Advanced Topics in Combinatorial Methods for Testing

Rick Kuhn
National Institute of
Standards and Technology
Gaithersburg, MD

Solutions to the oracle problem

How to automate checking correctness of output

- **Creating test data is the easy part!**
- How do we check that the code worked correctly on the test input?
  - **Crash testing** server or other code to ensure it does not crash for any test input (like 'fuzz testing')
    - Easy but limited value
  - **Built-in self test with embedded assertions** – incorporate assertions in code to check critical states at different points in the code, or print out important values during execution
  - **Full scale model-checking** using mathematical model of system and model checker to generate expected results for each input
    - expensive but tractable

NIST
National Institute of Standards and Technology

Crash Testing

- Like "fuzz testing" - send packets or other input to application, watch for crashes
- Unlike fuzz testing, input is non-random; cover all t-way combinations
- May be more efficient - random input generation requires several times as many tests to cover the t-way combinations in a covering array

Limited utility, but can detect high-risk problems such as:

- buffer overflows
- server crashes

National Institute of Standards and Technology

Built-in Self Test through Embedded Assertions

Simple example:

assert(x != 0);   // ensure divisor is not zero

Or pre and post-conditions:

/requires amount >= 0;

/ensures balance == \old(balance) - amount &&
\result == balance;

Built-in Self Test

Assertions check properties of expected result:

```
ensures balance  == \old(balance) - amount
  &&  \result == balance;
```

•Reasonable assurance that code works correctly across the range of expected inputs

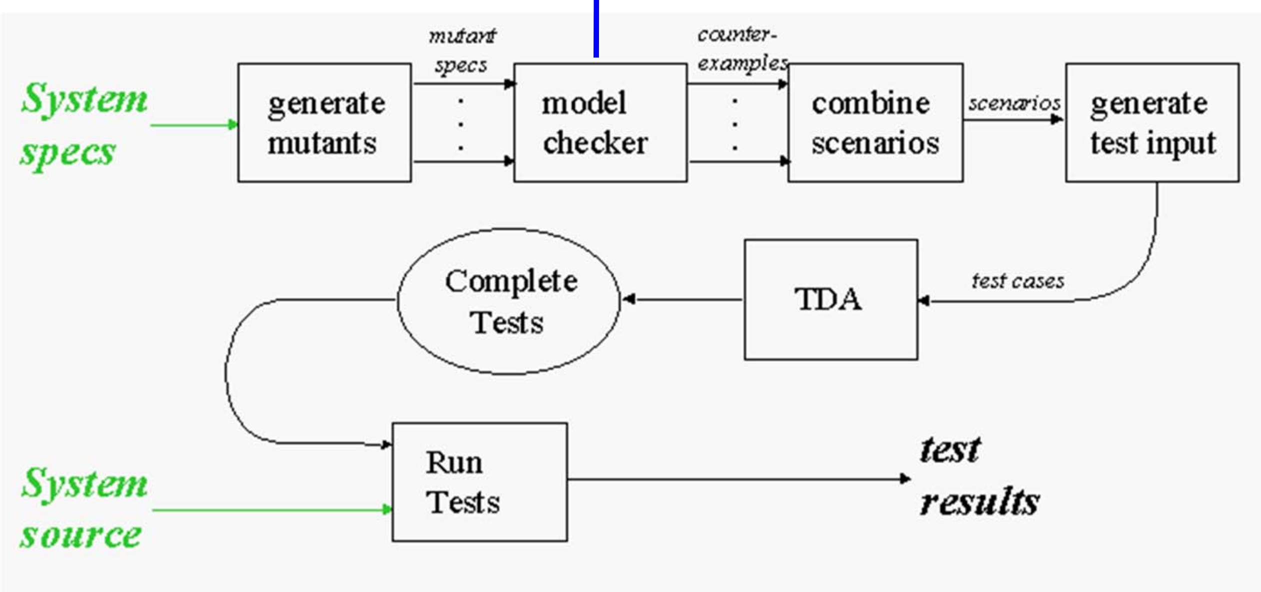
•May identify problems with handling unanticipated inputs

•Example: Smart card testing

- Used Java Modeling Language (JML) assertions
- Detected 80% to 90% of flaws

NIST
National Institute of Standards and Technology

Using model checking to produce tests



Black & Ammann, 1999

- Model-checker test production: if assertion is not true, then a counterexample is generated.
- This can be converted to a test case.

Model checking example

```
-- specification for a portion of tcas - altitude separation.
-- The corresponding C code is originally from Siemens Corp.
Research
-- Vadim Okun 02/2002
MODULE main
VAR
  Cur_Vertical_Sep : { 299, 300, 601 };
  High_Confidence : boolean;
...
init(alt_sep) := START_;
  next(alt_sep) := case
    enabled & (intent_not_known | !tcas_equipped) : case
      need_upward_RA & need_downward_RA : UNRESOLVED;
      need_upward_RA : UPWARD_RA;
      need_downward_RA : DOWNWARD_RA;
      1 : UNRESOLVED;
    esac;
    1 : UNRESOLVED;
  esac;
...
SPEC AG ((enabled & (intent_not_known | !tcas_equipped) &
!need_downward_RA & need_upward_RA) -> AX (alt_sep = UPWARD_RA))
-- "FOR ALL executions,
-- IF enabled & (intent_not_known ....
-- THEN in the next state alt_sep = UPWARD_RA"
```

Computation Tree Logic

- The usual logic operators,plus temporal:
  A φ - All: φ holds on all paths starting from the current state.
  E φ - Exists: φ holds on some paths starting from the current state.
  G φ - Globally: φ has to hold on the entire subsequent path.
  F φ - Finally: φ eventually has to hold
  X φ - Next: φ has to hold at the next state
- [others not listed]

- execution paths
- states on the execution paths
-
```
SPEC AG ((enabled & (intent_not_known |
!tcas_equipped) & !need_downward_RA & need_upward_RA)
-> AX (alt_sep = UPWARD_RA))
```

- "FOR ALL executions,
  IF enabled & (intent_not_known ....
  THEN in the next state alt_sep = UPWARD_RA"

What is the most effective way to integrate combinatorial testing with model checking?

- Given `AG(P -> AX(R))`
  "for all paths, in every state,
  if P then in the next state, R holds"
- For k-way variable combinations, `v1 & v2 & ... & vk`
- vi abbreviates "var1 = val1"
- Now combine this constraint with assertion to produce counterexamples. Some possibilities:

1. `AG(v1 & v2 & ... & vk & P -> AX !(R))`
2. `AG(v1 & v2 & ... & vk -> AX !(1))`
3. `AG(v1 & v2 & ... & vk -> AX !(R))`

National Institute of Standards and Technology

What happens with these assertions?

```
1. AG(v1 & v2 & ... & vk & P -> AX !(R))
```

P may have a negation of one of the v_i, so we get

```
0 -> AX !(R))
```

always true, so no counterexample, no test.

This is too restrictive!

```
1. AG(v1 & v2 & ... & vk -> AX !(1))
```

The model checker makes non-deterministic choices for variables not in v1..vk, so all R values may not be covered by a counterexample.

This is too loose!

