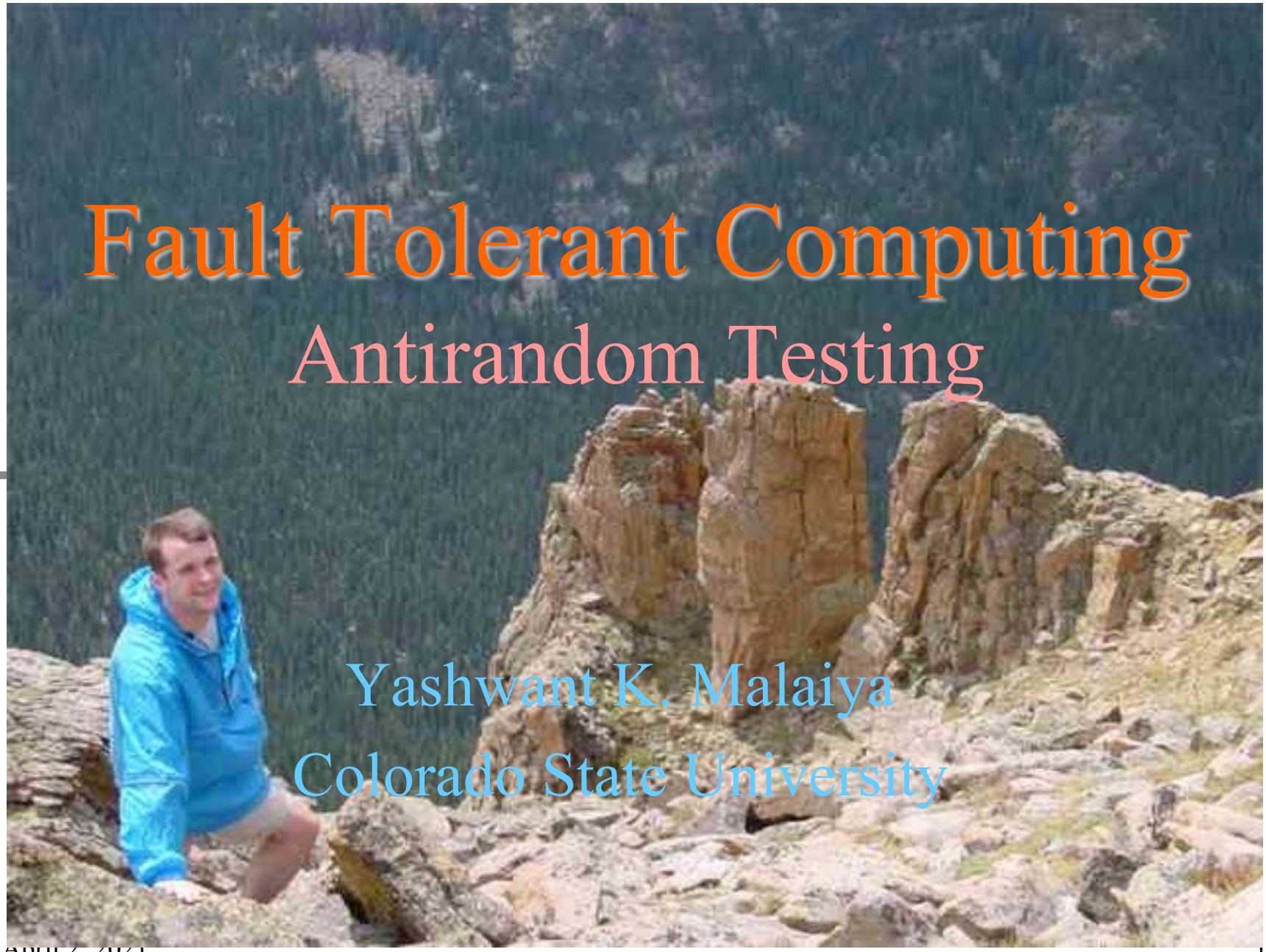
Fault Tolerant Computing

Antirandom Testing



Yashwant K. Malaiya

Colorado State University

Updates: Spring 21

- Project Reports due: PR: 4/9, Final 5/5
- Presentations: Powerpoint/Videos 4/21
- Extra Credit: prop 4/10, Final Rep 5/11

Automatic Test Generation using Checkpoint Encoding and Antirandom Testing

- Implementation of efficient automatic test generation.
- Antirandom Vs Random Testing. Tradeoffs.
- 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.
  - Exercise the unit under test more thoroughly
  - Find possible faults sooner
- Cartesian and Hamming Distance measures.
- Efficiently encode input space into binary.
- ATPG tool for binary test generation.

