Fault Tolerant Computing
Antirandom Testing

Yashwant K. Malaiya
Colorado State University

Automatic Test Generation using Checkpoint Encoding and Antirandom Testing

- Implementation of efficient automatic test generation.
- Checkpoint Encoded Antirandom Testing.
- Using code coverage to evaluate test effectiveness.
- Results and conclusions.
Getting better ROI from Testing

- Random testing doesn’t exploit info available for black-box testing. Inefficient for hard-to-test faults.
- Antirandom testing uses info about previous tests to find faults sooner.
- Checkpoints: to automat test generation

Antirandom Testing

- Each test in the antirandom sequence considers all previously applied tests.
- Each new test is as far away as possible from all other previously applied tests.
- Cartesian and Hamming Distance measures.
- Efficiently encode input space into binary.
- ATPG tool for binary test generation.
Cartesian and Hamming Distances

Given the variables of the vectors are all binary,

\[ CD(A, B) = \sqrt{\left| a_N - b_N \right| + \left| a_{N-1} - b_{N-1} \right| + \cdots + \left| a_0 - b_0 \right|} = \sqrt{HD(A, B)} \]

Maximal Distance Antirandom Test Sequence chooses each test \( t_i \) such that sum of distances from \( t_1, t_2, \ldots, t_{i-1} \) is maximum.

MCDATS is more strict than MHDATS.

Example: Generating Antirandom (partial) binary test sequence

\( \bullet \) Choose \( t_0 \) arbitrarily, say \( t_0 = 000000 \).

\( \bullet \) Next two valid MHDATS and MCDATS:

\[
\begin{align*}
  t_0 & = 000000 & t_0 & = 000000 \\
  t_1 & = 111111 & t_1 & = 111111 \\
  t_2 & = 101010 & t_2 & = 000001 \\
\end{align*}
\]

Only the first sequence is valid MCDATS.
Checkpoint Encoding

- An integral part of antirandom testing
- Enables efficient capture of proper combinations of typical, boundary and illegal test cases.
- Motivation is to exercise not only usual program behavior but also boundary cases.

Proposed Checkpoint Encoded Antirandom Testing (CEAR) scheme
Experiments based on CEAR scheme

- Generate *checkpoint encoding* from program specifications.
  - Generate *Antirandom Test* vector sequence (with checkpoint encoding).
  - *Random Testing with checkpoint* encoding.
- Use *code coverage* (branch, loop, etc.) to evaluate effectiveness of test approaches.

**STRMAT** program - Given a *string* 0-80 chars, a *pattern* of upto 3 chars long, returns the *position* of string where it matches the pattern.

<table>
<thead>
<tr>
<th><strong>Text length</strong></th>
<th>b2,b1,b0</th>
<th>110</th>
<th>010</th>
<th>011</th>
<th>rest</th>
<th>0</th>
<th>80 (max)</th>
<th>80&lt;length&lt;100 (illegal)</th>
<th>1&lt;length&lt;79</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Pattern position</strong></td>
<td>b5,b4,b3</td>
<td>110</td>
<td>010</td>
<td>011</td>
<td>rest</td>
<td>Outside (illegal)</td>
<td>Beginning</td>
<td>End</td>
<td>Middle</td>
</tr>
<tr>
<td><strong>Pattern length</strong></td>
<td>b8,b7,b6</td>
<td>110</td>
<td>010</td>
<td>011</td>
<td>rest</td>
<td>0</td>
<td>3 (pmax)</td>
<td>3&lt;plen&lt;10 (illegal)</td>
<td>1&lt;plen&lt;2</td>
</tr>
</tbody>
</table>
STRMAT: Branch Coverage

AE - Antirandom with checkpoint Encoding
RE - Random with checkpoint Encoding
RW1, RW2 - Pure random with two different seeds

STRMAT: Loop Coverage

AE - Antirandom with checkpoint Encoding
RE - Random with checkpoint Encoding
RW1, RW2 - Pure random with two different seeds
STRMAT: Relational Coverage

