

Spin and Formal Verification

- . What is Spin?
- . Why we need model verification
- . Promela and model description
- . What's new in Version 2.0 and 3.0
- . Demonstration

What is Spin?

- . Spin is a tool for analyzing the logical consistency of concurrent systems, specifically of data communication protocols
- . Spin uses a modeling language called Promela (Process Meta Language) to describe the behavior and interaction between different processes, invariants can be represented as assertions.
- . Spin works on the Promela description and checks for absence of deadlocks, unspecified receptions, and unexecutable code.
- . Spin is designed to attack software problem, but is also helpful in logical design, because it provides a method of formal verification.

How spin works?

- . Simulation
Spin can work on the Promela description by performing a random simulation
- . Verification
 - 1) Spin can also generate a C program, which performs an exhaustive search in the whole state space to verify whether the design is error-free.
 - 2) Very large verification problem is attacked by 'bit-state-storage' technique, also known as **supertrace**, which takes a divide-and-conquer like strategy with little side effects.

Why we need formal verification?

- . Guarantee correctness of system design
- . Sometimes, it might be life saver...
 - 1) 1994, a FDIV problem occurred in Intel Pentium processor
 - 2) Intel spent \$475M to cover replacement and inventory writedown cost.
 - 3) SCL (strategic CAD labs) founded afterwards, formal verification is one of its tasks.

Promela and Model Description

- . Promela provides a high level abstraction of protocol or distributed system in general. With the C-like syntax, it describes the behavior and interaction among processes yet suppressing unrelated details.
- . Concepts in Promela
 - 1) Process
Processes are global objects that run concurrently. Its used to describe behavior.
 - 2) Channels and Variables
Channels and variables describe the environment in which processes run.

Promela and Model Description

- . Important aspects of Promela
 - 1) Executability
In Promela, execution of each statement depends on its executability. Thus it provides a basic method of synchronization by providing the concept of critical section
ex1: (a==b) will execute only if it's true.
ex2:

```
proctype A() {
    x = true; t = Bturn;
    (y == false || t == Aturn);
    /* critical section */
    x = false }

proctype B() {
    y = true; t = Aturn;
    (x == false || t == Bturn);
    /* critical section */
    y = false }

init { run A(); run B() }
```

Promela and Model Description

2) Atomic sequence

Atomic sequence provides a monitor-like synchronization mechanism like what critical section does.

```
ex1: byte state = 1;
    proctype A() {
        atomic { (state==1) -> state = state+1 }
    }
    proctype B() {
        atomic { (state==1) -> state = state-1 }
    }
    init { run A(); run B() }
```

Promela and Model Description

3) Message passing

Message channels are used to model the transfer of data from one process to another. They can be declared locally or globally.

```
ex1: chan qname = [16] of { short } // simple data type
ex2: chan qname = [16] of { byte, int, chan, byte } // data structure
ex3: sending qname!exp1, exp2, exp3...
    Receiving qname?var1,var2,var3 ...
```

The sending is executable only if the channel is not full, the receiving is executable only if the channel is non empty.

Promela and Model Description

4) Rendezvous communication

Rendezvous communication is synchronous compared with message passing, by declaring a zero size channel.

```
ex1: #define msgtype 33
    chan name = [0] of { byte, byte }; // rendezvous port of size 0
    proctype A() { name!msgtype(124); // sender
        name!msgtype(121) // unexecutable without receiver
    }
    proctype B() { byte state; name?msgtype(state) // receiver
    }
    init { atomic { run A(); run B() } }
```

Promela and Model Description

5) Data types

```
bit or bool, range(0..1)
byte, short, range(0..266), range(-2^15 - 1 .. 2^15 - 1)
int, range(-2^31 - 1 .. 2^31 - 1)
```

```
Array type
ex1: byte A[N]
```

```
Process type
ex1: proctype A() {
    byte state;
    state = 3
}
```

Promela and Model Description

6) Case selection

```
ex1: if
    :: (a != b) -> option1 // guard (a != b)
    :: (a == b) -> option2 // guard (a==b)
fi
```

Only one case above will be executed according to the guard (first statement)

- a) What will happen if more than one guards are executable, which one will run, option1 or option 2 ? (nondeterministic)
b) How if none of the guards is true? (blocked)

Promela and Model Description

7) Repetition

```
ex1: do
    :: count = count + 1
    :: count = count - 1
    :: (count == 0) -> break
od
ex2: do
    :: (count != 0) -> if
        :: count = count + 1
        :: count = count - 1
        fi
    :: (count == 0) -> break
od
```

In ex1, one branch is chosen to execute repeatedly, and it terminate only if count equal zero. To guarantee that it always terminate if count==1, we then get ex2.

