Automatic Test Generation using Checkpoint Encoding and Antirandom Testing

H. Yin
Z. Lebne-Dengel*
Y. K. Malaiya

Colorado State University
*Hewlett-Packard Co.

Outline

◆ 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

◆ Exhaustive testing is expensive.
◆ Efficiency and effectiveness in testing.
◆ Random testing doesn’t exploit info available for black-box testing.
◆ Antirandom testing attempts to use info about previous tests.
◆ The role of automated generation of tests.

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,

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

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

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

◆ Next two valid MHDATS and MCDATS:

\[ t_0 = 000000 \quad t_0 = 000000 \]
\[ t_1 = 111111 \quad t_1 = 111111 \]
\[ t_2 = 101010 \quad t_2 = 000001 \]

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.
  **Pure Random Testing.**

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

Tests

Branch Coverage %

STRMAT: Loop Coverage

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

Tests

Loop coverage %
STRMAT: Relational Coverage

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

![Graph showing relational coverage over tests](image)

STRMAT: Total Coverage

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

![Graph showing total coverage over tests](image)
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

![Graph showing coverage comparison](image-url)
**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></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