Using Model Checking to Generate Tests from Requirement Specifications

Authors: A. Gargantini & C. Heitmeyer

Presented By
Dishant Langayan

Université d’Ottawa | University of Ottawa
Overview

• Automate construction of test sequences from a SCR requirements specification
• Model checking to produce counterexamples
• Published in 1999
  – Proceedings of the 7th European software engineering conference
Introduction

• Software Cost Reduction (SCR)
  – Improving software quality by detecting errors in requirements specification
  – Automated consistency checker
  – Simulator
  – Model Checker
• Automation of test sequences from requirements specification
Approach (Overview)

- Use requirements specification for
  - Generating a valid sequence of inputs
  - Oracle that determines the set of output associated with each input
- Input sequence are constrained to satisfy the input model
- Organize input sequences into equivalence classes
- Generate one or more test sequences for each equivalence class
Background - SCR Requirements Method

- Formulated in 1978 for the Operational Flight Program (OFP)
- Describes
  - System environment (non-deterministic)
  - Required system behaviour (deterministic)
- In SCR model:
  - Environmental quantities: monitored and controlled variables
  - A system: 4-tuple \((S, S_0, E^m, T)\)
    - \(S\) is a set of states
    - \(S_0 \subseteq S\) is a the initial state set
    - \(E^m\) is the set of input events
    - \(T\) is the transform describing allowed state transitions
Background - SCR Requirements Method

- $T$ is constructed from two kinds of tables in SCR requirements specifications:
  - Event tables
  - Condition tables
- Example expression:

$$\text{@T(c) WHEN } d \triangleq \neg c \land c' \text{ WHEN } d,$$
Attributes of an Effective Suite of Tests Sequences

• Pushbutton (as automatic as possible)
• Focus on black-box conformance testing
  – Generate test sequences
  – Compare outputs with oracle
• To produce an effective suite of test sequences
  – The number of test sequences in the suite should be small
  – The test suite should cover all errors that any implementation may contain
Generating Test Sequence

• Model checking used for generating test sequences
  – Used as an oracle to compute expected outputs
  – Use model checker’s ability to generate counterexamples to construct test sequences
• Trap properties
An Example System

• Safety Injection System (SIS)
  – Monitors water pressure and injects coolant into the reactor core when the pressure falls below some threshold
  – System can be overridden and reset
  – Some example requirements:
    • Water pressure cannot change more than 3 psi from one state to the next
    • Threshold is 10, determine when the water pressure is in an unsafe region
  – Example Property:

\[ \text{@T(WaterPres < Low) WHEN Block = On \land Reset = Off } \Rightarrow \text{SafetyInjection'} = \text{Off}. \]
Generating Trap Properties (SMV)

- SMV: a symbolic model checker
Generating Trap Properties (SMV)...

P: \( \neg \neg \neg \neg (\text{WaterPres} < \text{Low}) \) WHEN Block = On \land Reset = Off \Rightarrow \text{SafetyInjection'} = \text{Off}. \)
Generating Trap Properties (SMV)...  

\[ P: \ @ T(WaterPres < Low) \text{ WHEN } Block = On \land Reset = Off \Rightarrow SafetyInjection' = Off. \]

Normal Approach

1. Operational Specification
2. SMV Language
3. CTL

- translate \( P \)
- translate \( P' \)
Generating Trap Properties (SMV)...

P: \(@T\text{(WaterPres < Low)}\) \text{WHEN Block = On} \land \text{Reset = Off} \Rightarrow \text{SafetyInjection'} = \text{Off.}

Normal Approach

- Operational Specification
-\text{translate P}

SMV Language
-\text{translate P‘}

CTL

Author’s Approach

- Operational Specification
-\text{translate P}

SMV Language
-\text{translate negation of P‘}

CTL
Generating Trap Properties (SMV)...

