SWEN-220
Mathematical Models of Software

Verification in SPIN
Topics

- Deterministic vs Non-Deterministic
- Random Simulation
- SPIN Verifier
Verification in SPIN

• Just like in Alloy, in SPIN assertions are used to verify the correctness of a model.

• **Assertions** are placed between program statements and are evaluated by the model checker for counterexamples.
Example

byte x = 7;

active proctype increment() {
    int oldx = x;

    printf("Hello, world\n");
    printf("x = %d\n", x);
    x++;
    printf("x = %d\n", x);
    assert(x >= oldx);
}

Verification

• Running a random simulation may point out the error

• What if you have many conditions?

• This is where the verifier comes into play
Verifier in SPIN

• In a **deterministic** program, there is only one computation. Therefore, a single simulation will suffice

• In a **nondeterministic** or concurrent program, we need to check all possible computations
Verifier in SPIN

• Verifier systematically checks the correctness among all possible computations

• How do we get all possible computations?
active proctype P()
{
    int a = 5, b=5;
    int max=0;

    if
      :: a >= b -> max = a;
      :: b >= a -> max = b;
    fi;
}

**State:** value of variables and location counters

**Computation:** sequence of states

**State space:** set of all states that can possibly occur
Verifier in SPIN

• As the program complexity increases, the number of computations will increase
• SPIN achieves efficiency by generating an optimized program called a verifier for a model
• If there is an error, a trail file is created. SPIN stops at the first violated assertion
• In jSPIN select the “Verify” button.
SPIN Verify

SPIN requires a C compiler to perform the Verify.
SPIN Verify - C Compiler Check

1. Make sure you have a C compiler installed, you can get one at http://www.mingw.org/
2. Make sure your environment variable PATH includes the location of the gcc compiler, e.g. \mingw\bin

Check that you can compile and run a minimal C program successfully, e.g. hello.c

    #include <stdio.h>
    int main(void) { printf("hello\n"); }

From the command line you should be able to compile & run:

> gcc -o hello hello.c
> .\hello
State Transition Diagram generated by the SPIN Verifier

Each possible state is “explored” by the Verifier.

Where does deadlock occur?

Listing 4.3. Abbreviated solution for the critical section problem

```c
bool wantP = false, wantQ = false;
active proctype P() {
    do :: wantP = true;
    !wantQ;
    wantP = false
    od
}
active proctype Q() {
    do :: wantQ = true;
    !wantP;
    wantQ = false
    od
}
```

Steps of SPIN Verification

Promela Program \(\xrightarrow{\text{Generation}}\) Verifier (C) \(\xrightarrow{\text{Compilation}}\) Compiled version of verifier

Generated from Promela source

Execution

Trail \(\xrightarrow{\text{Some computations contain errors}}\)

Report \(\xrightarrow{\text{All computations are correct}}\)

"Verify"
Verifier Report

\textit{jSPIN \rightarrow select Verify button:}

\begin{verbatim}
bin\spin.exe -a critical-verifier.pml ... done!
c:\cygwin\bin\gcc-4.exe -DSAFETY -o pan pan.c ... done!
Z:\Transfer\Courses\Course Content\SWEN-220\SPIN\pan -m2000 -X ... done!
\end{verbatim}

\textbf{pan:1: invalid end state (at depth 4)}

pan: wrote critical-verifier.pml.trail
(Spin Version 6.1.0 -- 4 May 2011)
Warning: Search not completed
+ Partial Order Reduction

Full statespace search for:
- never claim - (none specified)
+ assertion violations
- cycle checks - (disabled by -DSAFETY)
+ invalid end states

\textbf{State-vector 16 byte, depth reached 5, \textbullet\textbullet\textbullet errors: 1 \textbullet\textbullet\textbullet}

- 6 states, stored
- 1 states, matched
- 7 transitions (= stored+matched)
- 0 atomic steps

hash conflicts: 0 (resolved)

2.195 memory usage (Mbyte)

pan: elapsed time 0.007 seconds
Guided Simulation – Trail Files

• Helps you locate *where* the error occurred
• Uses the Trail file to reconstruct computations
• Note that trail files are only created if an error was detected.
Trail File (Guided)

\textit{jSPIN} -> select \textit{Guided} button:

\begin{verbatim}
1 Q 13 wantQ = 1
Process Statement wantQ
1 Q 14 !(wantP) 1
0 P 5 wantP = 1 1
Process Statement wantP wantQ
1 Q 15 wantQ = 0 1 1
1 Q 13 wantQ = 1 1 0
\end{verbatim}

\textbf{spin: trail ends after 5 steps}

\#processes: 2

5: proc 1 (Q) critical-verifier.pml:14 (state 2)
5: proc 0 (P) critical-verifier.pml:6 (state 2)

2 processes created

Exit-Status 0