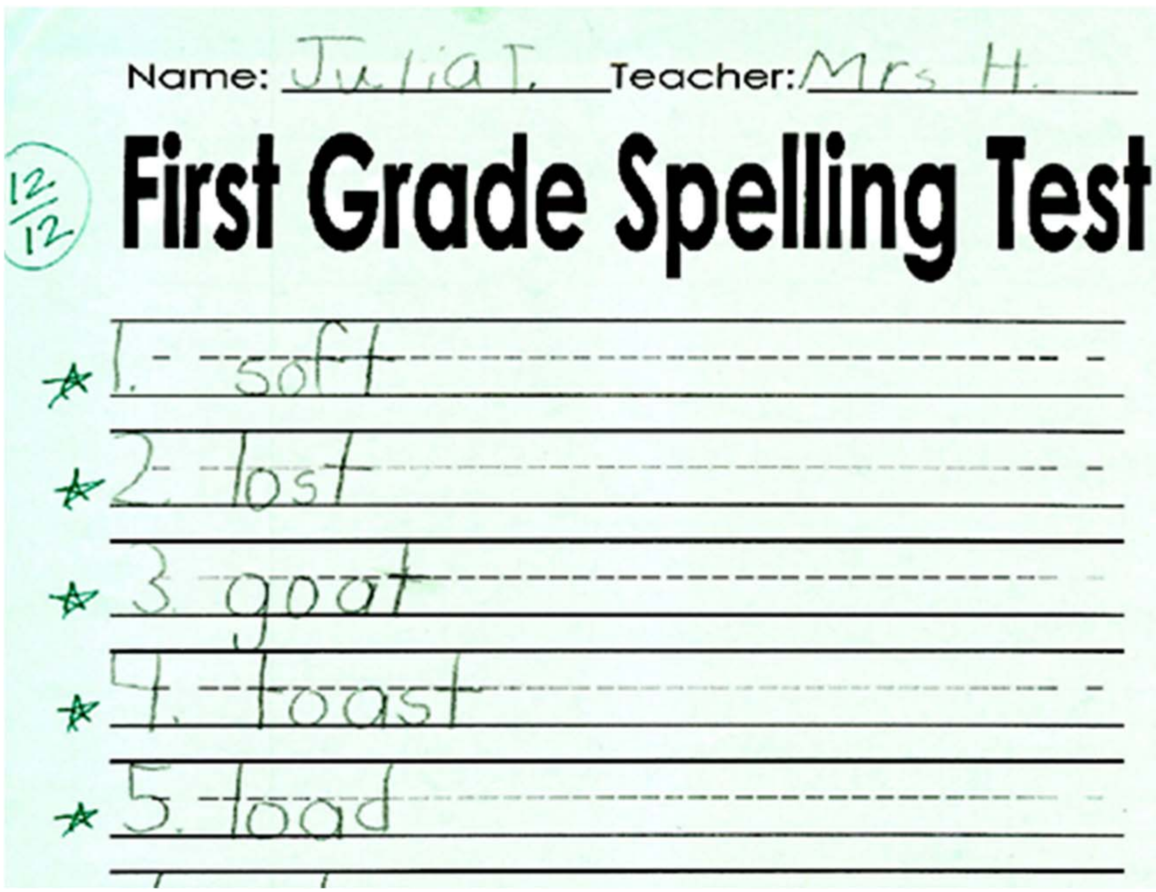
```
2. AG(v1 & v2 & ... & vk -> AX !(R))
```

Forces production of a counterexample for each R.

This is just right!

NIST
National Institute of Standards and Technology

More testing Examples



Buffer Overflows

- Empirical data from the National Vulnerability Database
  - Investigated > 3,000 denial-of-service vulnerabilities reported in the NIST NVD for period of 10/06 – 3/07
  - Vulnerabilities triggered by:
    - Single variable – 94.7%
      example: *Heap-based buffer overflow in the SFTP protocol handler for Panic Transmit … allows remote attackers to execute arbitrary code via a long ftps:// URL.*
    - 2-way interaction – 4.9%
      example: *single character search string in conjunction with a single character replacement string, which causes an "off by one overflow"*
    - 3-way interaction – 0.4%
      example: *Directory traversal vulnerability when register_globals is enabled and magic_quotes is disabled and .. (dot dot) in the page parameter*

Example: Finding Buffer Overflows

```
1.    if (strcmp(conn[sid].dat->in_RequestMethod, "POST")==0) {
2.            if (conn[sid].dat->in_ContentLength<MAX_POSTSIZE) {
              ......
3.    conn[sid].PostData=calloc(conn[sid].dat->in_ContentLength+1024,
sizeof(char));
                    ......
4.           pPostData=conn[sid].PostData;
5.           do {
6.                   rc=recv(conn[sid].socket, pPostData, 1024, 0);
                       ......
7.                   pPostData+=rc;
8.                   x+=rc;
9.           } while ((rc==1024)||(x<conn[sid].dat->in_ContentLength));
10.   conn[sid].PostData[conn[sid].dat->in_ContentLength]='\0';
11.    }
```

Interaction: request-method=”POST”, content-length = -1000, data= a string > 24 bytes

```
if (strcmp(conn[sid].dat->in_RequestMethod, "POST")==0) {
        if (conn[sid].dat->in_ContentLength<MAX_POSTSIZE) {
        ......
conn[sid].PostData=calloc(conn[sid].dat->in_ContentLength+1024, sizeof(char));
                ......
        pPostData=conn[sid].PostData;
        do {
                rc=recv(conn[sid].socket, pPostData, 1024, 0);
                  ......
                pPostData+=rc;
                x+=rc;
        } while ((rc==1024)||(x<conn[sid].dat->in_ContentLength));
conn[sid].PostData[conn[sid].dat->in_ContentLength]='\0';
 }
```

Interaction: request-method="POST", content-length = -1000, data= a string > 24 bytes

true branch

```
if (strcmp(conn[sid].dat->in_RequestMethod, "POST")==0) {
        if (conn[sid].dat->in_ContentLength<MAX_POSTSIZE) {
        ......
conn[sid].PostData=calloc(conn[sid].dat->in_ContentLength+1024,
sizeof(char));
                ......
        pPostData=conn[sid].PostData;
        do {
                rc=recv(conn[sid].socket, pPostData, 1024, 0);
                  ......
                pPostData+=rc;
                x+=rc;
        } while ((rc==1024)||(x<conn[sid].dat->in_ContentLength));
conn[sid].PostData[conn[sid].dat->in_ContentLength]='\0';
 }
```

Interaction: request-method=”POST”, content-length = -1000, data= a string > 24 bytes

```
if (strcmp(conn[sid].dat->in_RequestMethod, "POST")==0) {
        if (conn[sid].dat->in_ContentLength<MAX_POSTSIZE)
        ......
        conn[sid].PostData=calloc(conn[sid].dat->in_ContentLength+1024, sizeof(char));
                ......
        pPostData=conn[sid].PostData;
        do {
                rc=recv(conn[sid].socket, pPostData, 1024, 0);
                ......
                pPostData+=rc;
                x+=rc;
        } while ((rc==1024)||(x<conn[sid].dat->in_ContentLength));
    conn[sid].PostData[conn[sid].dat->in_ContentLength]='\0';
}
```

true branch

National Institute of Standards and Technology

Interaction: request-method="POST", content-length = -1000, data= a string > 24 bytes

```
if (strcmp(conn[sid].dat->in_RequestMethod, "POST")==0) {
        if (conn[sid].dat->in_ContentLength<MAX_POSTSIZE)
        ......
        conn[sid].PostData=calloc(conn[sid].dat->in_ContentLength+1024,
sizeof(char));
              ......
        pPostData=conn[sid].PostData;
        do {
                rc=recv(conn[sid].socket, pPostData, 1024, 0);
                ......
                pPostData+=rc;
                x+=rc;
        } while ((rc==1024)||(x<conn[sid].dat->in_ContentLength));
    conn[sid].PostData[conn[sid].dat->in_ContentLength]='\0';
 }
```

true branch

Allocate -1000 + 1024 bytes = 24 bytes

Interaction: request-method="POST", content-length = -1000, data= a string > 24 bytes

```
if (strcmp(conn[sid].dat->in_RequestMethod, "POST")==0) {
        if (conn[sid].dat->in_ContentLength<MAX_POSTSIZE)
        ......
        conn[sid].PostData=calloc(conn[sid].dat->in_ContentLength+1024,
sizeof(char));
              ......
        pPostData=conn[sid].PostData;
        do {
                rc=recv(conn[sid].socket, pPostData, 1024,
                ......
                pPostData+=rc;
                x+=rc;
        } while ((rc==1024)||(x<conn[sid].dat->in_ContentLength));
conn[sid].PostData[conn[sid].dat->in_ContentLength]='\0';
}
```

true branch

Allocate -1000 + 1024 bytes = 24 bytes

Boom!