P: $\Diamond T(\text{WaterPres} < \text{Low})$ WHEN $\text{Block} = \text{On} \land \text{Reset} = \text{Off} \Rightarrow \text{SafetyInjection}' = \text{Off}$.

\[\downarrow\]

$\text{AG!} (\ EX(\text{WaterPres}<\text{Low}) \ & \ !\ \text{WaterPres}<\text{Low} \ & \ \text{Block} = \text{On} \ & \ \text{Reset} = \text{Off} \ ),$

where

$\text{AG!} = \text{“never”}$

$\text{EX} = \text{“next”}$

$! = \text{negation}$
Test Sequence (SMV)

Table 1: Test Sequence Constructed from SMV Counterexample.

<table>
<thead>
<tr>
<th>Step No.</th>
<th>Monitored Var. Value</th>
<th>Controlled Var. Value</th>
<th>Mode Class Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>WaterPres=2 Block=Off Reset=On</td>
<td>SafetyInjection=On</td>
<td>Pressure=TooLow</td>
</tr>
<tr>
<td>1</td>
<td>Reset=Off</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>WaterPres=5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>WaterPres=8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>WaterPres=10</td>
<td>SafetyInjection=Off</td>
<td>Pressure=Permitted</td>
</tr>
<tr>
<td>5</td>
<td>Block=On</td>
<td></td>
<td>Pressure=TooLow</td>
</tr>
<tr>
<td>6</td>
<td>WaterPres=8</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\(<(r,\text{off}; -), (w,5; -), (w,8; -), (w, 10; s,\text{off}), (b,\text{on}; -), (w,8; -) >\)
Pros and Cons (SMV)

• Pros
  – Can test many critical aspects of the system behaviour

• Cons
  – Formulation of system properties by customers
  – Incompleteness of the test sequence
  – Assumes correctness of both operational specification and the properties
Generating Trap Properties (Spin)

- Does not depend of system properties
- Automatically translates operational requirements specification to language of model checker
- Automatically and systematically generates test sequences
  - Event table
  - Condition table
Generating Trap Properties (Spin)...

Table 2. Event Table Defining the Mode Class Pressure.

<table>
<thead>
<tr>
<th>Old Mode</th>
<th>Event</th>
<th>New Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>TooLow</td>
<td>$@T(Water\text{Pres} \geq \text{Low})$</td>
<td>Permitted</td>
</tr>
<tr>
<td>Permitted</td>
<td>$@T(Water\text{Pres} \geq \text{Permit})$</td>
<td>High</td>
</tr>
<tr>
<td></td>
<td>$@T(Water\text{Pres} &lt; \text{Low})$</td>
<td>TooLow</td>
</tr>
<tr>
<td>High</td>
<td>$@T(Water\text{Pres} &lt; \text{Permit})$</td>
<td>Permitted</td>
</tr>
</tbody>
</table>
Generating Trap Properties (Spin)...

Listing 1. Function Defining Pressure With a Single else Clause.

```plaintext
if
  □ Pressure = TooLow
    ∧ @T(WaterPres ≥ Low)  →  Pressure' = Permitted
  □ Pressure = Permitted
    ∧ @T(WaterPres ≥ Permit)  →  Pressure' = High
  □ Pressure = Permitted
    ∧ @T(WaterPres < Low)  →  Pressure' = TooLow
  □ Pressure = High
    ∧ @T(WaterPres < Permit)  →  Pressure' = Permitted
  □ (else)
    →  Pressure' = Pressure
fi
```
Generating Trap Properties (Spin)...

Listing 2. Function Defining Pressure With One else Clause per Mode.

```plaintext
if
  □ Pressure = TooLow
    if
      □ @T(WaterPres ≥ Low) -> Pressure' = Permitted C1
      □ (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
```
Generating Trap Properties (Spin)...


