Slide 1 of 1

CS477 Formal Software Development Methods

Elsa L Gunter
2112 SC, UIUC
egunter@illinois.edu
http://courses.engr.illinois.edu/cs477

Slides based in part on previous lectures by Mahesh Vishwanathan, and by Gul Agha

April 19, 2013
bit flag; /* signal entering/leaving the section */
byte mutex; /* # procs in the critical section. */
proctype P(bit i) {
    flag != 1;
    flag = 1;
    mutex++; 
    printf("MSC: P(\%d) has entered section.\n", i);
    mutex--; 
    flag = 0;
}
proctype monitor() {
    assert(mutex != 2);
}
init {
    atomic { run P(0); run P(1); run monitor(); } 
}
SPIN as Simulator

bash-3.2$ spin mutexwrong1.pml
   MSC: P(0) has entered section.
   MSC: P(1) has entered section.
4 processes created
bash-3.2$ !s
spin mutexwrong1.pml
   MSC: P(1) has entered section.
   MSC: P(0) has entered section.
4 processes created
bash-3.2$ spin -a mutexwrong1.pml
bash-3.2$ cc -o pan pan.c
bash-3.2$ ./pan
hint: this search is more efficient if pan.c is compiled
-DSAFETY
pan:1: assertion violated (mutex!=2) (at depth 11)
pan: wrote mutexwrong1.pml.trail

(Spin Version 6.2.4 -- 8 March 2013)
Warning: Search not completed
+ Partial Order Reduction

Full statespace search for:
ever claim - (none specified)
assertion violations +
acceptance cycles - (not selected)
invalid end states +
bit x, y;       /* signal entering/leaving the section */
byte mutex;     /* # of procs in the critical section. */

active proctype A() {
  x = 1;
  y == 0;
  mutex++;
  printf ("Process A is in the critical section\n");
  mutex--;
  x = 0;
}

active proctype B() {
    y = 1;
    x == 0;
    mutex++;
    printf ("Process B is in the critical section\n");
    mutex--;
    y = 0;
}

active proctype monitor() {
    assert(mutex != 2);
}

bash-3.2$ spin mutexwrong2.pml
    Process A is in the critical section
    Process B is in the critical section
3 processes created
bash-3.2$ spin mutexwrong2.pml
    timeout
#processes: 2
x = 1
y = 1
mutex = 0
  3: proc  1 (B) mutexwrong2.pml:15 (state 2)
  3: proc  0 (A) mutexwrong2.pml:6 (state 2)
3 processes created
bash-3.2$ spin -a mutexwrong2.pml
bash-3.2$ cc -o pan pan.c
bash-3.2$ ./pan

hint: this search is more efficient if pan.c is compiled -DSAFETY

pan:1: invalid end state (at depth 3)
pan: wrote mutexwrong2.pml.trail

(Spin Version 6.2.4 -- 8 March 2013)
Warning: Search not completed
+ Partial Order Reduction

Full statespace search for:
never claim - (none specified)
assertion violations +
acceptance cycles - (not selected)
invalid end states +
Examining Error Traces: mutexwrong3.pml

/* File: mutexwrong3.pml */
byte cnt;
byte x, y, z;

active [2] proctype user()
{ byte me = _pid + 1; /* me either 1 or 2 */

again:
    x = me;
    if
        :: (y == 0 || y == me) -> skip
        :: else -> goto again;
    fi;

    z = me;
Examining Error Traces: mutexwrong3.pml

if
:: (x == me) -> skip
:: else -> goto again;
fi;

y = me;
if
:: (z == me) -> skip
:: else -> goto again;
fi;

/* enter the critical section */
cnt = cnt + 1;
assert (cnt == 1);
cnt = cnt - 1;
goto again
}
Generating Error Traces: mutexwrong3.pml

bash-3.2$ spin -a mutexwrong2.pml
bash-3.2$ cc -o pan pan.c
bash-3.2$ ./pan
hint: this search is more efficient if pan.c is compiled
    -DSAFETY
pan:1: invalid end state (at depth 3)
pan: wrote mutexwrong2.pml.trail
Examining Error Traces: `mutexwrong1.pml`

How did `mutexwrong1.pml` go wrong?

```bash
bash-3.2$
spin -p -s -r -v -n123 -l -g -k mutexwrong1.pml.trail -u10000 mutexwrong1.pml
```

Simulator options (incomplete):

- `-p`: Print at each state which process took which step
- `-s`: Print send statements and their effects
- `-r`: Print receive statements and their effects
- `-v`: verbose
- `-nN`: Use $N$ as random seed, instead of clock (good for reproducibility)
- `l` Show changes to local variables
- `g` Show changes to global variables
- `-uN` Limit number of steps taken to $N$
- `-k filename` use the trail file stored in `filename`
Examining Error Traces: mutexwrong1.pml

How did mutexwrong1.pml go wrong?

spin: mutexwrong1.pml:0, warning, proctype P, 'bit i'
variable is never used (other than in print stmts)
using statement merging
Starting P with pid 1
  1: proc 0 (:init:) mutexwrong1.pml:15 (state 1) [(run P(0))]
Starting P with pid 2
  2: proc 0 (:init:) mutexwrong1.pml:15 (state 2) [(run P(1))]
Starting monitor with pid 3
  3: proc 0 (:init:) mutexwrong1.pml:15 (state 3)
[(run monitor())]
  4: proc 2 (P) mutexwrong1.pml:4 (state 1) [((flag!=1))]
  5: proc 1 (P) mutexwrong1.pml:4 (state 1) [((flag!=1))]
  6: proc 2 (P) mutexwrong1.pml:5 (state 2) [flag = 1]
flag = 1
Examining Error Traces: mutexwrong1.pml

7: proc 2 (P) mutexwrong1.pml:6 (state 3)
(mutex = (mutex+1))

mutex = 1

MSC: P(1) has entered section.