Example: Modeling & Simulation

- "Simured" network simulator
  - Kernel of ~ 5,000 lines of C++ (not including GUI)
- Objective: detect configurations that can produce deadlock:
  - Prevent connectivity loss when changing network
  - Attacks that could lock up network
- Compare effectiveness of random vs. combinatorial inputs
- Deadlock combinations discovered
- Crashes in >6% of tests w/ valid values (Win32 version only)

NIST National Institute of Standards and Technology

Simulation Input Parameters

Parameter		Values
1	DIMENSIONS	1,2,4,6,8
2	NODOSDIM	2,4,6
3	NUMVIRT	1,2,3,8
4	NUMVIRTINJ	1,2,3,8
5	NUMVIRTEJE	1,2,3,8
6	LONBUFFER	1,2,4,6
7	NUMDIR	1,2
8	FORWARDING	0,1
9	PHYSICAL	true, false
10	ROUTING	0,1,2,3
11	DELFIFO	1,2,4,6
12	DELCROSS	1,2,4,6
13	DELCHANNEL	1,2,4,6
14	DELSWITCH	1,2,4,6

5x3x4x4x4x4x2x
2x2x4x4x4x4x4
= 31,457,280
configurations

Are any of them dangerous?

If so, how many?

Which ones?

National Institute of Standards and Technology

Network Deadlock Detection

Deadlocks Detected: combinatorial

t	Tests	500 pkts	1000 pkts	2000 pkts	4000 pkts	8000 pkts
2	28	0	0	0	0	0
3	161	2	3	2	3	3
4	752	14	14	14	14	14

Average Deadlocks Detected: random

t	Tests	500 pkts	1000 pkts	2000 pkts	4000 pkts	8000 pkts
2	28	0.63	0.25	0.75	0. 50	0. 75
3	161	3	3	3	3	3
4	752	10.13	11.75	10.38	13	13.25

Network Deadlock Detection

Detected 14 configurations that can cause deadlock:

$14/ 31{,}457{,}280 = 4.4 \times 10^{-7}$

Combinatorial testing found more deadlocks than random, including some that might never have been found with random testing

Why do this testing? Risks:

- accidental deadlock configuration: low
- deadlock config discovered by attacker: **much higher**

(because they are looking for it)

Coverage Measurement

Combinatorial Coverage Measurement

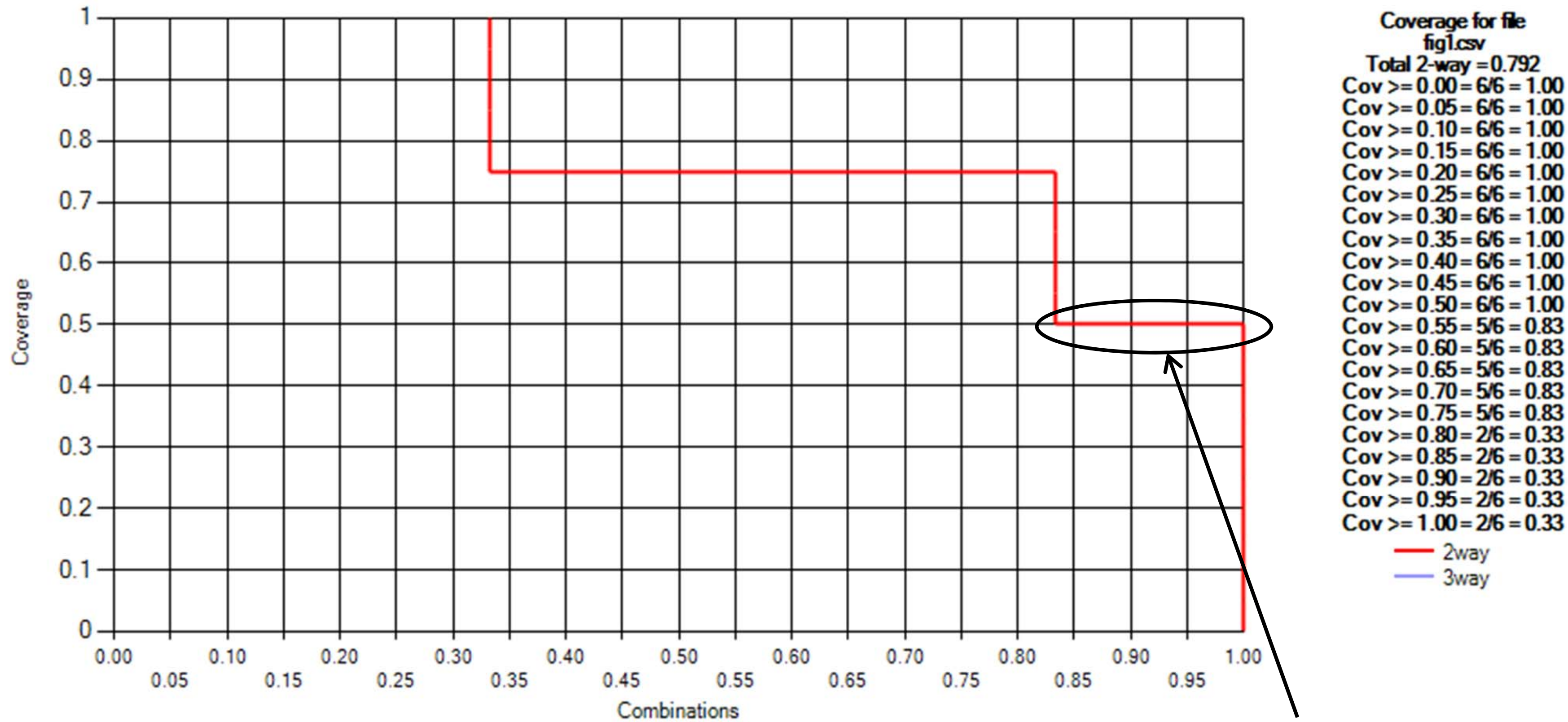
Tests	Variables			
	a	**b**	**c**	**d**
1	0	0	0	0
2	0	1	1	0
3	1	0	0	1
4	0	1	1	1
5	0	1	0	1
6	1	0	1	1
7	1	0	1	0
8	0	1	0	0

Variable pairs	Variable-value combinations covered	Coverage
ab	00, 01, 10	.75
ac	00, 01, 10	.75
ad	00, 01, 11	.75
bc	00, 11	.50
bd	00, 01, 10, 11	1.0
cd	00, 01, 10, 11	1.0

