An Industry Proof-of-Concept Demonstration of MC/DC from Automated Combinatorial Testing

Redge Bartholomew
Software Defects Drive Development Cost
... especially for safety-critical, embedded systems

• NIST: combinatorial tests can detect most latent defects
  – ... e.g., all 6-way combinations of input variable values

• NIST developed an approach to automating such testing
  – ... generation, execution, analysis

• Many systems require structural coverage (e.g., MC/DC)

• An experiment gauged the effectiveness of NIST approach
  – ... industry verification of a software radio’s monitor & control loop
The Approach

NIST/UT tool generates combinatorial test vectors;

User provides test vector generator with input variable definitions & values

... utility finds/exports counter example states that contain ACTS vectors (i.e., test cases);

... model checker generates counter examples;

... test harness imports, executes, analyzes test cases; identifies failures; measures coverage.

ACTS

VectorCAST

User provides model & properties of inputs & expected outputs

NuSMV
Defining the Input Space

ACTS model defines test inputs: more values yields better test coverage but also greatly increases the number of combinations.

To prevent state space explosion, values are limited to equivalence class representatives.

9 vectors (rows) contain all 37 two-way combinations of values.
Defining the Expected Outputs

Developer models input-output relationships

```
MODULE main
VAR
  a : {0,15,16};
  b : {255,256};
  c : {true,false};
  d : {-1,0,1};
DEFINE
  e :=
  case
    (c = true) : a + b;
    TRUE : a * d;
  esac;
```

```
When c = true, e = a + b, otherwise, e = a * d.
```

Utility creates properties from ACTS vectors

```
SPEC
AG !(a = 0 & b = 256 & c = false & d = -1)
AG !(a = 0 & b = 256 & c = true & d = 0)
AG !(a = 0 & b = 256 & c = true & d = 1)
AG !(a = 15 & b = 255 & c = true & d = -1)
AG !(a = 15 & b = 256 & c = false & d = 0)
AG !(a = 15 & b = 256 & c = false & d = 1)
AG !(a = 16 & b = 256 & c = true & d = -1)
AG !(a = 16 & b = 256 & c = true & d = 1)
```

Utility creates properties from ACTS vectors

```
Property specifies negative of ACTS vector

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>256</td>
<td>false</td>
<td>-1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>255</td>
<td>true</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>255</td>
<td>true</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
```
Creating & Exporting Test Cases

Model checker generates a counter example for each property

-- spec AG ... is 
false ...

-> State: 2.1 <-
  a = 0
  b = 255
  c = true
  d = -1
  e = 255

-> State: 2.2 <-
  a = 0
  b = 255
  c = true
  d = 0
  e = 255

-- spec AG ! ...
Unit Tests of Monitor & Control Loop

• 2775 tests generated, executed, analyzed in ~1 hour
  – 600 lines of C code with 34 inputs, 4 outputs of interest
  – 200+ defects arbitrarily seeded across code versions

• All defects were detected; achieved 95% MC/DC

• There were issues – e.g., state space explosion
  – In addition to using equivalence class values
  – … used multiple sets of vectors limited to interacting variables only
    (test harness set non-interacting variables to default values)
Another Issue: MC/DC of Nested Decisions

• MC/DC tests every path of every condition/decision
  – Each condition must *independently* affect the decision’s outcome

• Accepted practice: execute each decision
  – … when all conditions are true, and
  – … when each condition is false but the others true

• In some cases, this was non-trivial
  – … e.g., when a loop decision was nested within a decision
  – … that used the same condition variable as the loop decision
MC/DC of a Nested Decision

The *while-loop* must be tested when:
- radio_state \(\neq\) applications_running \& \(\neq\) no_waveform_available
- radio_state \(\neq\) applications_running \& = no_waveform_available
- radio_state = applications_running \& \(\neq\) no_waveform_available

The first two cases are ok; the third is a problem:
- radio_state’s value must change within the loop
- … but the toolset allows only 1 value/variable/vector
- … and only 1 vector/test
Required a Runtime Work-Around

Stub uses test-only variables to force decision outcomes

```c
if (radio_state != APPLICATIONS_RUNNING)
{
    while ((radio_state != APPLICATIONS_RUNNING) &&
           (radio_state != NO_WAVEFORM_AVAILABLE))
    {
        ...  
        radio_state = Current_Radio_State();
    }
}

bool first_pass = true;
int Current_Radio_State( void )
{
    int loop_condition;
    if (first_pass == true)
    {
        loop_condition = state_1;
        first_pass = false;
    }
    else loop_condition = state_2;
    return loop_condition;
}
```

control loop

ACTS model

stub
Summary: The Approach Was Effective

- Significant detection of latent defects
- Significant structural coverage
- Moderate effort
  - Learning to properly define the input space
  - Learning to write properties for expected outputs