Outline

- Idea of Antirandom sequences Shinsho-ji Temple, Narita-san
- Software testing: checkpoint encoding
- Hardware Testing
- Open questions
- Recent developments

Hamming & Cartesian Distances

Hamming distance between two binary strings is given by

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

Cartesian distance between two binary strings is given by

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

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

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, … 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 MHDATSs:

$t_0 = 000000$	$t_0 = 000000$
$t_1 = 111111$	$t_1 = 111111$
$t_2 = 101010$	$t_2 = 000001$

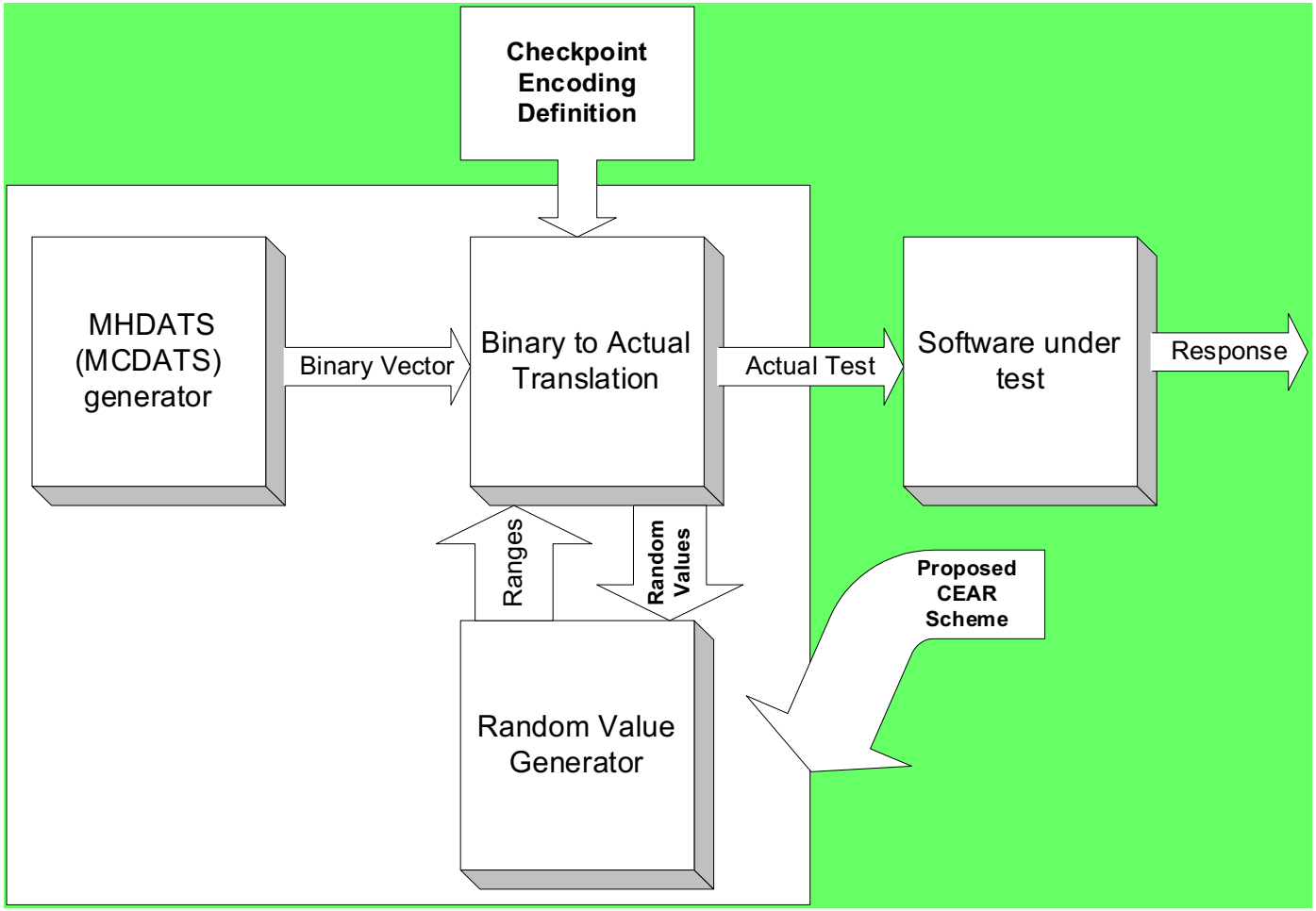
Only the first sequence is valid MCDATS.

- How to construct sequences? Later.