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

NIST
National Institute of Standards and Technology

Graphing Coverage Measurement

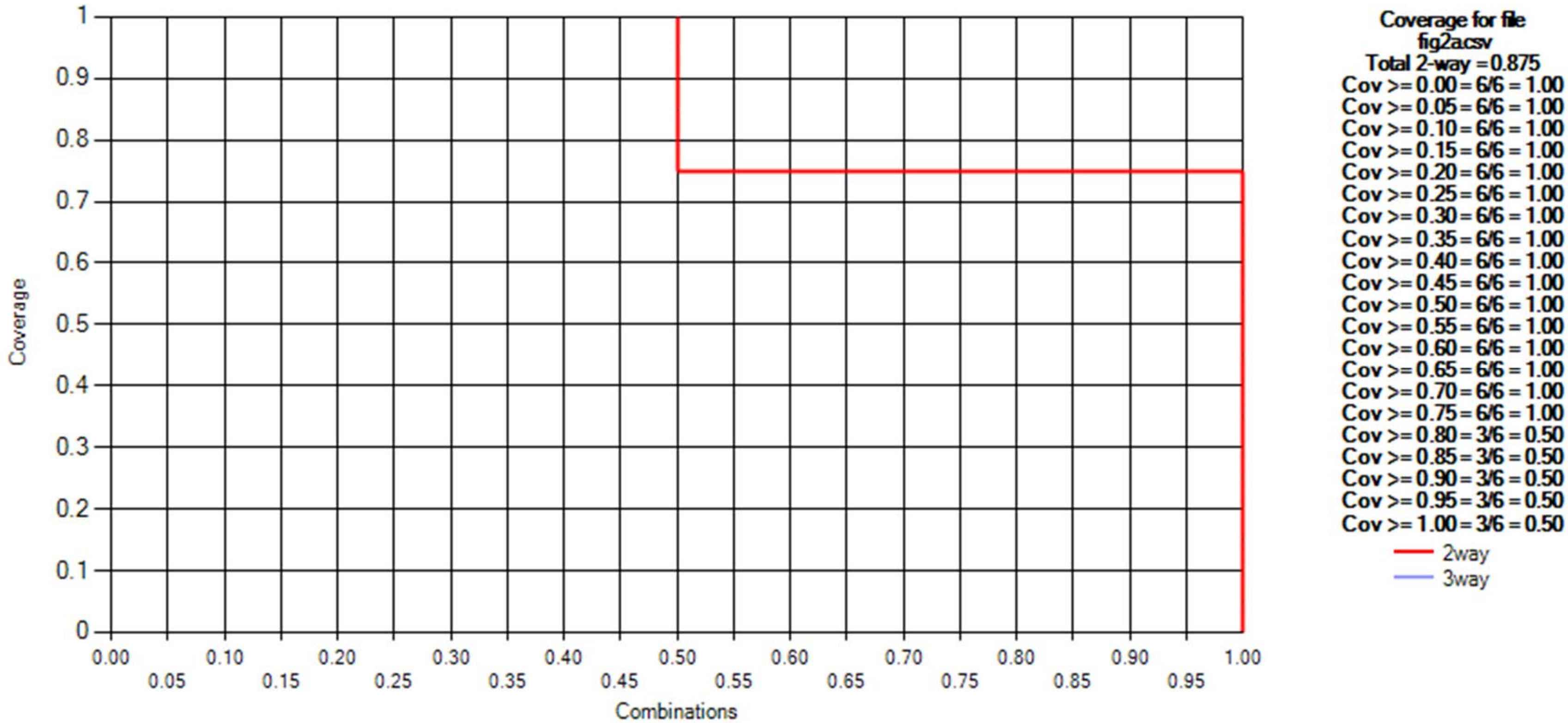


100% coverage of 33% of combinations
75% coverage of half of combinations
50% coverage of 16% of combinations