8: proc 2 (P) mutexwrong1.pml:7 (state 4)
[printf('MSC: P(%d) has entered section.\n', i)]

9: proc 1 (P) mutexwrong1.pml:5 (state 2) [flag = 1]
10: proc 1 (P) mutexwrong1.pml:6 (state 3)
(mutex = (mutex+1))

mutex = 2

MSC: P(0) has entered section.

11: proc 1 (P) mutexwrong1.pml:7 (state 4)
[printf('MSC: P(%d) has entered section.\n', i)]

spin: mutexwrong1.pml:12, Error: assertion violated

spin: text of failed assertion: assert((mutex!=2))

12: proc 3 (monitor) mutexwrong1.pml:12 (state 1)
[assert((mutex!=2))]
Examining Error Traces: mutexwrong1.pml

spin: trail ends after 12 steps

#processes: 4

flag = 1

mutex = 2

12: proc 3 (monitor) mutexwrong1.pml:13 (state 2) <valid end state>
12: proc 2 (P) mutexwrong1.pml:8 (state 5)
12: proc 1 (P) mutexwrong1.pml:8 (state 5)
12: proc 0 (:init:) mutexwrong1.pml:16 (state 5) <valid end state>

4 processes created
Demo of ispin
never Claims

- **never** claims used to describe systemwide behavior that *should* be impossible
- **monitor** process show similar idea
  - **monitor** checks property is true in some interleaved fashion
  - **never** claim check a property does not happen (anywhere in any execution)
  - **never** claim takes a step after every step of every other process
bit flag; /* signal entering/leaving the section */
byte mutex; /* # procs in the critical section. */

proctype P(bit i) {
    flag != 1;
    flag = 1;
    mutex++;
    printf("MSC: P(%d) has entered section\n", i);
    mutex--;
    flag = 0
}

never{ do
    :: ((mutex != 0)&&(mutex != 1)) -> break
    :: else
    od }

init { atomic { run P(0); run P(1) } }
SPIN Checking never claim

bash-3.2$ spin -p -v -n123 -l -g -k mutexwrong1a.pml.trail mutex

spin: mutexwrong1a.pml:0, warning, proctype P, 'bit i' variable

starting claim 1

using statement merging

1: proc  (never_0) mutexwrong1a.pml:15 (state 3) [else]

Never claim moves to line 15 [else]

Starting P with pid 2

2: proc 0 (:init:) mutexwrong1a.pml:20 (state 1) [(run P(0))]

Starting P with pid 3

3: proc 0 (:init:) mutexwrong1a.pml:20 (state 2) [(run P(1))]

4: proc  (never_0) mutexwrong1a.pml:15 (state 3) [else]

5: proc 2 (P) mutexwrong1a.pml:4 (state 1) [((flag!=1))]

6: proc  (never_0) mutexwrong1a.pml:15 (state 3) [else]

7: proc 1 (P) mutexwrong1a.pml:4 (state 1) [((flag!=1))]

8: proc  (never_0) mutexwrong1a.pml:15 (state 3) [else]
9: proc 2 (P) mutexwrong1a.pml:5 (state 2) [flag = 1]

flag = 1

10: proc - (never_0) mutexwrong1a.pml:15 (state 3) [else]

11: proc 2 (P) mutexwrong1a.pml:6 (state 3)

[mutex = (mutex+1)]

mutex = 1

12: proc - (never_0) mutexwrong1a.pml:15 (state 3) [else]

MSC: P(1) has entered section.

13: proc 2 (P) mutexwrong1a.pml:7 (state 4)

[printf('MSC: P(%d) has entered section.\n',i)]

14: proc - (never_0) mutexwrong1a.pml:15 (state 3) [else]

15: proc 1 (P) mutexwrong1a.pml:5 (state 2) [flag = 1]

16: proc - (never_0) mutexwrong1a.pml:15 (state 3) [else]

17: proc 1 (P) mutexwrong1a.pml:6 (state 3)

[mutex = (mutex+1)]

mutex = 2
18: proc  - (never_0) mutexwrong1a.pml:14 (state 1) 
  [(((mutex!=0)&&(mutex!=1)))]
Never claim moves to line 14 [(((mutex!=0)&&(mutex!=1)))]
spin: trail ends after 19 steps
#processes: 3
flag = 1
mutex = 2
  19: proc  2 (P) mutexwrong1a.pml:8 (state 5)
  19: proc  1 (P) mutexwrong1a.pml:7 (state 4)
  19: proc  0 (:init:) mutexwrong1a.pml:21 (state 4) <valid end state>
  19: proc  - (never_0) mutexwrong1a.pml:17 (state 7) <valid end state>
3 processes created