Using Combinatorial Methods to Determine Test Set Size

NASA 2014 International IV&V Workshop
Fairmont, WV

Rick Kuhn        Raghu Kacker                                Yu Lei
National Institute of Standards & Technology        University of Texas Arlington
kuhn@nist.gov,   raghu.kacker@nist.gov                               ylei@uta.edu
How many tests do we need?

Traditional approaches:

• Structural coverage based
  – Statement, branch; Stronger – MCDC etc.
• Requirements based
  – What qualifies as covering a requirement?
  – How many tests per requirement?
• Requirements + structural coverage
  – DO 178B,C
• Statistical
  – Error detection rate
  – Assumed distribution of faults
• Ad hoc
  – “Test thoroughly”
  – Until we run out of budget
Can we use test data in estimating?

- Input values
  - Cover all single values
  - Cover all pairs of values, 3-way interactions of values, etc.
- How far should we take this approach?
- Can it help with structural coverage?
- Using characteristics of test data in a new way to estimate
  - number of tests,
  - when to stop testing.
Background: how are faults distributed by interaction level?

• Interactions e.g., failure occurs if
  pressure < 10  (1-way interaction)
  pressure < 10 & volume > 300  (2-way interaction)
  pressure < 10 & volume > 300 & velocity = 5  (3-way interaction)

• Maximum interactions for fault triggering was 6
The Interaction Rule

• Most failures are triggered by one or two parameters, and progressively fewer by three or more parameters.

• Therefore if all faults in a system can be triggered by a combination of $t$ or fewer parameters, then testing all $t$-way combinations of parameter values is pseudo-exhaustive with a high rate of fault detection.

• The number of tests required to cover all $t$-way combinations is proportional to $v^t \log n$, for $n$ variables with $v$ values each.
**Combinatorial Coverage Measurement**

<table>
<thead>
<tr>
<th>Tests</th>
<th>Variables</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Variable pairs</th>
<th>Variable-value combinations covered</th>
<th>Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>ab</td>
<td>00, 01, 10</td>
<td>.75</td>
</tr>
<tr>
<td>ac</td>
<td>00, 01, 10</td>
<td>.75</td>
</tr>
<tr>
<td>ad</td>
<td>00, 01, 11</td>
<td>.75</td>
</tr>
<tr>
<td>bc</td>
<td>00, 11</td>
<td>.50</td>
</tr>
<tr>
<td>bd</td>
<td>00, 01, 10, 11</td>
<td>1.0</td>
</tr>
<tr>
<td>cd</td>
<td>00, 01, 10, 11</td>
<td>1.0</td>
</tr>
</tbody>
</table>

100% coverage of 33% of combinations
75% coverage of half of combinations
50% coverage of 16% of combinations
Graphing Coverage Measurement

100% coverage of 33% of combinations
75% coverage of half of combinations
50% coverage of 16% of combinations

Bottom line:
All combinations covered to at least 50%
Adding a test

Coverage after adding test [1,1,0,1]
Adding another test

Coverage after adding test [1,0,1,1]
Additional test completes coverage

Coverage after adding test [1,0,1,0]
All combinations covered to 100% level, so this is a covering array.
Combinatorial Coverage Measurement

Combinatorial Coverage Measurement

Number of tests: 7489
Number of parameters: 82

Values for this parameter:
0, 1

Coverage for file:
Total 3-way: 0.000

2-way stats:
Combinations: 3,321
Var/val coms: 14,761
Total coverage: 0.940

3-way stats:
Combinations: 88,560
Var/val coms: 828,135
Total coverage: 0.831
Minimum coverage value

\[ M = .50 \Rightarrow \text{all combinations covered to at least 50\% level} \]
Fault coverage

- Proportion of combinations that trigger faults covered by a test set
- Example coverage of a 2-way array

<table>
<thead>
<tr>
<th>Fault distribution</th>
<th>Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>60% single value</td>
<td>100% 1-way</td>
</tr>
<tr>
<td>25% 2-way</td>
<td>100% 2-way</td>
</tr>
<tr>
<td>10% 3-way</td>
<td>80% 3-way</td>
</tr>
<tr>
<td>5% 4-way</td>
<td>50% 4-way</td>
</tr>
</tbody>
</table>

Fault coverage = $1(.6) + 1(.25) + .8(.1) + .5(.05) = .955$
Why does this matter?

- Fault detection capacity of a t-way covering array can be greater than might be expected. Example:

![Graph showing fault distribution and fault coverage](image)

**Fault distribution**

**Fault coverage**
Decision predicates

code segment:

```
if (x <= 0 && y <= 0)  branch A
else            branch B
```

input model partitions of values for x and y:

```
x = {-9999, -1, 0, 1, 9999}  y = {-9999, -1, 0, 1, 9999}
```

Then of 25 value pairs,

- 9/25 go to A
- 16/25 go to B

\[ B_t = \text{minimum } \% \text{ of } t\text{-way settings triggering a branch in this code segment} \]

\[ B_t = \frac{9}{25} = 36\% \]
Branch coverage condition

- Where $B_t = \text{minimum proportion of } t\text{-way combinations to trigger a branch in code}$
- $M_t = \text{minimum } t\text{-way coverage}$
- If $M_t + B_t > 1$ then 100% branch coverage is achieved where all variables in decision predicates have values from the test variables with coverage characteristic $M_t$
Why does this matter?

• Allows determining how many tests needed for branch coverage

• In many cases, branch coverage is possible without a full covering array.

• Example:
  – 10 variables, 5 values each
  – Decision predicates with 2-way combinations
  – 309 tests required for 2-way covering array
  – If at least 2 combinations cause branch for every predicate, 225 tests provide full branch coverage

• Not as good as full covering array, but may still be highly effective
Implications for testing

- Uncertainty and range of estimates – wide variation for t<4, but 4-way fault coverage provides good estimates.
- Impact of # values per variable – as # values increases, supplemental coverage decreases, so more tests required for same fault coverage.
- Branch coverage condition – for branch predicates with t variables, usually don’t need full t-way array to achieve 100% branch coverage.
- Requirements specification – fault coverage is an additional dimension that can be specified in test goals.
- Use results to determine test set size, based on goals.
Factors in determining test set size

- Estimated fault distribution, e.g.
  - Based on previous similar systems
  - Conservative assumptions of distribution

- Assurance level goal
  - How much do extra tests contribute to assurance?

- Allowable range of uncertainty
  - How much is it affected by combinatorial coverage?

- Coverage goals: requirements, structural; requirements + structural coverage
  - Impact of combinatorial coverage
Est. fault coverage achieved with t = 2..5-way tests

Faults
>2-way
>3-way
1 0 0
2 0 0
3 0.5 0
4 0.25 0.5
5 0.25 0.25
6 0 0.25
Assurance level and range of uncertainty

Est. fault coverage achieved w/ $t = 2..5$-way tests

Lower assurance level and greater uncertainty

Higher assurance level and less uncertainty
Summary

• Combinatorial methods applied to input test data set provide a new dimension in evaluating test effectiveness
• Can be used in determining test set size commensurate with test goals and resources
• Establishes a relationship between combinatorial coverage (static) and code coverage (dynamic) useful for understanding test effectiveness and thoroughness
Contact: Rick Kuhn
kuhn@nist.gov

http://csrc.nist.gov/acts