Bottom line:
All combinations
covered to at least 50%

NIST
National Institute of
Standards and Technology

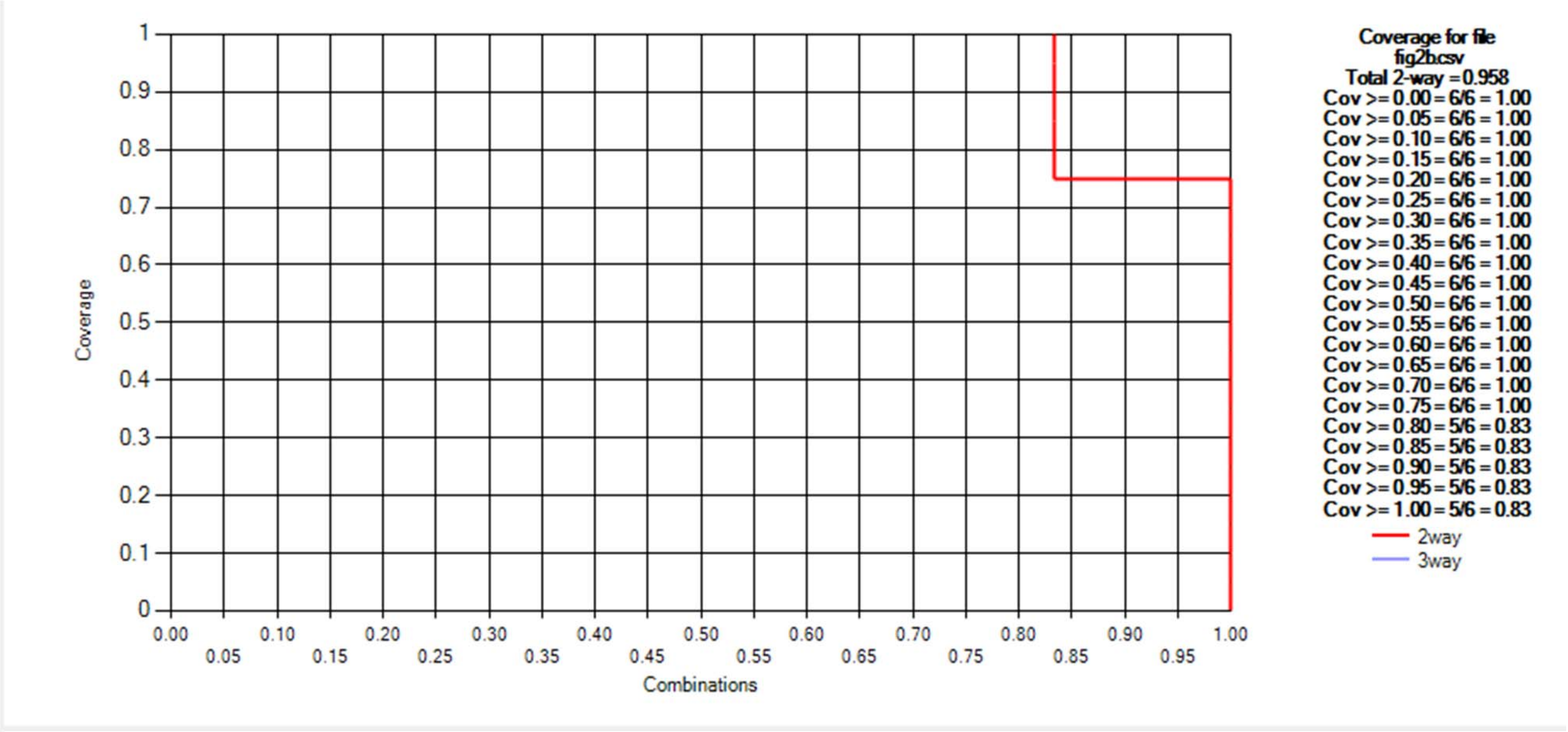
Adding a test



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

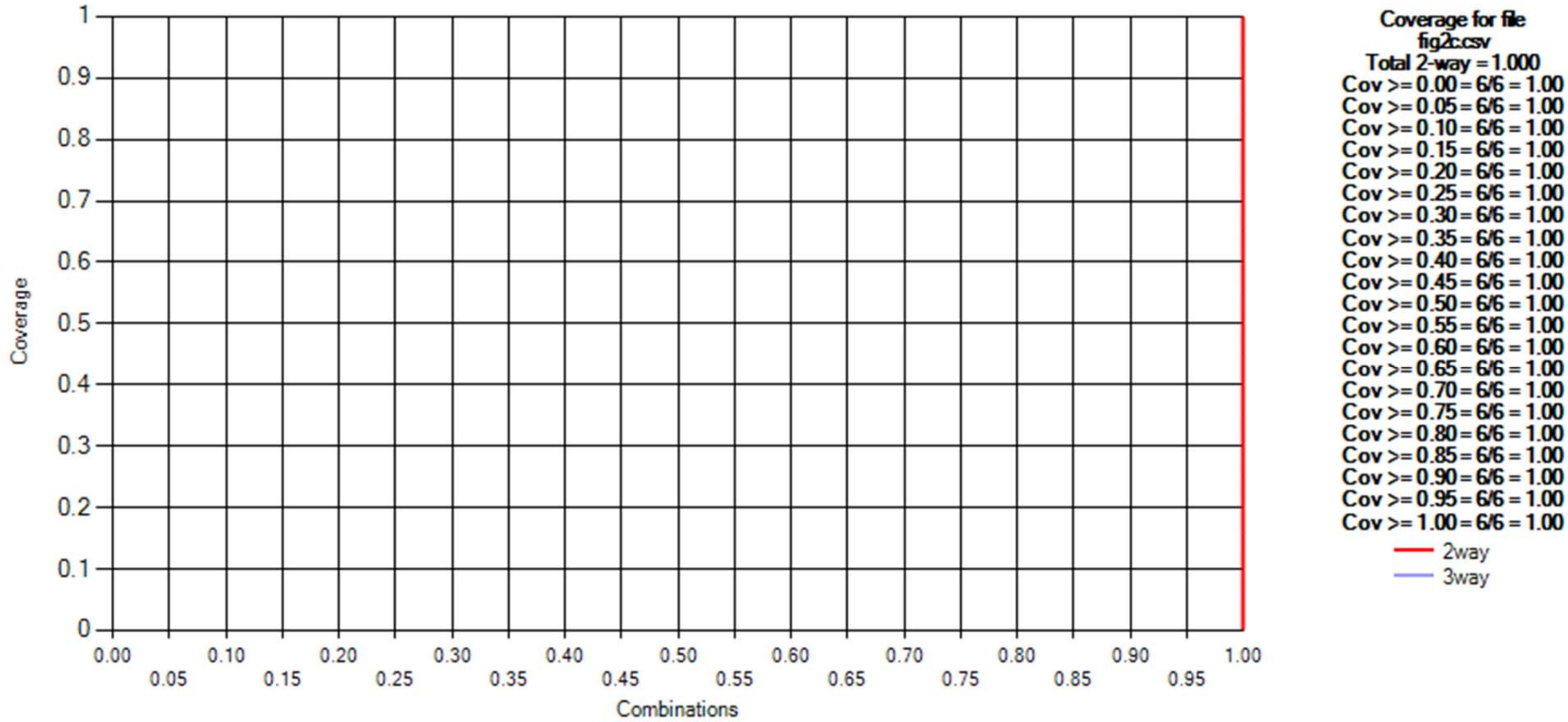
NIST
National Institute of Standards and Technology