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

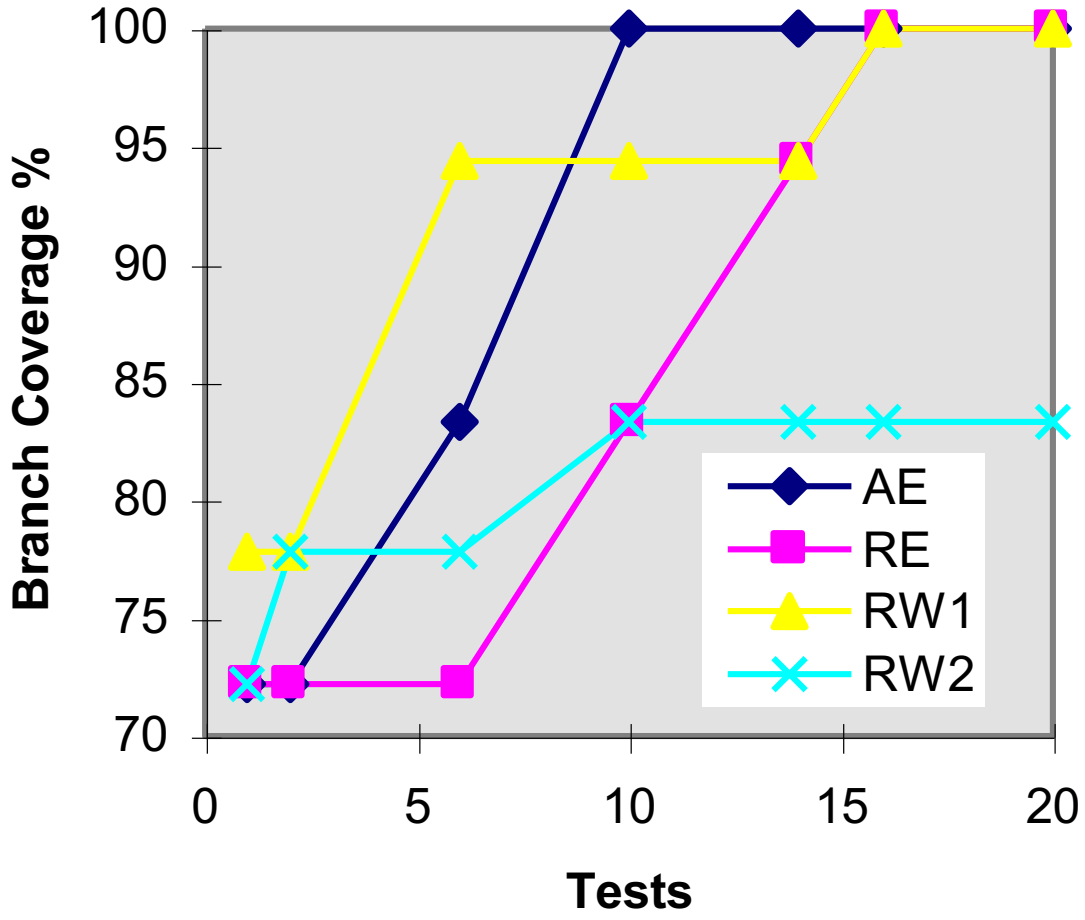
- Generate *checkpoint encoding* from program specifications.
- Generate
  - *Antirandom Test* vector sequence (with checkpoint encoding).
  - *Random Testing with checkpoint* encoding.
  - *Pure* Random Testing.
- Use *code coverage* (branch, loop, etc.) to evaluate effectiveness of test approaches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

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.

Text length	b2,b1,b0	110 010 011 rest	0 80 (max) 80<length<100 (illegal) 1<length<79
Pattern position	b5,b4,b3	110 010 011 rest	Outside (illegal) Beginning End Middle
Pattern length	b8,b7,b6	110 010 011 rest	0 3 (pmax) 3<plen<10 (illegal) 1<plen<2

Taken from a set of benchmark programs
W. E. Wong et al, ICSE, 1995, pp. 41-50.

