The Spin Model Checker : Part I/II

Moonzoo Kim
CS Dept. KAIST
Motivation: Tragic Accidents Caused by SW Bugs
June 2002

“Software bugs, or errors, are so prevalent and so detrimental that they cost the U.S. economy an estimated $59.5 billion annually, or about 0.6 percent of the gross domestic product

…

At the national level, over half of the costs are borne by software users and the remainder by software developers/vendors.”

…

The study also found that, although all errors cannot be removed, more than a third of these costs, or an estimated $22.2 billion, could be eliminated by an improved testing infrastructure that enables earlier and more effective identification and removal of software defects.”
Model Checking

- Specify requirement properties and build system model
- Generate possible states from the model and then check exhaustively whether given requirement properties are satisfied within the state space

(model checking)

\[ \square (\Phi \rightarrow \Diamond \Omega) \]
Model Checking (cont.)

- Developed independently by Clarke and Emerson and by Queille and Sifakis in early 1980’s.
- Model checking *complements* testing/simulation.

**Advantages**
- **No proofs!!!**
- **Fast** (compared to other rigorous methods)
- **Diagnostic counterexamples**
- **Logics can easily express many concurrency properties**
Example. Mutual Exclusion Algorithm

```
char cnt=0,x=0,y=0,z=0;
void process() {
    char me = _pid + 1; /* me is 1 or 2*/
    again:
        x = me;
        if (y ==0 || y== me) ;
        else goto again;

        z =me;
        if (x == me) ;
        else goto again;

        y=me;
        if(z==me);
        else goto again;

        /* enter critical section */
        cnt++;/* assert( cnt ==1); */
        cnt --;
        goto again;
    }
```

<table>
<thead>
<tr>
<th>Mutual Exclusion Algorithm</th>
<th>Counter Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process 0</td>
<td>Process 1</td>
</tr>
<tr>
<td>x = 2</td>
<td>y==0</td>
</tr>
<tr>
<td>y = 0</td>
<td></td>
</tr>
<tr>
<td>z = 2</td>
<td></td>
</tr>
<tr>
<td>x = 1</td>
<td></td>
</tr>
<tr>
<td>y == 0</td>
<td></td>
</tr>
<tr>
<td>cnt++</td>
<td></td>
</tr>
<tr>
<td>y = 2</td>
<td></td>
</tr>
<tr>
<td>(z ==2)</td>
<td></td>
</tr>
<tr>
<td>cnt++</td>
<td></td>
</tr>
<tr>
<td>z = 1</td>
<td></td>
</tr>
<tr>
<td>x = 1</td>
<td></td>
</tr>
<tr>
<td>y = 1</td>
<td></td>
</tr>
<tr>
<td>z == 1</td>
<td></td>
</tr>
<tr>
<td>cnt++</td>
<td></td>
</tr>
</tbody>
</table>
Overview of the Spin Architecture

System Spec. In Promela

Spin Model Checker

C compiler

pan.c

C compiler

a.out

Req. Spec. In LTL

Overview of the Spin Architecture

<table>
<thead>
<tr>
<th>Feature</th>
<th>SPIN</th>
<th>NuSMV</th>
</tr>
</thead>
<tbody>
<tr>
<td>Req. Logic</td>
<td>Linear Temporal Logic</td>
<td>CTL + LTL</td>
</tr>
<tr>
<td>Main model of execution</td>
<td>Asynchronous execution (SW)</td>
<td>Synchronous (HW)</td>
</tr>
<tr>
<td>State management</td>
<td>Explicit (hash table)</td>
<td>Symbolic (BDD)</td>
</tr>
<tr>
<td>Main communication model</td>
<td>Explicit buffered channel</td>
<td>Global variable</td>
</tr>
<tr>
<td>Tool maturity</td>
<td>High</td>
<td>Medium</td>
</tr>
</tbody>
</table>

Counter Example(s)

OK
Spin’s modeling language - PROMELA

Promela (process meta-language)

- Syntax is similar to that of C, but simplified
  - No float type, no functions, no pointers etc

Underlying paradigm

- 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
Overview of the Promela

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

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

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

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

Init {
  run B(2);
}
```
Variables and Types

- **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 $\leq 255$
- Each process has only finite length of codes
- Each variable is of finite datatype
- All message channels have bounded capability $\leq 255$
Basic Statements

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 !=0

If an expression statement in blocked, it remains blocked until other process changes the condition
- an expression e is equivalent to while(!e); in C
assert Statement

- **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
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
}
```
Critical Section Example

```c
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
crit.pml  pan.b  pan.c  pan.h  pan.m  pan.t
[root@moonzoo spin_test]# ls

```
[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.)

```c
[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
```

Critical Section Example (cont.)
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
Now you have learned all necessary techniques to verify common problems in the SW development.
Spin home page

http://www.spinroot.com

• Tool downloads and documents (tutorials, online reference, etc)

The Spin Model Checker by G. Holzmann
– 2nd ed, Addison Wesley