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

Listing 4. Promela assert statement (trap property).

```assert (CasePressure != 1).```
## Test Sequence (Spin)

Table 3. Test Sequence Derived from Spin Counterexample for Case C1 of Listing 2.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Block On</td>
<td></td>
<td>11</td>
<td>WaterPres 4</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Reset Off</td>
<td></td>
<td>12</td>
<td>Block On</td>
<td>SafetyInjection Off</td>
</tr>
<tr>
<td>3</td>
<td>Block Off</td>
<td></td>
<td>13</td>
<td>Block Off</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Block On</td>
<td>SafetyInjection Off</td>
<td>14</td>
<td>Reset On</td>
<td>SafetyInjection On</td>
</tr>
<tr>
<td>5</td>
<td>Block Off</td>
<td></td>
<td>15</td>
<td>Block On</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>WaterPres 3</td>
<td></td>
<td>16</td>
<td>Reset Off</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Block On</td>
<td></td>
<td>17</td>
<td>WaterPres 5</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Reset On</td>
<td>SafetyInjection On</td>
<td>18</td>
<td>WaterPres 6</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Block Off</td>
<td></td>
<td>19</td>
<td>WaterPres 9</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Reset Off</td>
<td></td>
<td>20</td>
<td>WaterPres 10</td>
<td>SafetyInjection Off</td>
</tr>
</tbody>
</table>
A Tool for Automation

- Developed in Java
- Works with SMV and Spin
- Outputs each test sequence to a file
- Can discard test sequences if current case covers previous test sequences
- Optionally extra test sequence can be constructed at data boundaries
## Experimental Results

Table 5. Automatic Generation of Test Sequences Using Spin and SMV.

<table>
<thead>
<tr>
<th>Specif.</th>
<th>No. of Vars</th>
<th>No. of Branches</th>
<th>Total Test Seq.</th>
<th>Useful Test Seq.</th>
<th>Exec. Time</th>
<th>Total Steps</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Spin</td>
<td>SMV</td>
<td>Spin</td>
<td>SMV</td>
</tr>
<tr>
<td>Small SIS</td>
<td>6</td>
<td>33</td>
<td>7</td>
<td>14</td>
<td>5</td>
<td>11</td>
</tr>
<tr>
<td>Large SIS</td>
<td>6</td>
<td>33</td>
<td>12</td>
<td>14</td>
<td>3</td>
<td>11</td>
</tr>
<tr>
<td>Cruise Cont.</td>
<td>5</td>
<td>27</td>
<td>19</td>
<td>23</td>
<td>7</td>
<td>15</td>
</tr>
<tr>
<td>WCP1</td>
<td>55</td>
<td>50</td>
<td>15</td>
<td>-</td>
<td>10</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Spin vs. SMV

Table 6. Unreachable Cases and Test Sequence Lengths for Four Specifications.

<table>
<thead>
<tr>
<th>Specif.</th>
<th>Spin-Generated Test Sequences</th>
<th>SMV-Generated Test Sequences</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Useful</td>
<td>Unreach.</td>
</tr>
<tr>
<td>Small SIS</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>Large SIS</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>Cruise Cont.</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>WCP1</td>
<td>10</td>
<td>2?</td>
</tr>
</tbody>
</table>

Using Model Checking to Generate Tests from Requirement Specifications
Table 6. Unreachable Cases and Test Sequence Lengths for Four Specifications.

<table>
<thead>
<tr>
<th>Specif.</th>
<th>Spin-Generated Test Sequences</th>
<th>SMV-Generated Test Sequences</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Useful</td>
<td>Unreach.</td>
</tr>
<tr>
<td>Small SIS</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>Large SIS</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>Cruise Cont.</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>WCP1</td>
<td>10</td>
<td>2?</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Spin vs. SMV

Table 6. Unreachable Cases and Test Sequence Lengths for Four Specifications.

<table>
<thead>
<tr>
<th>Specif.</th>
<th>Spin-Generated Test Sequences</th>
<th>SMV-Generated Test Sequences</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Useful</td>
<td>Unreach.</td>
</tr>
<tr>
<td>Small SIS</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>Large SIS</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>Cruise Cont.</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>WCP1</td>
<td>10</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Conclusion

• Use of operational requirements specification for input and outputs
• Automatic generation of suite of test sequences
• Supports SMV and Spin
• Efficient use of model checking