STRMAT: Branch Coverage

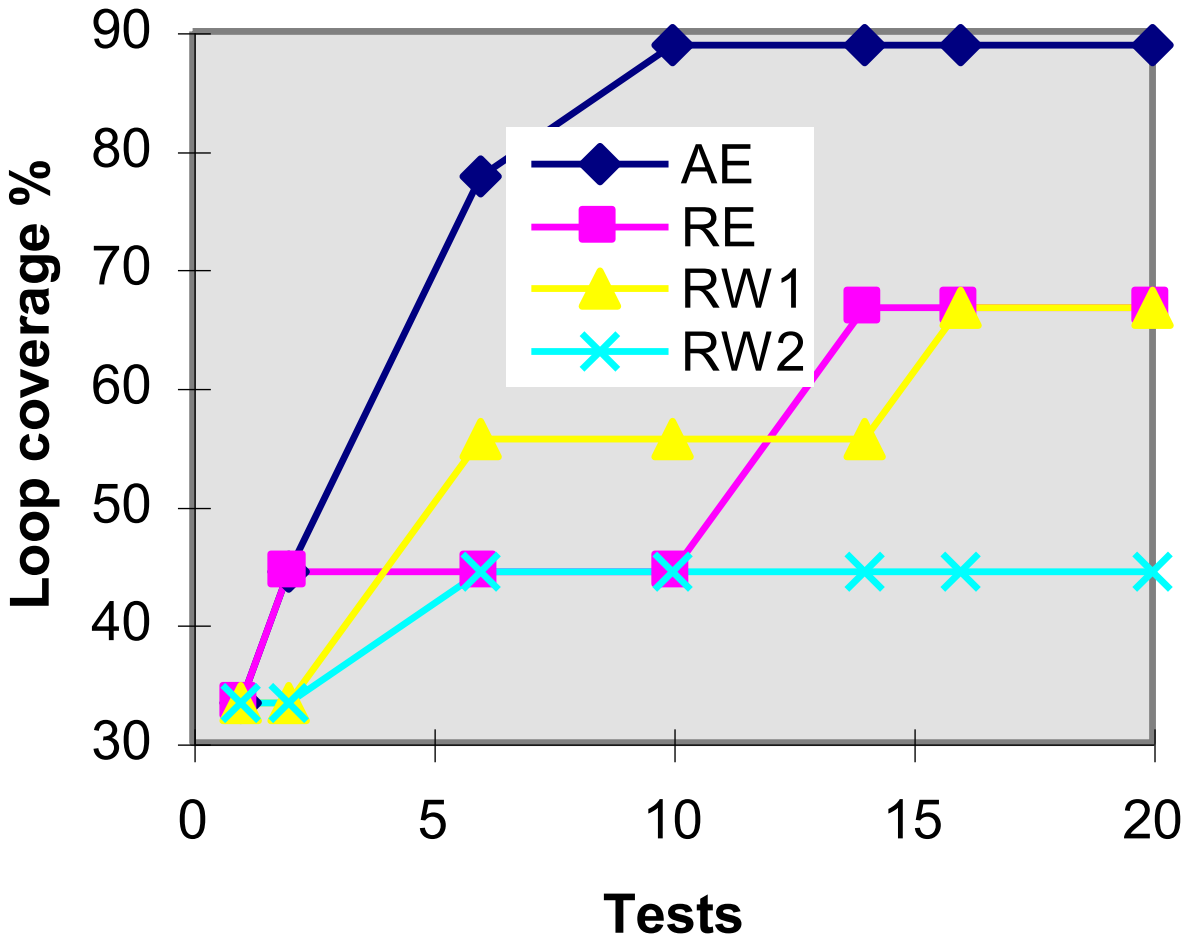


AE- Antirandom with checkpoint Encoding

RE- Random with checkpoint Encoding

RW1, RW2- Pure random with two different seeds

STRMAT: Loop Coverage

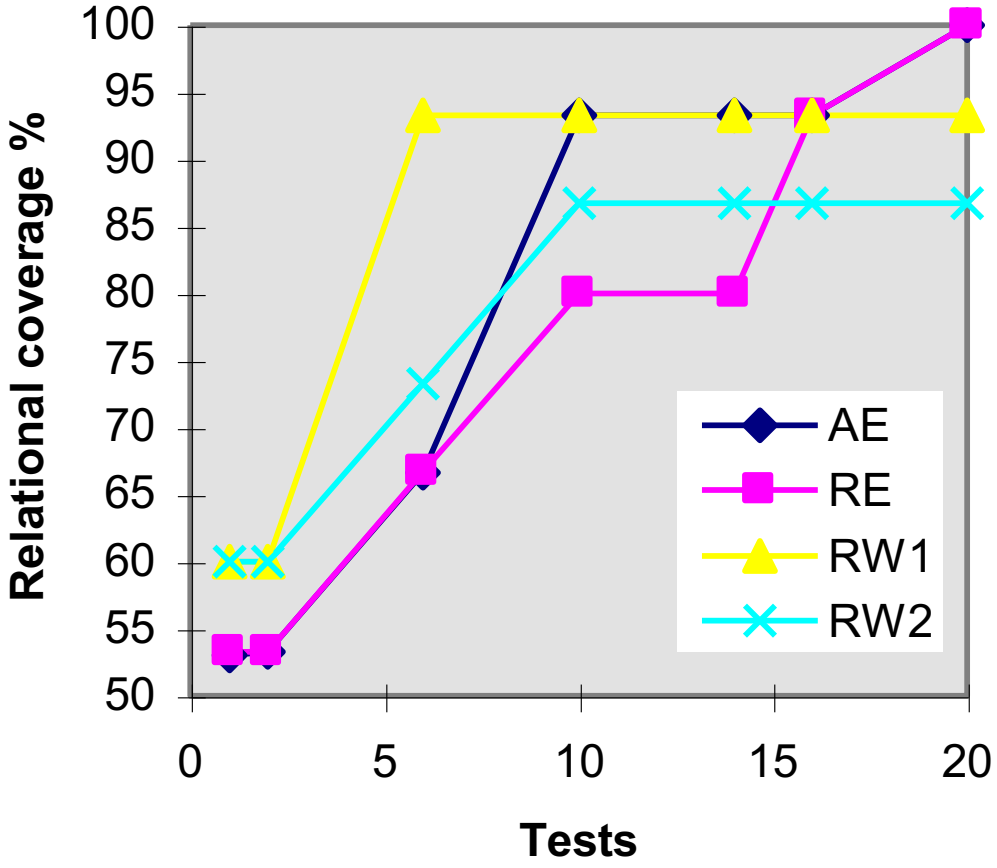


AE- Antirandom with checkpoint Encoding

RE- Random with checkpoint Encoding

RW1, RW2- Pure random with two different seeds