- **AE**: Antirandom with checkpoint Encoding
- **RE**: Random with checkpoint Encoding
- **RW1, RW2**: Pure random with two different seeds

![Graph of Relational Coverage](image_url)

STRMAT: Total Coverage

- **AE**: Antirandom with checkpoint Encoding
- **RE**: Random with checkpoint Encoding
- **RW1, RW2**: Pure random with two different seeds

![Graph of Total Coverage](image_url)
TRIANGLE: Given length of three sides, is it a triangle? Which kind?

| Not a triangle | b4,b3,b2,b1,b0 | X1111 | a+b≤c, a!=b or a=b |
|               |               | X1001 | b+c≤a, b!=c or b=c |
|               |               | X0011 | a+c≤b, a!=c, or a=c |
|               |               | X0100 | a+b=c, b!=c or b=c |
|               |               | X0101 | b+c=a, b!=c or b=c |
|               |               | X1100 | a+c=b, a!=c or a=c |

| Legal triangle | b4,b3,b2,b1,b0 | 01010 | a=b (isosceles) |
|               |               | 11010 | a=c (isosceles) |
|               |               | 00110 | b=c (isosceles) |
|               |               | 10110 | a=b=c (equilateral) |
|               |               | rest  | Scalene |

TRIANGLE: Coverage Comparison

AE- Antirandom with checkpoint Encoding
RE- Random with checkpoint Encoding
RW1, RW2- Pure random with two different seeds
**FIND** program - Takes an integer array B of size \( S \geq 1 \) and index \( F \). Sort s.t. elements to left of \( B(F) \), are no larger than \( B(F) \); and elements to right of \( B(F) \) are no smaller than \( B(F) \).

<table>
<thead>
<tr>
<th>Field</th>
<th>Bits</th>
<th>Value</th>
<th>Significance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Array Size</td>
<td>b1, b0</td>
<td>01</td>
<td>1,2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>rest</td>
<td>&gt;2</td>
</tr>
<tr>
<td>Array status</td>
<td>b4,b3,b2</td>
<td>110</td>
<td>Already ordered</td>
</tr>
<tr>
<td></td>
<td></td>
<td>100</td>
<td>Reverse ordered</td>
</tr>
<tr>
<td></td>
<td></td>
<td>011</td>
<td>All equal</td>
</tr>
<tr>
<td></td>
<td></td>
<td>rest</td>
<td>Randomly ordered</td>
</tr>
<tr>
<td>Element values</td>
<td>b7,b6,b5</td>
<td>010</td>
<td>All positive</td>
</tr>
<tr>
<td></td>
<td></td>
<td>101</td>
<td>All negative</td>
</tr>
<tr>
<td></td>
<td></td>
<td>rest</td>
<td>Mixed</td>
</tr>
<tr>
<td>F points to</td>
<td>b9,b8</td>
<td>1</td>
<td>First element</td>
</tr>
<tr>
<td></td>
<td></td>
<td>01</td>
<td>Last element</td>
</tr>
<tr>
<td></td>
<td></td>
<td>rest</td>
<td>A middle element</td>
</tr>
</tbody>
</table>

**FIND: Coverage Comparison**

AE- Antirandom with checkpoint Encoding
RE- Random with checkpoint Encoding
RW1, RW2- Pure random with two different seeds
Remarks

- Using a coverage measure as indicator of effectiveness. Limitations.
- Shows automatic test generation using a more intelligent approach.
- The CEAR scheme can be used automatic testing for large programs.

