Advanced Research Center for Software Testing and Quality Assurance

Combinatorial Design-based Test Generation

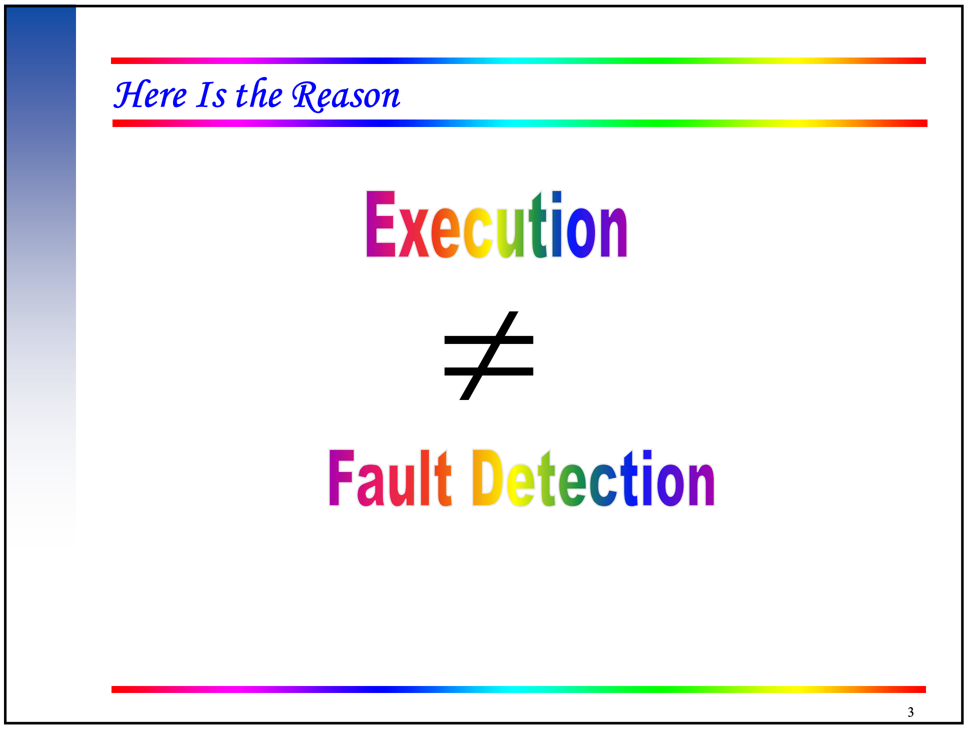
W. Eric Wong
Department of Computer Science
The University of Texas at Dallas
http://www.utdallas.edu/~ewong

stqa

1

Speaker Biographical Sketch

- Professor & Founding Director
  Advanced Research for Software Testing & Quality Assurance
  Department of Computer Science
  University of Texas at Dallas
- Guest Researcher
  Computer Security Division
  National Institute of Standards and Technology (NIST)
- Editor-in-Chief, IEEE Transactions on Reliability
- Engineer of the Year, 2014, IEEE Reliability Society
- Vice President, IEEE Reliability Society (2012−2015)
- Secretary, ACM SIGAPP (Special Interest Group on Applied Computing) 2009 - 2013
- Steering Committee Chair of the QRS conference (*IEEE International Conference on Software Quality, Reliability and Security*) & IWPD (*IEEE International Workshop on Program Debugging*)