STRMAT: Relational Coverage

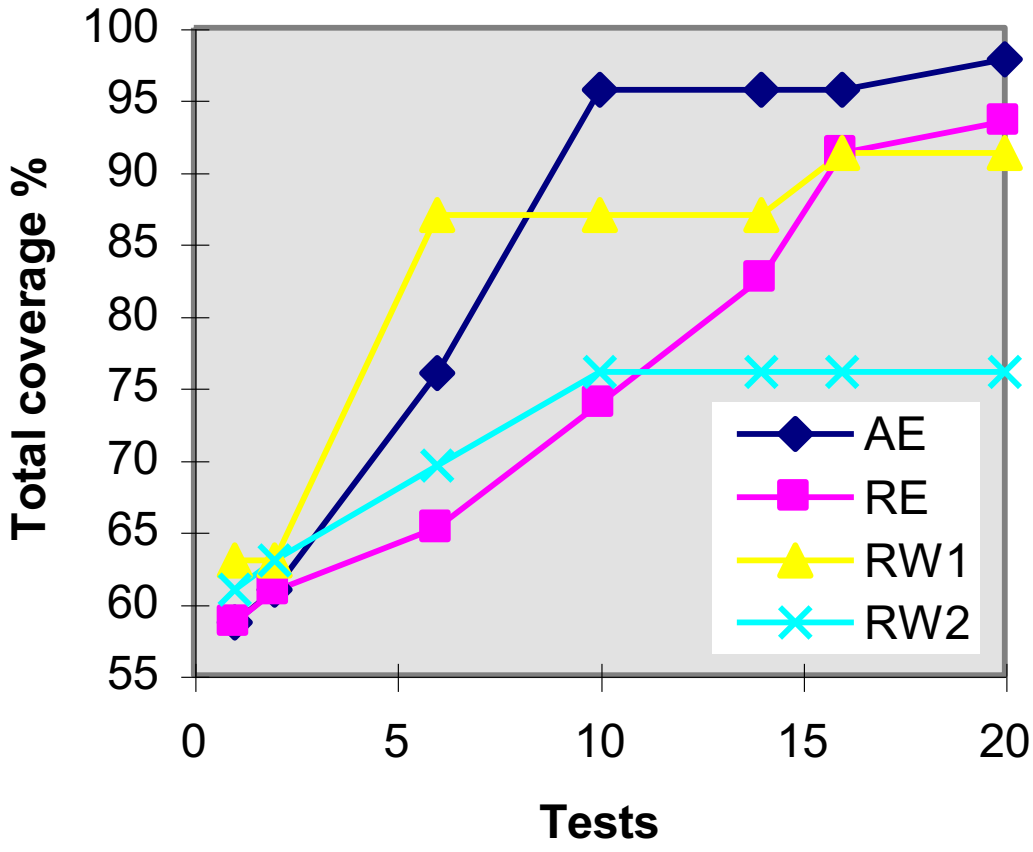


AE- Antirandom with checkpoint Encoding

RE- Random with checkpoint Encoding

RW1, RW2- Pure random with two different seeds

STRMAT: Total Coverage



AE- Antirandom with checkpoint Encoding

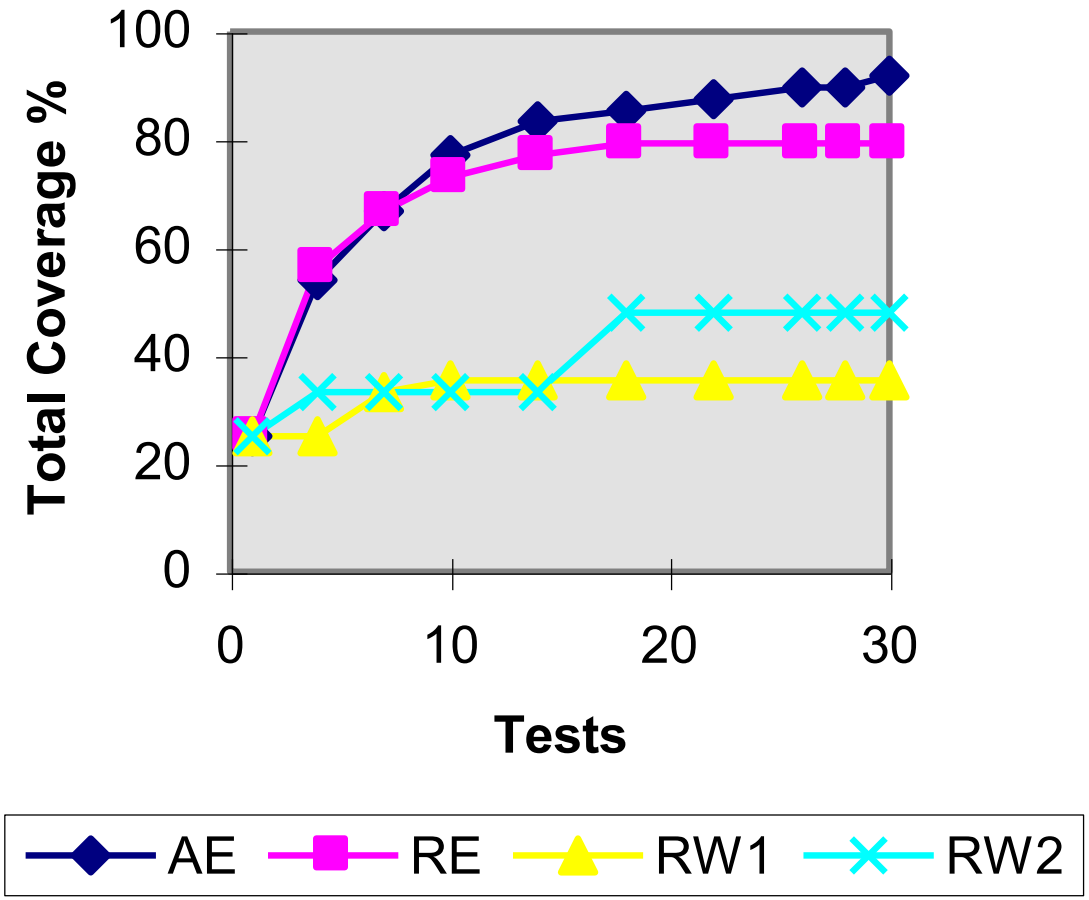
RE- Random with checkpoint Encoding

RW1, RW2- Pure random with two different seeds

TRIANGLE: Given length of three sides, is it a triangle? Which kind?

Not a triangle	b4,b3,b2,b1,b0	X1111 X1001 X0011 X0100 X0101 X1100	a+b<c,a!=b or a=b b+c<a, b!=c or b=c a+c<b, a!=c, or a=c a+b=c, b!=c or b=c b+c=a, b!=c or b=c a+c=b, a!=c or a=c
Legal triangle	b4,b3,b2,b1,b0	01010 11010 00110 10110 rest	a=b (isosceles) a=c (isosceles) b=c (isosceles) a=b=c (equilateral) Scalene

