Common SPIN Workflow

- Write SPIN model; put in file filename
- Debug syntax with: spin -u1000 filename
- Check assertions, bad end states with:
  - spin -a filename
  - gcc -o pan pan.c
  - ./pan
- Read the output
- If you have an error trail: spin -t -p filename
- To see if an LTL formula does not hold:
  - Put LTL formula in file ltlfile
  - spin -F ltlfile > neverclaimfile
  - spin -a -N neverclaimfile filename
  - gcc -o pan pan.c
  - ./pan
- Read the output
- If you have an error trail: spin -t -p filename

Never Claims: mutextwrong1a.pml

```c
bit flag; /* signal entering/leaving the section */
byte mutex; /* # procs in the critical section. */

proctype P(bit i) {
    byte mutex; /* # procs in the critical section. */
    bit flag; /* signal entering/leaving the section */

    flag = 0;
    mutex--;  // entering
    printf("MSC: P(%d) has entered section\n", i);
    mutex++;
    flag = 1;
    mutex--;  // leaving
}

ever( do
    ((mutex != 0)&&(mutex != 1)) -> break
    :: else
    od )
init { atomic { run P(0); run P(1) } }
```

SPIN Checking never claim

```bash
bash-3.2$ spin -p -v -n123 -l -g -k mutexwrong1a.pml.trail mutexwrong1a.pml
spin: mutexwrong1a.pml:0, warning, proctype P, 'bit i' variable is never used (other than in print stmnts)
```

SPIN Commandline Options

The following are some useful commandline options:

- `-a`: Generate code for project-specific verifier
- `-d`: Default: run SPIN as a simulator
- `-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)
- `-t`: Run simulation driven by an error trail
- `-k filename` use the trail file stored in filename
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]
13: proc 2 (P) mutexwrong1a.pml:7 (state 4)
[printf('MSC: P(%d) has entered section.
',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 7) <valid end state>
3 processes created

Traffic Light Example

mtype {NS, EW, Red, Yellow, Green}

bit Turn = 0;
mtype Color[2];

proctype Light(bit myId) {
    mtype otherId = 1 - myId;
do:: Turn == myId && Color[myId] == Red
            -> Color[myId] = Green
        :: Color[myId] == Green -> Color[myId] = Yellow
        :: Color[myId] == Yellow
            -> Color[myId] = Red; Turn = otherId
    od
}

bash-3.2$ cat trafficlightnever.pml
never { /* []((Color[0] = Red) || (Color[1] = Red)) */
    accept_init:
    TO_init:
do:: (((Color[0] = Red) || (Color[1] = Red)))
            -> goto TO_init
    od;
}

Traffic Light Example

// File: trafficlight.pml */

mtype = {NS, EW, Red, Yellow, Green};
bit Turn = 0;
mtype Color[2];

proctype Light(bit myId) {
    bit otherId = 1 - myId;
do:: Turn == myId && Color[myId] == Red
            -> Color[myId] = Green
        :: Color[myId] == Green
            -> Color[myId] = Yellow
        :: Color[myId] == Yellow
            -> Color[myId] = Red; Turn = otherId
    od
}
init { atomic(Color[0] = Red; Color[1] = Red); atomic(run Light(0); run Light(1))
} /* End of File: trafficlight.pml */

Can test this with
bash-3.2$ spin -f '<>()^{}((p) -> ((q) || (r)))' >& trafficlightnever.pml
bash-3.2$ cat trafficlightnever.pml
never

To use file containing never claim:
bash-3.2$ spin -a -N trafficlightnever.pml ...
bash-3.2$ cat trafficlightnever.pml
never /* <>()^{}((p) -> ((q) || (r))) */

Properties (1)

- Model checking tools automatically verify whether $M \models \phi$
  holds, where $M$ is a (finite-state) model of a system and $\phi$ is stated in some formal notation.

