breadth-First Search

Type of Run

- ☑ Use Partial Order Reduction
- □ Use Compression
- □ Add Complexity Profiling
- □ Compute Variable Ranges

Basic Verification Options

Correctness Properties

- ◆ Safety (state properties)
  - ☑ Assertions
  - ☑ Invalid Endstates
- ◇ Liveness (cycles/sequences)
  - ◇ Non-Progress Cycles
  - ◇ Acceptance Cycles
  - □ With Weak Fairness
- □ Apply Never Claim (If Present)
- ☑ Report Unreachable Code
- □ Check xr/xs Assertions

Search Mode

- ◆ Exhaustive
- ◇ Supertrace/Bitstate
- ◇ Hash-Compact

A Full Queue

- ◆ Blocks New Msgs
- ◇ Loses New Msgs