Using Model Checking To Generate Tests From Requirements Specifications

Angelo Gargantini and Constance Heitmeyer

Presented by : Susmita Ghose

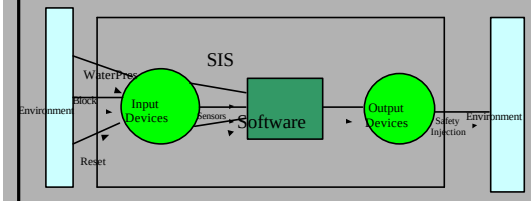
SCR Requirements Method

- Formulated in 1978 to specify the requirements of a Flight Safety Program of the US Navy
- SCR Toolset : Consistency Checker, Dependency Graph Browser, Model Checker…
- System represented as a state machine

SCR Requirements Method

- Monitored Variables
- Controlled Variables
- Input Event
- Output Event
- Auxiliary Variables: Modes (from Mode class) , Terms
- Constants

Safety Injection System



SIS

- Monitored variables :{WaterPres, Block,Reset}
- Controlled variable : {Safety Injection}
- Mode Class defined on WaterPres: {Pressure}
- Modes : { Too Low,Permitted, High}
- Term: {Overridden}
- Constants {Low=10, Permit=20}

SCR Requirements Method

- System is represented as a 4-tuple (S,S_o,E^m,T)
  - S : is the set of states
  - S_o : is the initial set of states
  - E^m: is the set of input events
  - T : is the transformation describing the allowed state transition

SCR Requirements Method

- Event Tables
- Condition Tables

- Event is a predicate defined on a pair of system states implying that the value of at least one variable is changed
- Condition is a predicate defined on a system state

Generating Test Sequences from an Operational Specification

- Derivation of functions from the condition / event table
- Translation of these into the language of the model checker
- Construction of test sequences using the model checker's ability to generate counter-examples

Event Table Defining the Mode Class "Pressure"

Old Mode	Events	New Mode
Too Low	@T(WaterPres>= Low)	Permitted
Permitted	@T(WaterPres>= Permit) @T(WaterPres< Low)	High TooLow
High	@T(WaterPres< Permit)	Permitted

Function Defining 'Pressure'

```
if
  Pressure = TooLow
   if
   @T(WaterPres>= Low) -> Pressure' = Permitted
   (else) -> Pressure' = Pressure     c2
   fi
Pressure = Permitted
   if
   @T(WaterPres >= Permit) -> Pressure' = High     c3
   @T(WaterPres < Low) -> Pressure' = TooLow      c4
   (else) -> Pressure' = Pressure     c5
   fi
Pressure = High
   if
   @T(WaterPres < Permit) -> Pressure' = Permitted     c6
   (else) -> Pressure' = Pressure     c7
   fi
fi
```

Promela Code for Cases *c1* and *c2*

```
if
  : : (Pressure == TooLow) ->
     if
     : : (WaterPresP >= Low) && ! WaterPres >= Low
          -> PressureP = Permitted;   CasePressure = 1;
     :: else                          CasePressure = 2;
     fi
. . .
fi
```

Test Sequence Derived from SPIN Counterexample for *c1*

- Trap Property:
  assert (CasePressure != 1)
- A test sequence of length 20 is generated

- The sequence concludes with two states (s,s') such that, in state s, WaterPres !>= Low and Pressure is TooLow (implied by WaterPres = 9 at step 19) and, in state s', WaterPres>Low and Pressure is Permitted (implied by WaterPres equals 10 at step 20).

Test Case from Condition Table

Mode	Condition	Condition
Too Low	Overridden	NOT Overridden
Permitted, High	True	False
Safety Injection	Off	On

```
if
 Pressure = TooLow
  if
   Overridden =true -> SafetyInjection = Off      c1
   Overridden = false -> SafetyInjection = On     c2
  fi
 Pressure = Permitted -> SafetyInjection = Off    c3
 Pressure = High -> SafetyInjection = Off         c4
fi
```

Branch Coverage

- In each condition table, every condition not equivalent to false is tested at least once
- In each event table, every event is tested at least once.
- In each event table, in each mode, a change in each monitored variable which does not change the value of the variable that the table defines is tested at least once.

A Tool for Automatically Generating Test Sequences

- Tool in java
  - Automatically translates SCR into the language of the model checker (SMV or SPIN)
  - Constructs the different cases
  - Executes the model checker on each case
  - Derives the test sequences
  - Write each test sequence to a file
- Applied to four specifications

Conclusion

- Issues to be addressed
  - State explosion problem
  - Alternate methods for selecting test sequence for a given branch
  - Use the suite of test sequences to test a real software implementation