- With SPIN one may check the following type of properties:
  - deadlocks (invalid endstates)
  - assertions
  - unreachable code
  - LTL formulae
  - liveness properties
    - non-progress cycles (livelocks)
    - acceptance cycles

LTL to Never Claim

bash-3.2$ spin -f '<>()^{}((Color[0] == Red || Color[1] == Red))' >& trafficlightnever.pml
bash-3.2$ cat trafficlightnever.pml
never /* <>()^{}((Color[0] == Red || Color[1] == Red)) */

Properties (2)

- safety property
  - "nothing bad ever happens"

- liveness property
  - "something good will eventually happen"

  - invariant
    - $x$ is always less than 5
  - deadlock freedom
    - the system never reaches a state where no actions are possible
  - SPIN: find a trace leading to the "bad" thing. If there is not such a trace, the property is satisfied.

Historical Classification

- response
  - if action $X$ occurs then eventually action $Y$ will occur

- SPIN: find a (infinite) loop in which the "good" thing does not happen. If there is not such a loop, the property is satisfied.
Properties (3)

- LTL formulae are used to specify liveness properties.
  \[ \text{LTL = propositional logic + temporal operators} \]
  - \[ []P \] always P
  - \[ <>P \] eventually P
  - \[ P \lor Q \] P is true until Q becomes true

- Some LTL patterns
  - invariance \[ [](p) \]
  - response \[ [](p) \rightarrow (<>(q)) \]
  - precedence \[ [](p) \rightarrow ((q) \lor (x)) \]
  - objective \[ [](p) \rightarrow ((q) \lor ((c)) \]

Xspin contains a special "LTL Manager" to edit, save and load LTL properties.

Properties (4)

- Suggested further reading (on temporal properties):
  - [Bérard et. al. 2001]
    - Textbook on model checking.
    - One part of the book (six chapters) is devoted to "Specifying with Temporal Logic".
    - Also available in French.

- [Dwyer et. al. 1999]
  - classification of temporal logic properties
  - pattern-based approach to the presentation, codification and reuse of property specifications for finite-state verification.

Note: although this tutorial focuses on how to construct an effective Promela model M, the definition of the set of properties which are to be verified is equally important!

Invariance

- \[ []P \] where P is a state property
  - safety property
  - invariance = global universality or global absence
    - Dwyer et. al. 1999:
      - 25% of the properties that are being checked with model checkers are invariance properties
      - BTW, 48% of the properties are response properties
  - examples:
    - [ ] l=10
    - [ ] mutex != 2
  - SPIN supports (at least) 7 ways to check for invariance.

variant 1+2 - monitor process (single assert)

- proposed in SPIN’s documentation
- add the following monitor process to the Promela model:

  ```promela
  active proctype monitor()
  {
  assert(P);
  }
  ```

- Two variations:
  - 1. monitor process is created first
  - 2. monitor process is created last

variant 3 - guarded monitor process

-.Drawback of solution “1+2 monitor process” is that the assert statement is enabled in every state.

  ```promela
  active proctype monitor()
  {
  assert(P);
  }
  ```

- The atomic statement only becomes executable when P itself is not true.

variant 4 - monitor process (do assert)

- From an operational viewpoint, the following monitor process seems less effective:

  ```promela
  active proctype monitor()
  {
  do assert(P);
  }
  ```

- But the number of states is clearly advantageous.
variant 5 - never claim (do assert)

- also proposed in SPIN's documentation

never { do :: assert(P) od }

... but SPIN will issue the following unnerving warning: warning: for p.o. reduction to be valid the never claim must be stutter-closed (never claims generated from LTL formulae are stutter-closed) and this never claim has not been generated.

variant 6 - LTL property

- The logical way...
- SPIN translates the LTL formula to an accepting never claim.

never ( ![]P )
TO_init:
if
:: (!P) -> goto accept_all
:: (1)  -> goto TO_init
fi;
accept_all:
skip
}

variant 7 - unless (!P -> ...)

- Enclose the body of (at least) one of the processes into the following unless clause:

  ( body ) unless ( atomic ( !P -> assert(P) ; ) )

- Discussion
  - no extra process is needed: saves 4 bytes in state vector
  - local variables can be used in the property P
  - definition of the process has to be changed
    - the unless construct can reach inside atomic clauses
    - partial order reduction may be invalid if rendez-vous communication is used within body
    - the body is not allowed to end

Note: disabling partial reduction (-DNOREDUCE) may have severe negative consequences on the effectiveness of the verification run.