TRIANGLE: Coverage Comparison



AE- Antirandom with checkpoint Encoding

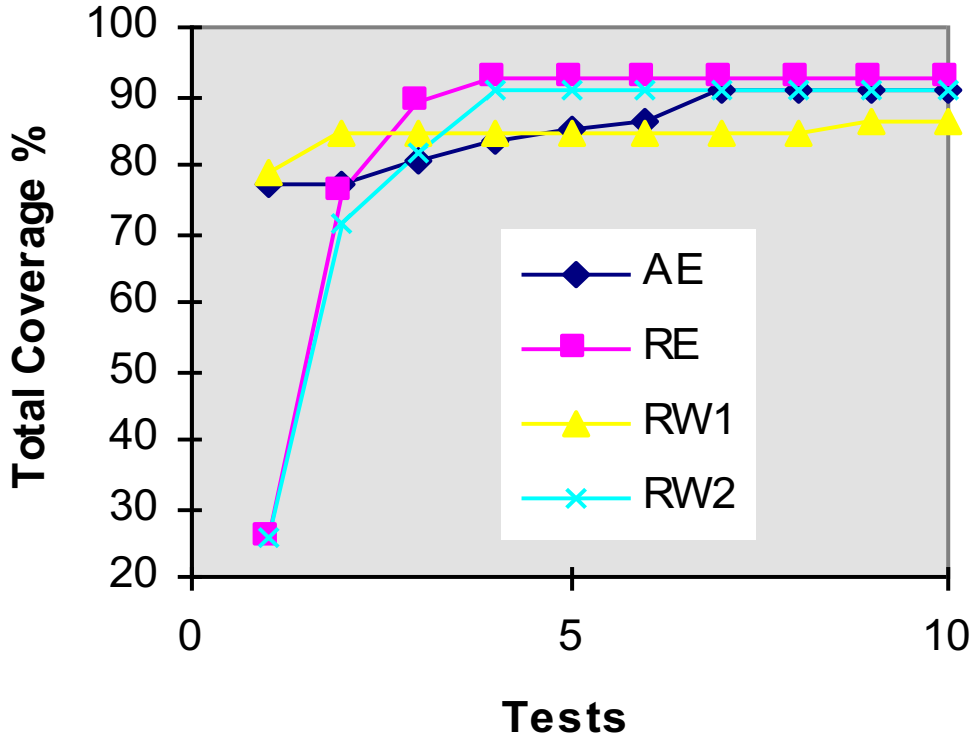
RE- Random with checkpoint Encoding

RW1, RW2- Pure random with two different seeds

FIND program - Takes an integer array B of size S>=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)

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

FIND: Coverage Comparison



AE- Antirandom with checkpoint Encoding

RE- Random with checkpoint Encoding

RW1, RW2- Pure random with two different seeds

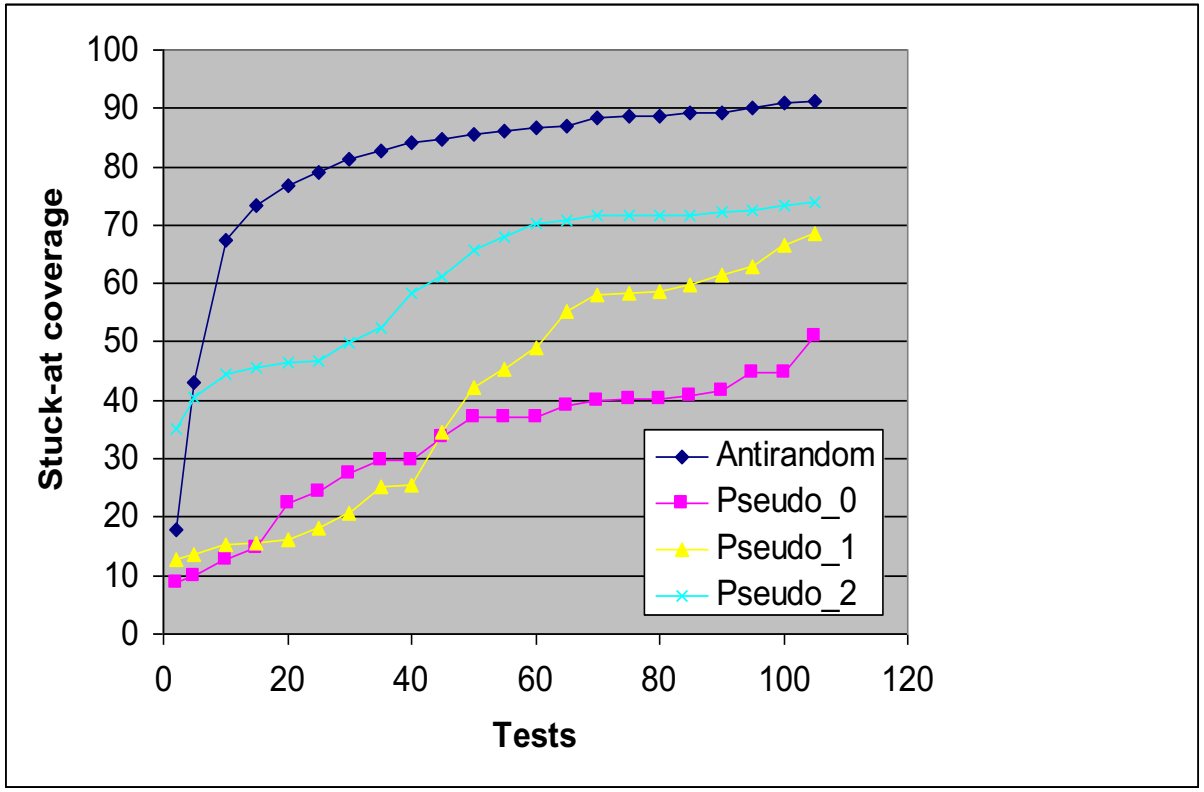
Remarks

- 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

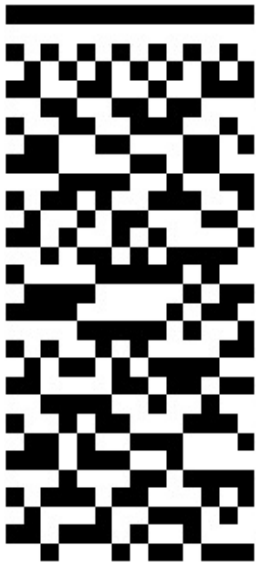
- Encoding significantly controls effectiveness.
- Distribution: usual Vs special combinations.
- Exploiting some implementation info.
- Larger and diverse programs.
- Process automation.