Adding another test



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

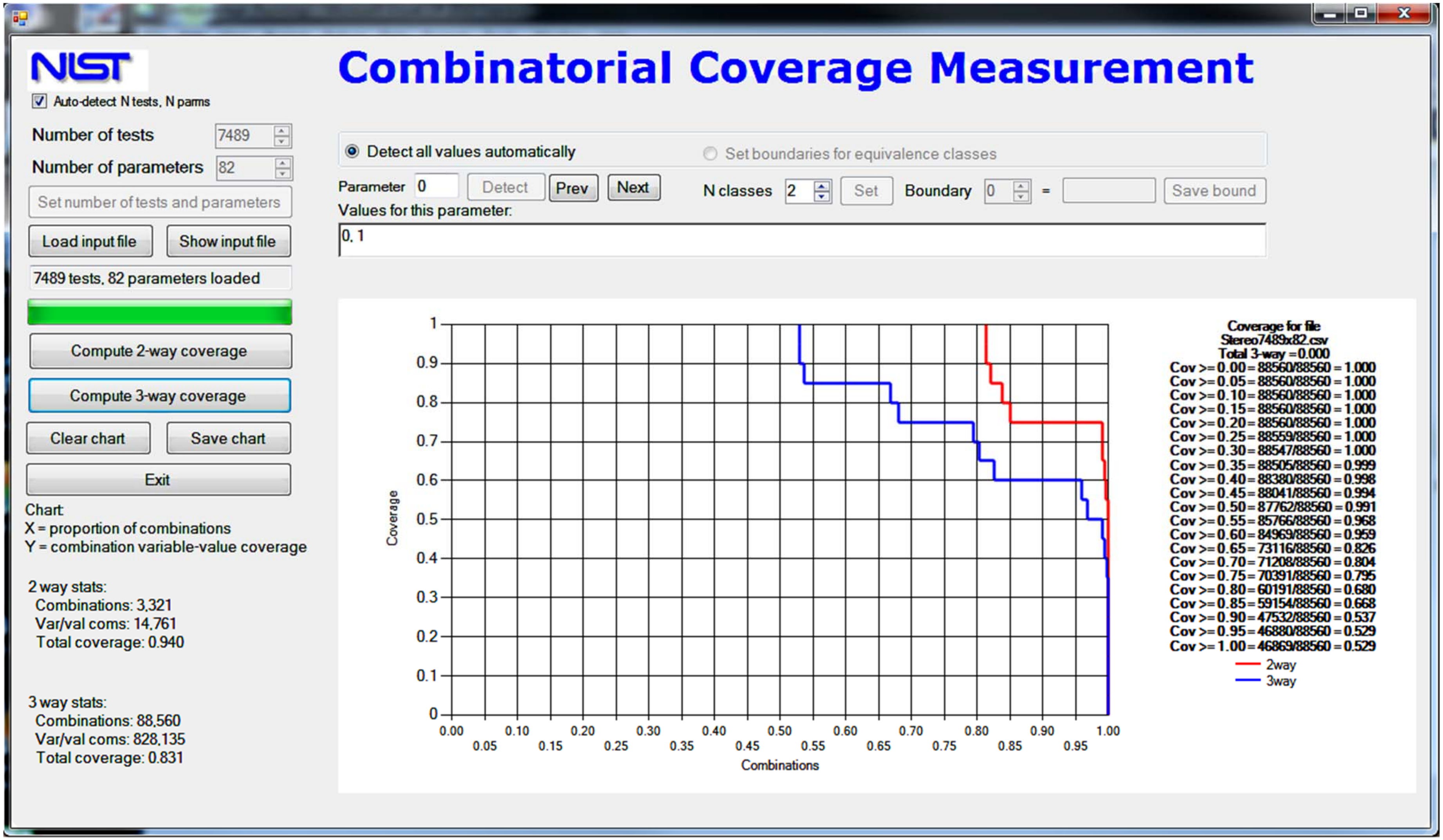
Additional test completes coverage



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

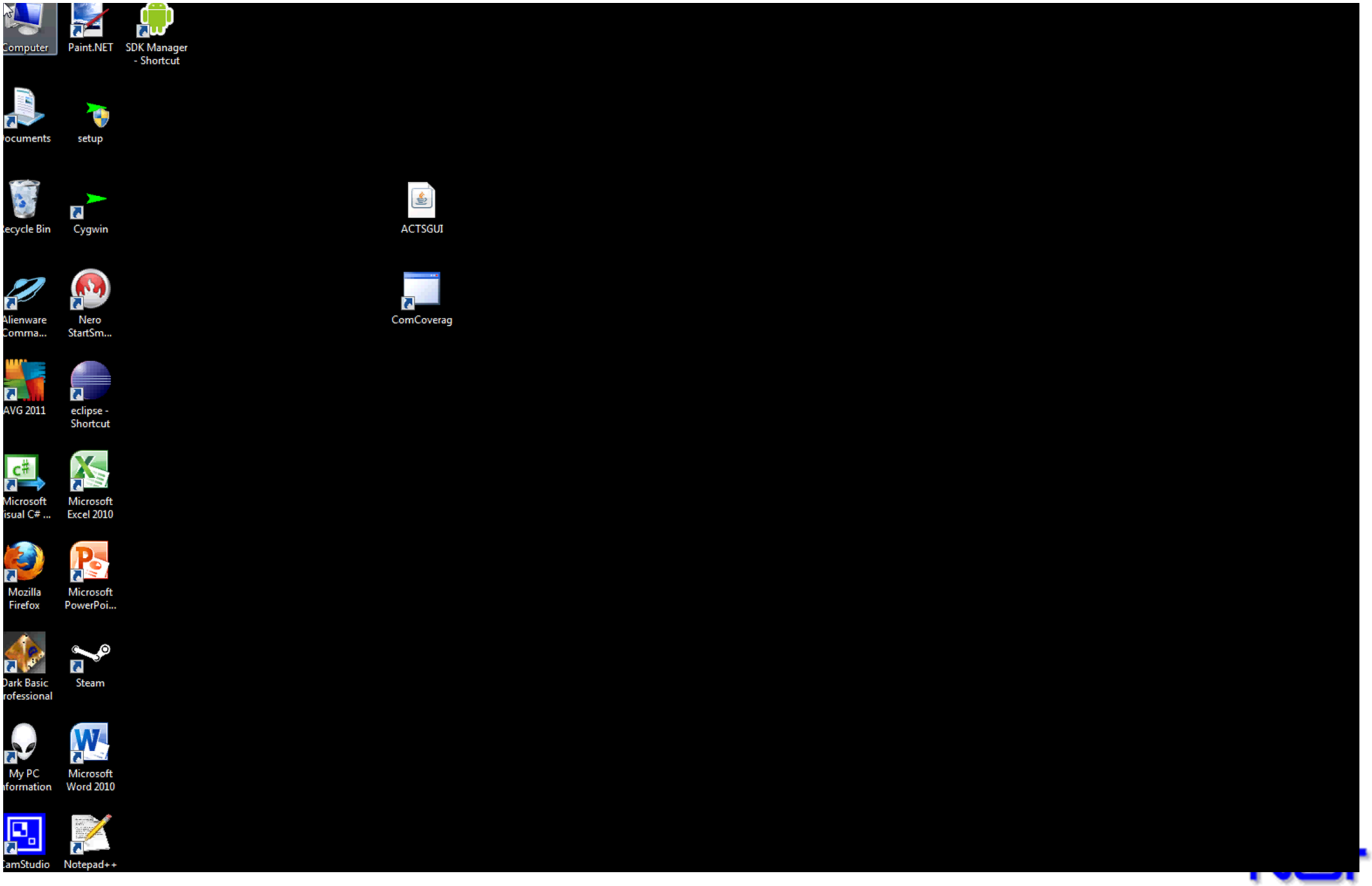
NIST
National Institute of Standards and Technology

Combinatorial Coverage Measurement



National Institute of Standards and Technology

Using Coverage Measurement



National Institute of Standards and Technology

Combinatorial Sequences for Testing

Combinatorial Sequence Testing

- We want to see if a system works correctly regardless of the order of events. How can this be done efficiently?
- Failure reports often say something like:
  'failure occurred when A started if B is not already connected'.
- Can we produce compact tests such that all t-way sequences covered (possibly with interleaving events)?

Event	Description
a	connect flow meter
b	connect pressure gauge
c	connect satellite link
d	connect pressure readout
e	start comm link
f	boot system

NIST
National Institute of Standards and Technology

Sequence Covering Array

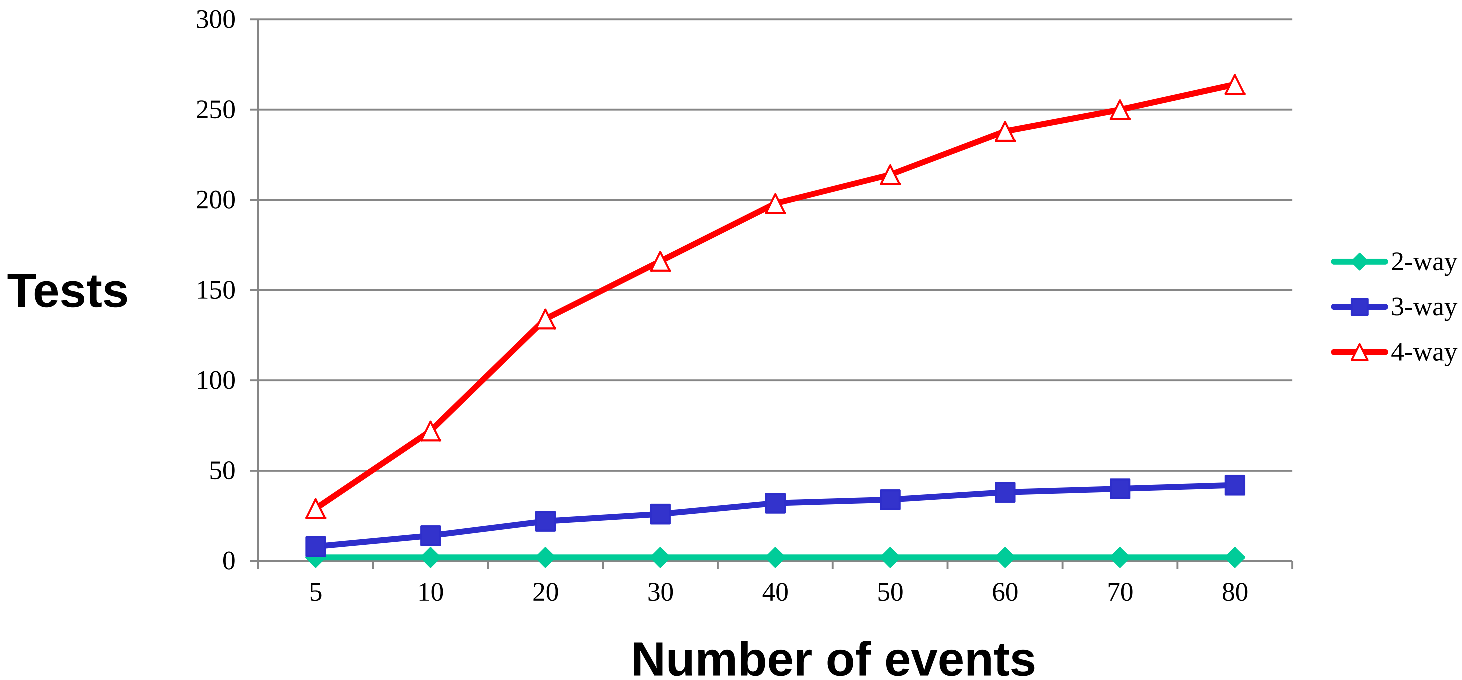
- With 6 events, all sequences = 6! = 720 tests
- Only 10 tests needed for all 3-way sequences, results even better for larger numbers of events
- Example: .*c.*f.*b.* covered. Any such 3-way seq covered.