Promela and Model Description

8) Modeling procedure and recursion

procedures and recursions are modeled by processes, they thus introduce a problem of concurrency running.

```
ex1: proctype fact(int n; chan p) {
  chan child = [1] of { int }; int result;
  if
  :: (n <= 1) -> p!1
  :: (n >= 2) -> run fact(n-1, child);
  child?result; // wait for the child to complete
  p!n*result
  fi
}
```

Promela and Model Description

9) Timeout and Assertion

Timeout models a special condition that allows the process a borts waiting when no other statements within the system is executable.

Assertion evaluate whether constraints are satisfied, just like in C/C++

```
ex1: proctype watchdog() {
  do
  :: timeout -> guard!reset // timeout
  od
}
ex2: assert(any_boolean_condition) // assertion
```

Promela and Model Description

10) End state

It's common that some system process may linger in an idle state or loop while the others will reach the end of predefined program. To make spin distinguish normal end state and abnormal ones, Promela introduces End-State-Label. (labels with 3 prefix letters as "end" are end state labels)

```
ex1: proctype dijkstra() {
  byte count = 1;
  end: do // it's normal the process is here at the end.
  :: (count == 1) -> sema!p; count = 0
  :: (count == 0) -> sema?v; count = 1
  od
}
```

Promela and Model Description

11) Progress state

progress states are transient states that other states have to go through one of them to make progress. Otherwise, it may indicate a starvation.

```
ex1: proctype dijkstra() {
  byte count = 1;
  end: do
  :: (count == 1) ->
  progress: sema!p; count = 0 // progress state
  :: (count == 0) -> sema?v; count = 1
  od
}
```

Promela and Model Description

12) LTL (Linear time Temporal Logic) and Promela

Common languages use a logic named propositional logic, LTL extends it with 4 temporal operators:

- 1) Always ([]), Always x > 0
2) Next, Next x > 0 (Whats the operator?)
3) Eventually(<->), Eventually x > 0
4) Until (U), x > 0 until y > 0

Promela doesn't support LTL, but spin can translate it as never claim.

```
ex1: x > 0 until ( y > 0 until z > 0 ) ( suitable for specification !)
```

Spin will express it as p U (q U r) , p q r are the conditions

Promela and Model Description

LTL (continued)

LTL operators used in Spin expression

- []: always, <->: eventually, !: not
U: until, V: (dual of U), &&: and, ||: or, ->: imply, <->: equivalence

```
ex1: spin -f "[ ] p"
ex2: spin -f "p U (q U r)"
```

How to use Spin ?

```

Simulation
$ spin [-p|-g|-l|-r|-s] promelafile
-p shows state changes of processes every time step
-g shows value of global variables every time step
-l shows local variables after process changes state
-r shows message receive events
-s shows message send events

Verification
$ spin -a [-m] promelafile      (generate a C verifier pan.c)
$ cc [-DBITSTATE] -o run pan.c  (compile pan.c)
( -m modified sending behavior, cause it always executable)
$ run      ( generate pan.trail which can be inspected by spin with option -t)
    
```

What's new in V2.0 and V3.0

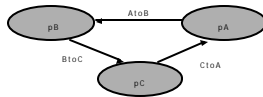
Many aspects of Spin and Promela language have been modified and enhanced concerning functionality, performance, robustness and portability.

- . Partial order reduction
- . Complexity profiling
- . New runtime options
- . Compliance with automata theory
- . Improved Promela language features

Just to name a few.

Example of Spin simulation

. Problem description



- 1) There are two processes and a two-way channel, they communication with each other by sending messages of types (ack, nak, err), err models distortion on channels. Error only occurs on channel from pB to pA, we model this by adding one more process pC.
- 2) They try to overcome the errors by acknowledgement and retransmission.

Example (continued)

. Modeling code

```

1 #define MIN 9
2 #define MAX 12
3 #define FILL 99
4
5 mtype= { ack, nak, err }
6
7 proctype transfer(chan chin, chout)
8 { byte o, i, last_i=MIN;
9
10 o = MIN+1;
11 do
12 :: chin?nak(i) ->
13     assert(i == last_i+1);
14     chout!ack(o)
15 :: chin?ack(i) ->
16     if
17         :: (o < MAX) -> o = o+1
18         :: (o >= MAX) -> o = FILL
19     fi;
20     chout!ack(o)
21 :: chin?err(i) ->
22     chout!nak(o)
23 od
24 }
    
```

Promela Source Code part I

Example (continued)

```

26 proctype channel(chan in, out)
27 { bytemd, mt;
28 do
29 :: in?m_md ->
30     if
31         :: out!m_tand
32         :: out!err,0
33     fi
34 od
35 }
37 init
38 { chan AtoB= [1] of { mtype, byte };
39   chan BtoC= [1] of { mtype, byte };
40   chan CtoA = [1] of { mtype, byte };
41   atomic {
42       run transfer(AtoB, BtoC);
43       run channel(BtoC, CtoA);
44       run transfer(CtoA, AtoB)
45   };
46   AtoBerr,0; /* start */
47   0 /* hang */
48 }
    
```

Promela source code part II