Antirandom Testing of Hardware

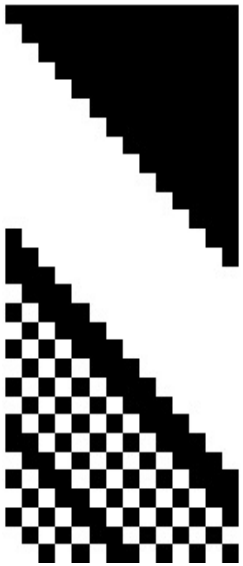


C880 stuck-at coverage

Antirandom Testing: Hardware



Antirandom



Psuedo-random
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 theTHD(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

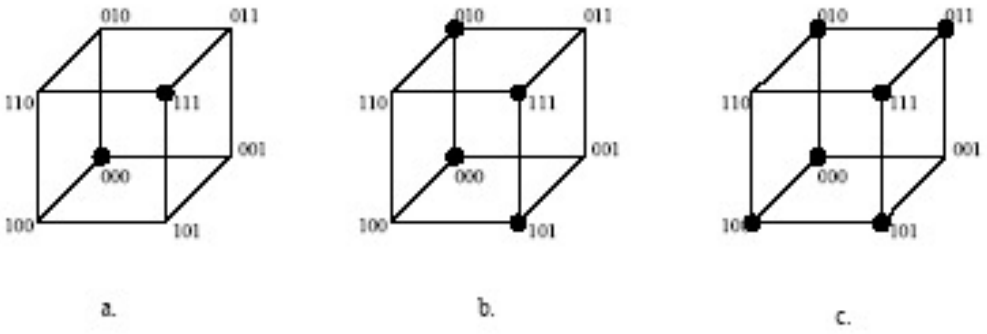


Table 1: 3-bit MHDTS (Example 3)

Test	xyz	THD	TCD
t_0	0 0 0		
t_1	1 1 1	3	1.7320
t_2	0 1 0	3	2.4142
t_3	1 0 1	6	4.146
t_4	1 0 0	6	4.8284
t_5	0 1 1	9	6.5604
t_6	1 1 0	9	7.2426
t_7	0 0 1	12	8.9746

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; : : :.

Expansion and unfolding (ATG)

Procedure 2. Expansion of MHDATS (MCDATS):

Step 1. Start with a complete MHDATS of N variables, $X_{N-1}, X_{N-2}, \ldots X_1, X_0$.

Step 2. For each vector t_i, i = 0, 1, ... $(2^N\text{-}1)$, add an additional bit corresponding to an added variable X_N, such that t_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. □

Antirandom Variations

- FAR: fast antirandom: starting with a partial sequence 1998
- Random-like: alternate vector flipped 1998
- Adaptive Random Testing 2004
- Controlled Random Tests 2017

•Y. K. Malaiya, "Antirandom Testing: Getting the most out of black-box testing," Proc. ISSRE, 1995, pp. 86-95. Selected as "30 years of ISSRE - Most influential papers"

•A. von Mayrhauser ,T. Chen and A. Hajjar and A. Bai and C. Anderson", "Fast Antirandom (FAR) Test Generation, Proc. HASE, 1998, pp. 262-269.

• S. Xu, J. Gao, "An efficient random-like testing," Proc. ATS '98, pp. 504-508S.

•H. Wu, S. Jandhyala, Y. K. Malaiya, A. P. Jayasumana, "Antirandom Testing: A Distance Based Approach," VLSI Design, 2008.

•T. Y. Chen, H. Leung, and I. K. Mak. Adaptive random testing. In Advances in Computer Science, pages 320–329, 2004.

Applications: examples

- Software Testing
- State based software or hardware testing
- Testing for vulnerabilities
- Hardware testing: stuck and delay faults
- VHDL testing
- Functional verification
- Cryptography
- Internet cookie collection testing
- Fault detection in clouds

Unsolved problems

- Proving that our procedure is an algorithm
- A more efficient algorithm
- Comparing and connecting approaches based on antirandom sequences
- Combing antirandom and deterministic approaches
- Obtaining and exploiting internal antirandom states