Test	Sequence					
1	a	b	c	d	e	f
2	f	e	d	c	b	a
3	d	e	f	a	b	c
4	c	b	a	f	e	d
5	b	f	a	d	c	e
6	e	c	d	a	f	b
7	a	e	f	c	b	d
8	d	b	c	f	e	a
9	c	e	a	d	b	f
10	f	b	d	a	e	c

NIST National Institute of Standards and Technology

Sequence Covering Array Properties

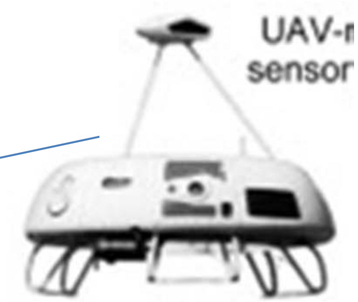
- 2-way sequences require only 2 tests
  (write events in any order, then reverse)
- For > 2-way, number of tests grows with log *n*, for *n* events
- Simple greedy algorithm produces compact test set



National Institute of Standards and Technology

Example: Laptop application

Problem: connect many peripherals, order of connection may affect application



Connection Sequences

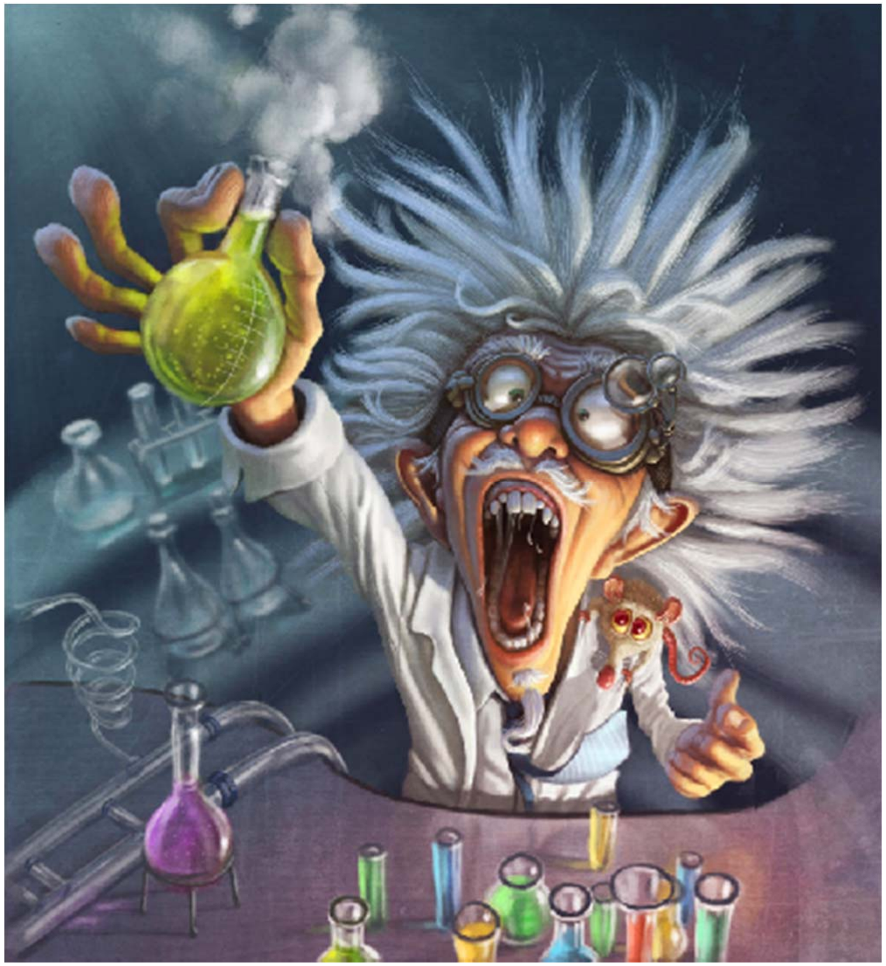
1	Boot	P-1 (USB-RIGHT)	P-2 (USB-BACK)	P-3 (USB-LEFT)	P-4	P-5	App	Scan
2	Boot	App	Scan	P-5	P-4	P-3 (USB-RIGHT)	P-2 (USB-BACK)	P-1 (USB-LEFT)
3	Boot	P-3 (USB-RIGHT)	P-2 (USB-LEFT)	P-1 (USB-BACK)	App	Scan	P-5	P-4
	etc...							

3-way sequence covering of connection events

Results

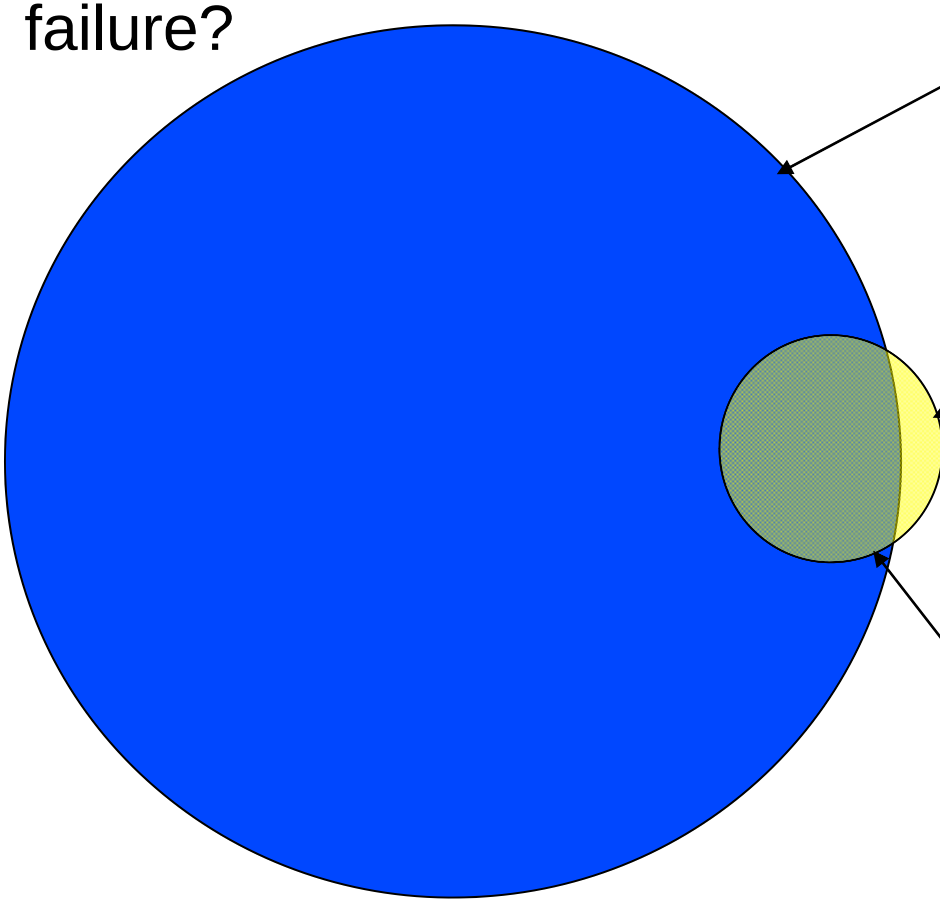
- Tested peripheral connection for 3-way sequences
- Some faults detected that would not have been found with 2-way sequence testing; may not have been found with random
  - Example:
  - If P2-P1-P3 sequence triggers a failure, then a full 2-way sequence covering array would not have found it
    (because 1-2-3-4-5-6-7 and 7-6-5-4-3-2-1 is a 2-way sequence covering array)

Research Questions



Fault location

Given: a set of tests that the SUT fails, which combinations of variables/values triggered the failure?

