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 30, 2014

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

Traffic Light Example

/* File: trafficlight.pml */
mtype = {NS, EW, Red, Yellow, Green};
bit Turn = 0;
mtype

init { atomic{Color[0] = Red; Color[1] = Red};
  atomic{run Light(0); run Light(1)}
}
/* End of File: trafficlight.pml */

LTL to Never Claim

bash-3.2$ spin -f '<>(!(Color[0] == Red || Color[1] == Red))' >& trafficlightnever.pml

spin -a filename

spin -a -N neverclaimfile

spin -F ltlfile neverclaimfile filename

spin -a -N neverclaimfile filename

spin -a -N neverclaimfile filename

spin -a -N neverclaimfile filename
Using never Claim in Separate File

To use file containing never claim:

bash-3.2$ spin -a -N trafficlightnever.pml trafficlight.pml
bash-3.2$ gcc -o pan pan.c
bash-3.2$ ./pan omissions

Full statespace search for:
never claim  + (never_0)
assertion violations  + (if within scope of claim)
acceptance cycles  - (not selected)
invalid end states  - (disabled by never claim)

State-vector 44 byte, depth reached 26, errors: 0
13 states, stored
1 states, matched
14 transitions (= stored*matched)
1 atomic steps
bash conflicts: 0 (resolved)

Properties (1)

- Model checking tools automatically verify whether \( M \models \phi \) holds, where \( M \) is a (finite-state) model of a system and property \( \phi \) 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

Properties (2)

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

- Some LTL patterns
  - invariance \([I (p) \]
  - response \([I ((p) \rightarrow (<> (q))) \]
  - precedence \([I ((p) \rightarrow ((q) U (x))) \]
  - objective \([I ((p) \rightarrow (!(q) \land (x))) \]

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

Properties (3)

Properties (4)

- Suggested further reading (on temporal properties):
  - [Bérand 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

• \([\neg P] \) where \(P\) is a state property
  – safety property
  – invariance = global universality or global absence

• \([\neg P]\) where \(P\) is a state 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

• SPIN supports (at least) 7 ways to check for invariance.

variant 1+2 - monitor process (single \texttt{assert})

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

\[
\text{active proctype monitor}() \\
\{ \\
\text{assert}(P) ; \\
\text{end} \\
\}\]

• 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
  \texttt{assert} statement is enabled in every state.

\[
\text{active proctype monitor}() \\
\{ \\
\text{assert}(P) ; \\
\text{end} \\
\}\]

• The \texttt{atomic} statement only becomes executable when \(P\)
  itself is not true.

variant 4 - monitor process (\texttt{do assert})

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

\[
\text{active proctype monitor}() \\
\{ \\
\text{do} :: \text{assert}(P) ; \\
\text{end} \\
\}\]

• But the number of states is clearly advantageous.

variant 5 - never claim (\texttt{do assert})

• also proposed in SPIN’s documentation

\[
\text{never} ( \{ \text{do} :: \text{assert}(P) \} )
\]

... but SPIN will issue the following unnnerving \texttt{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 \texttt{logical} way...
• SPIN translates the LTL formula to an accepting
  never claim.

\[
\text{never} ( [\neg P] ) \\
\text{TO_init:} \\
\{ \\
\text{if} :: (\neg P) \text{ goto accept_all} \\
:: (1) \text{ goto TO_init} \\
\text{fi; accept_all:} \\
\text{skip} \\
\}\]
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.