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

Redge Bartholomew

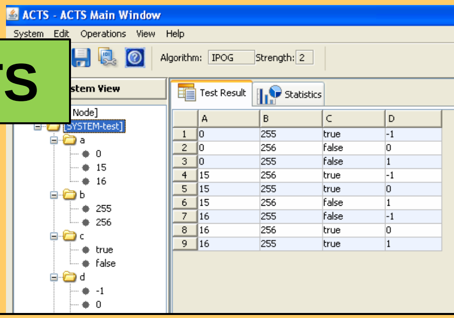
Software Defects Drive Development Cost

… 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

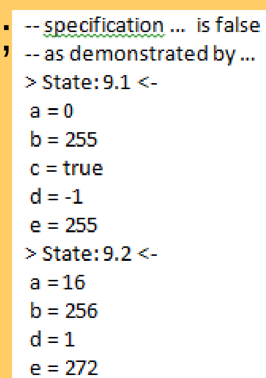
NIST/UT tool generates combinatorial test vectors;



ACTS

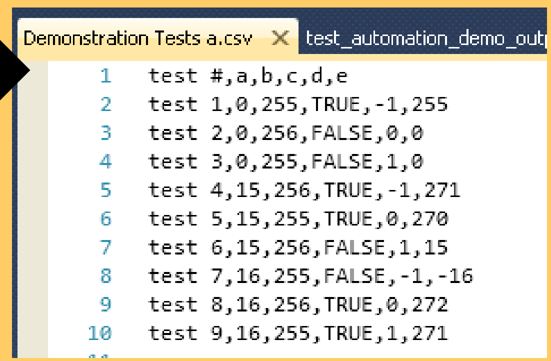
User provides test vector generator with input variable definitions & values

… model checker generates counter examples;



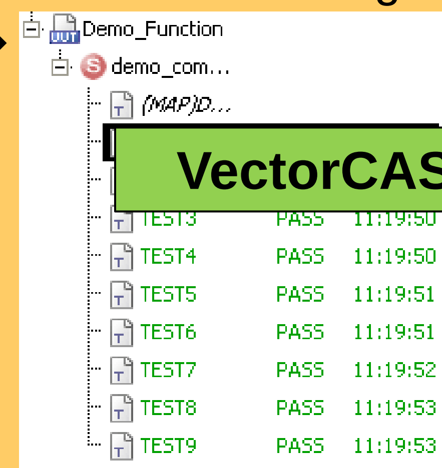
NuSMV

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



User provides model & properties of inputs & expected outputs

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

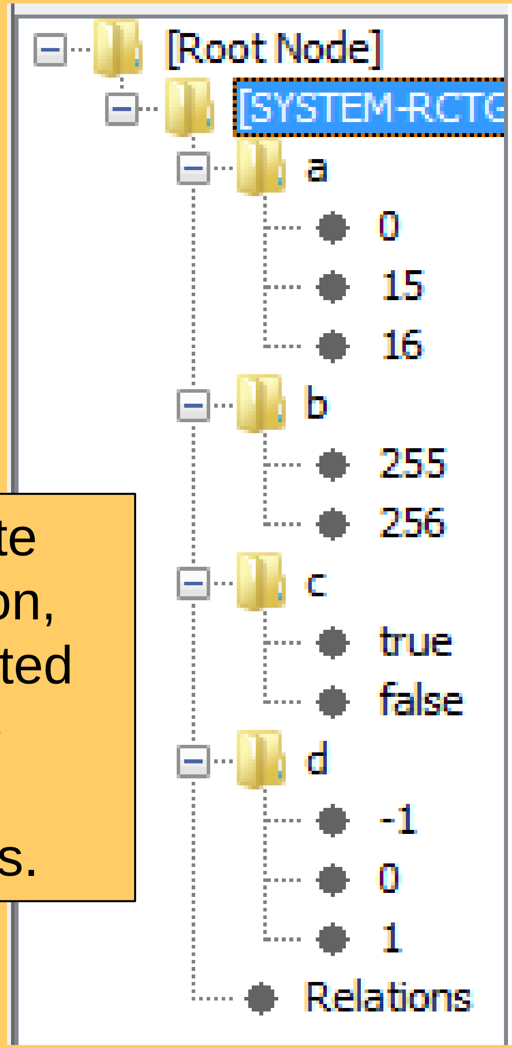


VectorCAST

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.



	A	B	C	D
1	0	256	false	-1
2	0	255	true	0
3	0	256	true	1
4	15	255	true	-1
5	15	256	false	0
6	15	255	false	1
7	16	256	true	-1
8	16	255	false	0
9	16	256	true	1

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

Property specifies negative of ACTS vector

	A	B	C	D
1	0	256	false	-1
2	0	255	true	0

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 ! ...
```

test case 2

ACTS vector 2

Utility finds state containing ACTS vector, reformats & exports it

test #	a	b	c	d	e
test 1	0	256	FALSE	-1	0
test 2	0	255	TRUE	0	255
test 3	0	256	TRUE	1	256
test 4	15	255	TRUE	-1	270
test 5	15	256	FALSE	0	0
test 6	15	255	FALSE	1	15
test 7	16	256	TRUE	-1	272
test 8	16	255	FALSE	0	0
test 9	16	256	TRUE	1	272

Unit Tests of Monitor & Control Loop

- 2775 tests generated, executed, analyzed in ~1hour
  - 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 ≠ applications_running & ≠ no_waveform_available
- radio_state ≠ applications_running & = no_waveform_available
- radio_state = applications_running & ≠ no_waveform_available

```
if (radio_state != applications_running)
{  while ((radio_state != applications_running) &&
   (radio_state != no_waveform_available))
   { ...
       radio_state = Current_Radio_State();
    }
}
```

control loop

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

Rockwell Collins

Required a Runtime Work-Around

Stub uses test-only variables to force decision outcomes

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

control loop

```
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;
}
```

stub

ACTS model

- state_1
  - APPLICATIONS_HALTED
  - APPLICATIONS_READY
  - APPLICATIONS_RUNNING
  - APPLICATION_IDLE_SENT
  - NO_WAVEFORM_AVAILABLE
- state_2
  - APPLICATIONS_RUNNING
  - NO_WAVEFORM_AVAILABLE

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