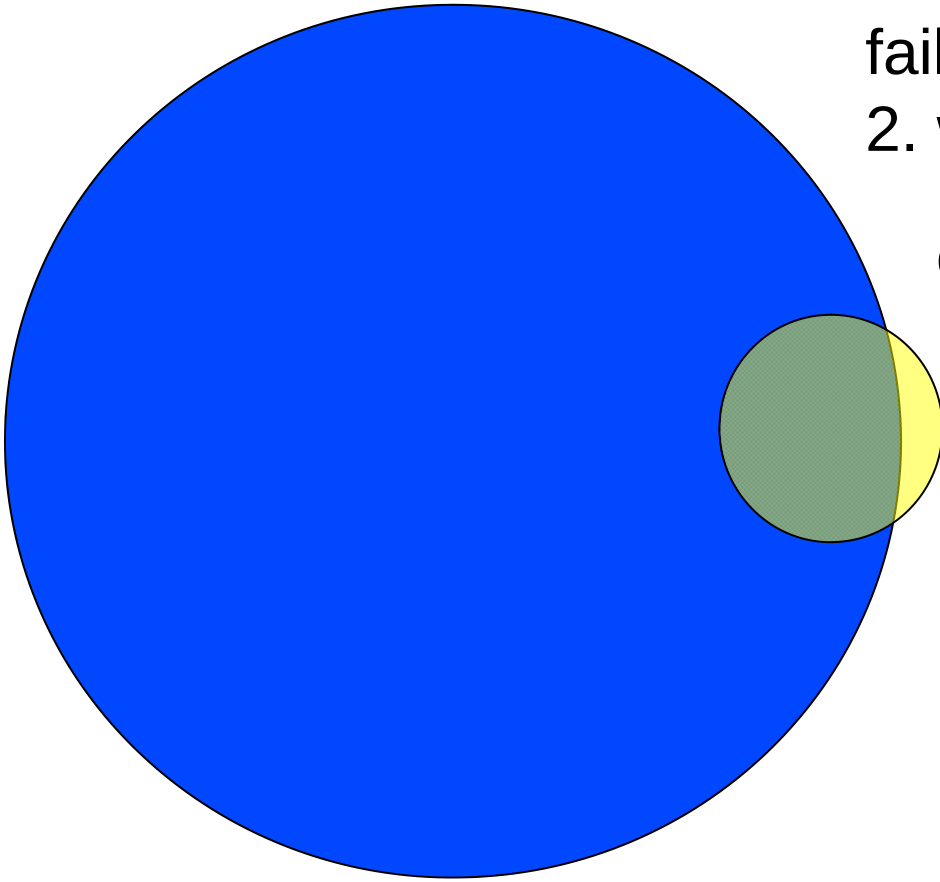


variable/value combinations in passing tests

These are the ones we want

variable/value combinations in failing tests

Fault location – what's the problem?



If they're in failing set but not in passing set:

1. which ones triggered the failure?
2. which ones don't matter?

out of $v^t\binom{n}{t}$ combinations

Example:
30 variables, 5 values each
= 445,331,250
5-way combinations

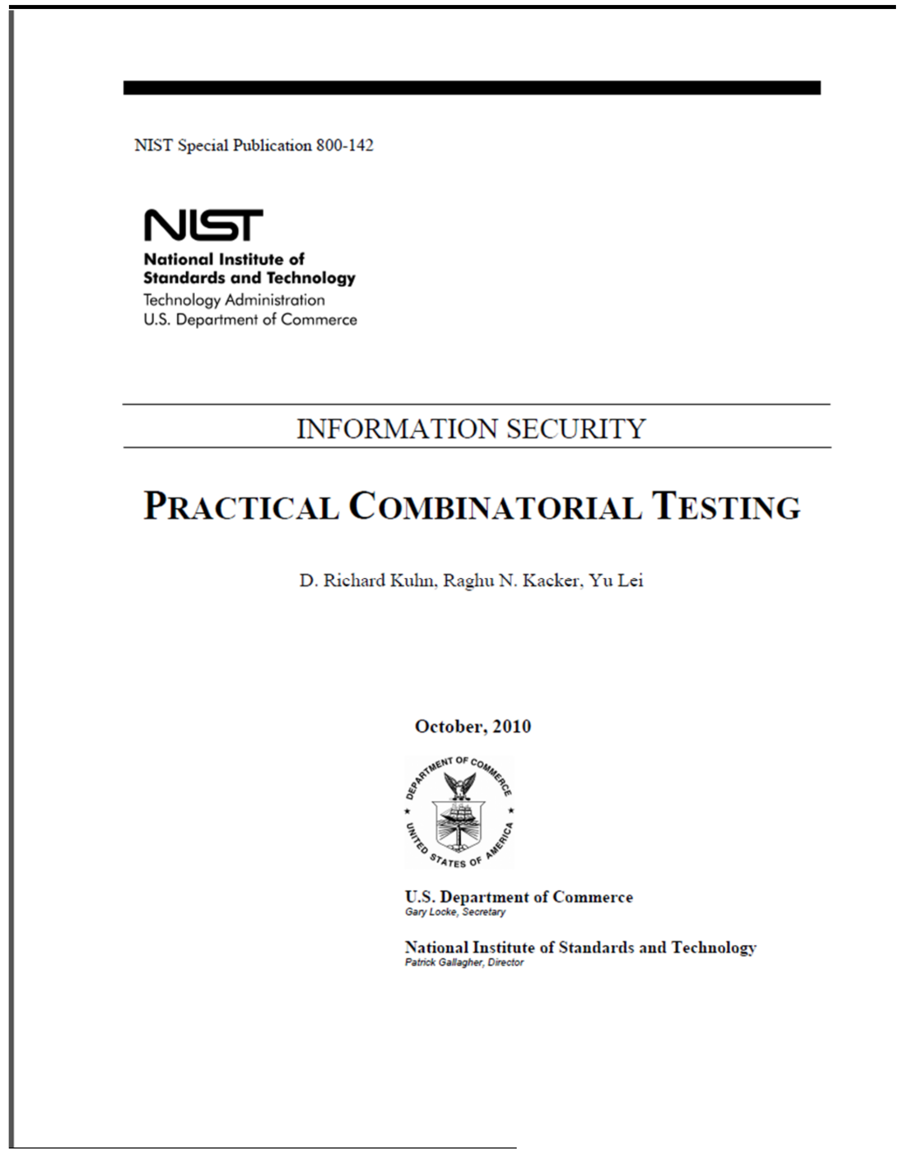
142,506 combinations in each test

National Institute of Standards and Technology

Integrating into Testing Program

- Test suite development
  - Generate covering arrays for tests OR
  - Measure coverage of existing tests and supplement

- Training
  - Testing textbooks – Mathur, Ammann & Offutt,
  - Combinatorial testing "textbook" on ACTS site →
  - User manuals
  - Worked examples


NIST Special Publication 800-142

NIST
National Institute of Standards and Technology
Technology Administration
U.S. Department of Commerce

INFORMATION SECURITY

PRACTICAL COMBINATORIAL TESTING

D. Richard Kuhn, Raghu N. Kacker, Yu Lei

October, 2010

U.S. Department of Commerce
Gary Locke, Secretary

National Institute of Standards and Technology
Patrick Gallagher, Director

Industrial Usage Reports

- Work with US Air Force on sequence covering arrays, submitted for publication
- World Wide Web Consortium DOM Level 3 events conformance test suite
- Cooperative Research & Development Agreement with Lockheed Martin Aerospace - report to be released 3rd or 4th quarter 2011

NIST
National Institute of Standards and Technology

Technology Transfer

- Tools obtained by 700+ organizations; NIST “textbook” on combinatorial testing downloaded 9,000+ times since Oct. 2010
- Collaborations: USAF 46th Test Wing, Lockheed Martin, George Mason Univ., Univ. of Maryland Baltimore County, Johns Hopkins Univ. Applied Physics Lab, Carnegie Mellon Univ.
- We are always interested in working with others!

Please contact us if you would like more information.

Rick Kuhn
kuhn@nist.gov

Raghu Kacker
raghu.kacker@nist.gov

http://csrc.nist.gov/acts

(Or just search “combinatorial testing”. We’re #1!)

NIST
National Institute of Standards and Technology