Conclusions & Continuing Work

- Encoding significantly controls effectiveness.
- Distribution: usual Vs special combinations.
- Exploiting some implementation info.
- Larger and diverse programs.
- Process automation.
Antirandom Testing of Hardware

C880 stuck-at coverage

Antirandom Testing: Hardware

Antirandom seed: 000..00
Psuedo-random seed: 0101..01
Construction of a MHDATS
(MCDATS)

Procedure 1: Exhaustive search

- **Step 1.** For each of N input variables, assign an arbitrarily chosen value to obtain the first test vector. As discussed below this does not result in any loss of generality.
- **Step 2.** To obtain each new vector, evaluate the THD(TCD) for each of the remaining combinations with respect to the combinations already chosen and choose one that gives maximal distance. Add it to the set of selected vectors.
- **Step 3.** Repeat step 2 until all $2^N$ combinations have been used.

This procedure uses exhaustive search. As we will see later, the computational complexity can be greatly reduced.

To illustrate the process of generating MDATS, we consider in detail the generation of a complete sequence for three binary variables.

Example: 3-bit antirandom sequence

<table>
<thead>
<tr>
<th>Test</th>
<th>xyz</th>
<th>THD</th>
<th>TCD</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_0$</td>
<td>0 0 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$t_1$</td>
<td>1 1 1</td>
<td>3</td>
<td>1.7320</td>
</tr>
<tr>
<td>$t_2$</td>
<td>0 1 0</td>
<td>3</td>
<td>2.4142</td>
</tr>
<tr>
<td>$t_3$</td>
<td>1 0 1</td>
<td>6</td>
<td>4.146</td>
</tr>
<tr>
<td>$t_4$</td>
<td>1 0 0</td>
<td>6</td>
<td>4.8284</td>
</tr>
<tr>
<td>$t_5$</td>
<td>0 1 1</td>
<td>9</td>
<td>6.5604</td>
</tr>
<tr>
<td>$t_6$</td>
<td>1 1 0</td>
<td>9</td>
<td>7.2426</td>
</tr>
<tr>
<td>$t_7$</td>
<td>0 0 1</td>
<td>12</td>
<td>8.9746</td>
</tr>
</tbody>
</table>
Theorems

**Definition**: If a sequence $B$ is obtained by reordering the variables of sequence $A$, then $B$ is a **variable-order-variant (VOV)** of $A$.

**Theorem 1**: If a sequence $B$ is variable-order-variant of a MHDATS (MCDATS) $A$, then $B$ is also a MHDATS (MCDATS).
The theorem follows from the fact that Hamming or Cartesian distance is independent of how the variables are ordered.

**Theorem 2**: If a sequence $B$ is a **polarity-variant** of a MHDATS (MCDATS) $A$, then $B$ is also MHDATS (MCDATS).
The theorem follows from the fact that for a pair of vectors the distance remains the same, if the same set of variables in both are complemented.

**Theorem 3**: A MHDATS (MCDATS) will always contain **complementary pair of vectors**, i.e. $t_{2k}$ will always be followed by $t_{2k+1}$ which is complementary for all bits in $t_{2k}$ where $k = 1, 2, \ldots$.

---

Expansion and unfolding (ATG)

Procedure 2. Expansion of MHDATS (MCDATS):
Step 1. Start with a complete MHDATS of $N$ variables, $x_{i1}, x_{i2}, \ldots, x_{iN}$.
Step 2. For each vector $x_i$, $i = 0, 1, \ldots, (2^N-1)$, add an additional bit corresponding to an added variable $x_j$, such that $x_i$ has the maximum total HD (CD) with respect to all previous vectors.

Procedure 3. Expansion and Unfolding of a MHDATS (MCDATS):
Step 0. Start with a complete (N-1) variable MHDATS (MCDATS) with $2^{N-1}$ vectors.
Step 1. Expand by adding a variable using Procedure 2. We now have the first $(2^N/2)$ vectors needed.
Step 2. Complement one of the columns and append the resulting vectors to first set of vectors obtained in Step 1. Here, it would be convenient to complement the variable added in Step 1.
Variations

- **FAR**: fast antirandom: starting with a partial sequence
- **Random-like**: alternate vector flipped

More at:

