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);
}
```

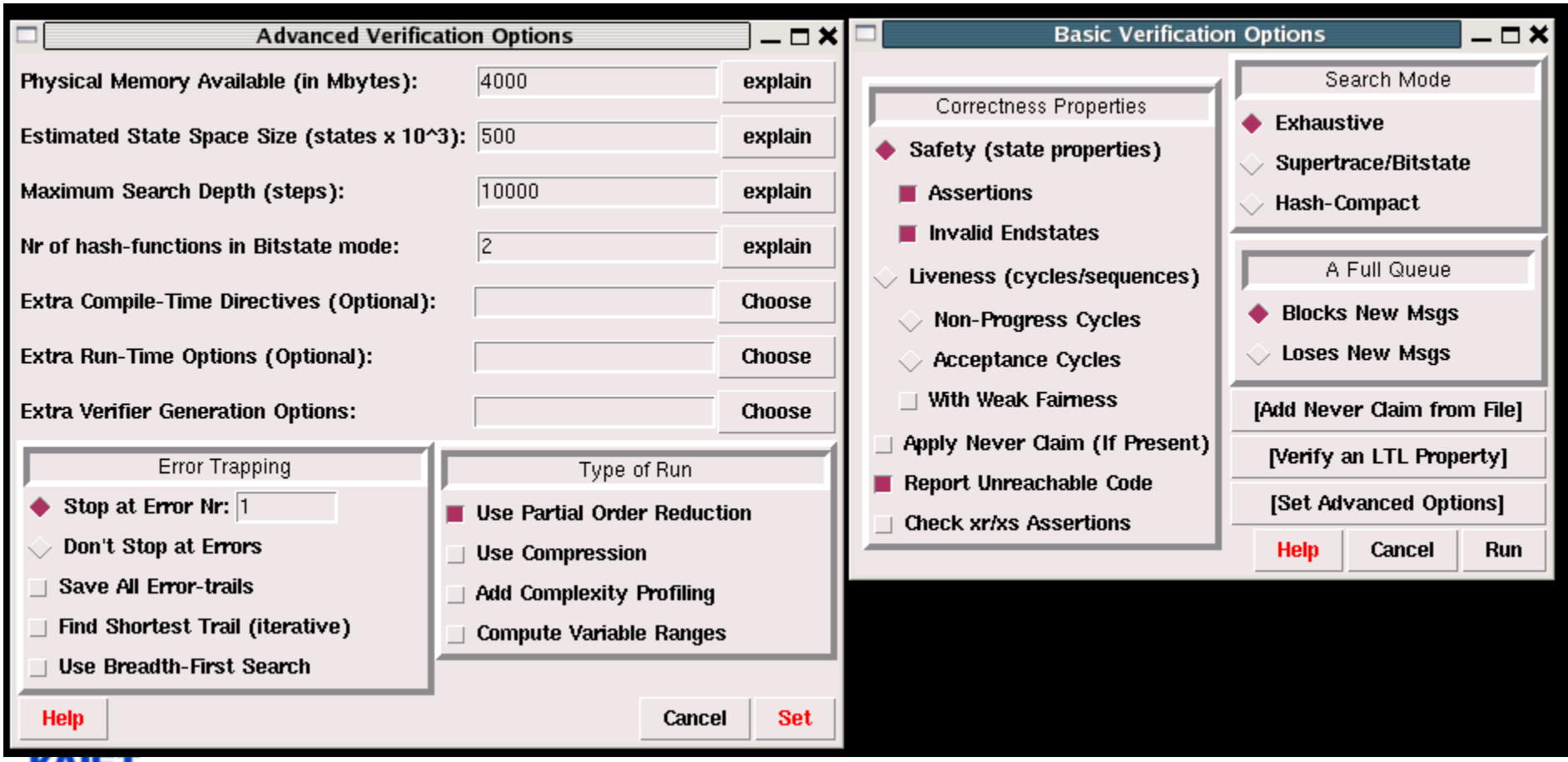
```
[[root@moonzoo deadlocked]# a.out
pan: invalid end state (at depth 3)

(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
```

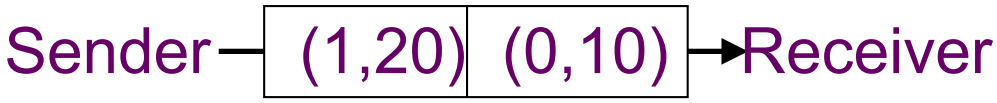
Options in XSPIN

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



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}



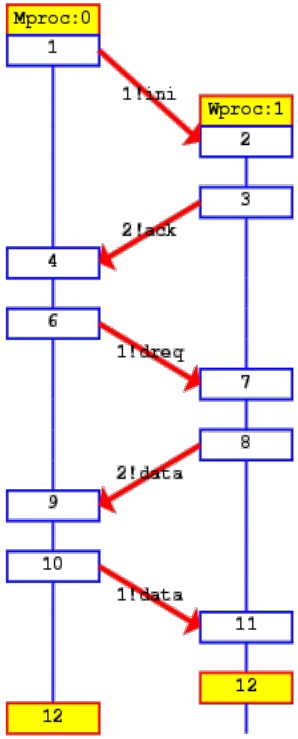
Operations on Channels

- **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)
  - `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

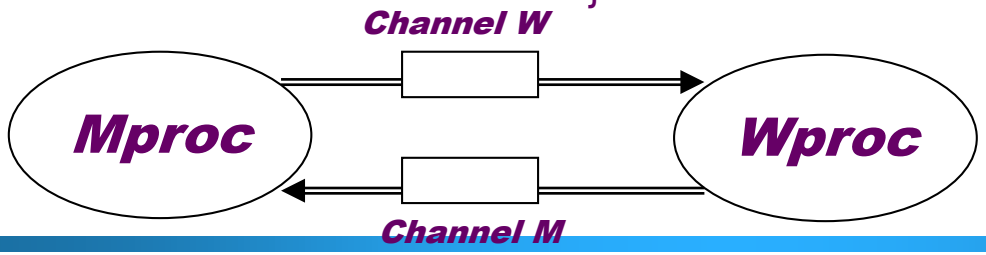
Faulty Data Transfer Protocol

(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;
}
```



KAIST

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
}
```