Here Is the Reason

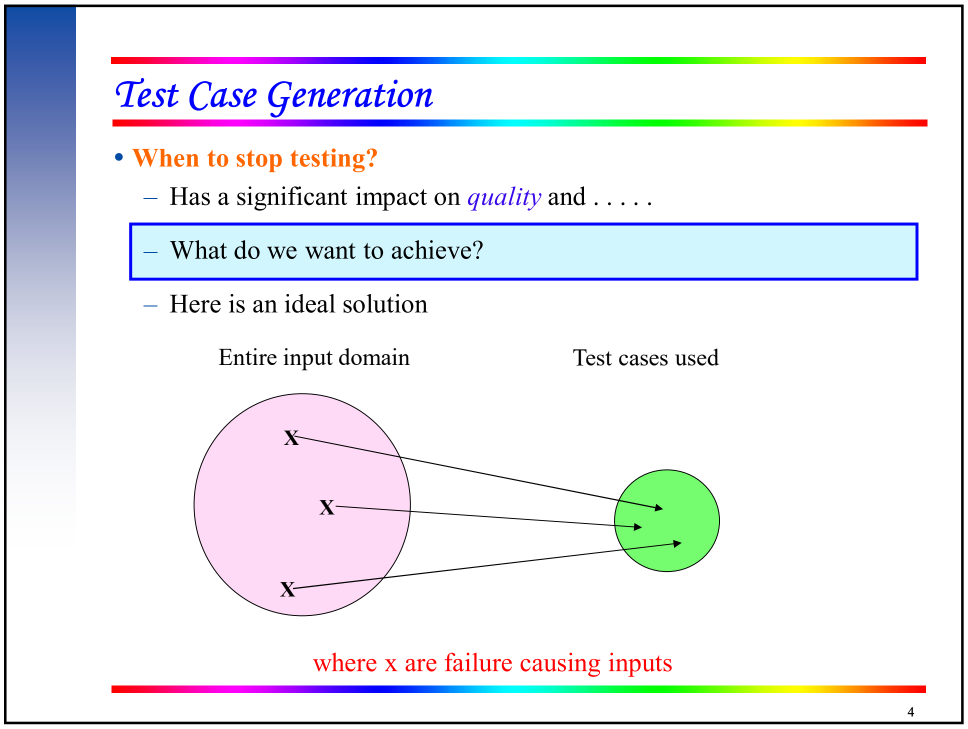
Execution

$\neq$

Fault Detection

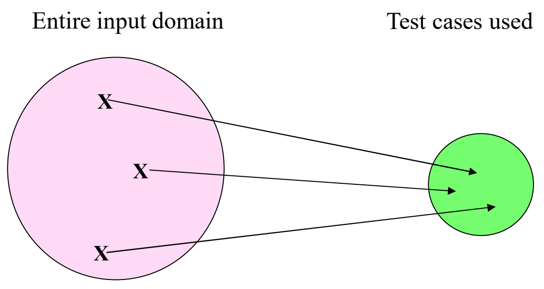
3

3



Test Case Generation

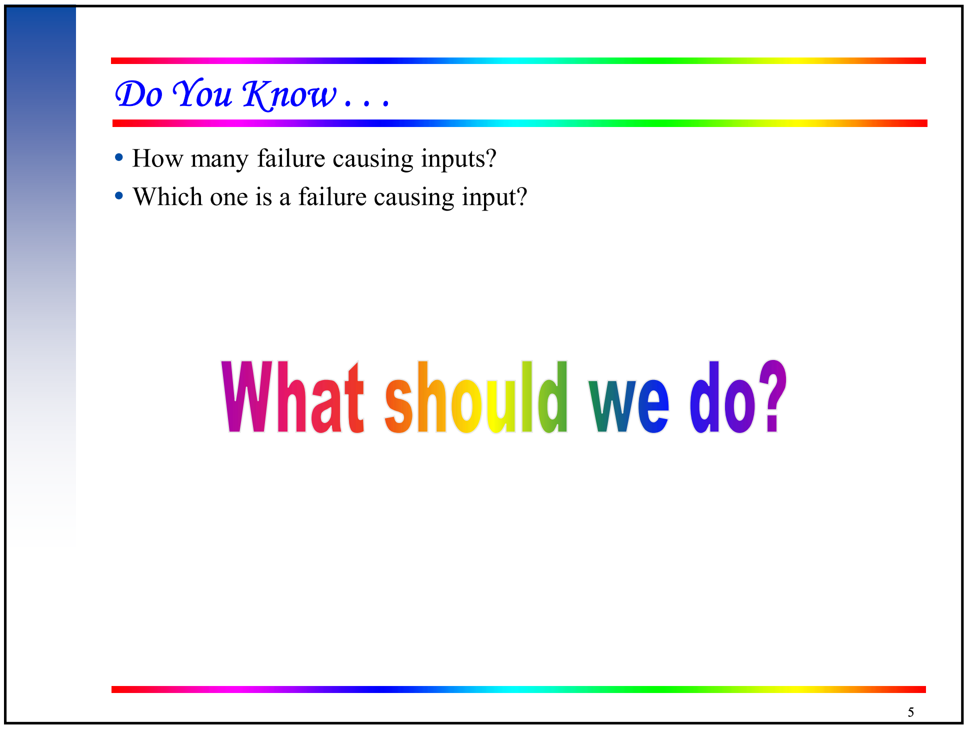
- **When to stop testing?**
  - Has a significant impact on *quality* and . . . . .
  - What do we want to achieve?
  - Here is an ideal solution



where x are failure causing inputs

4

4



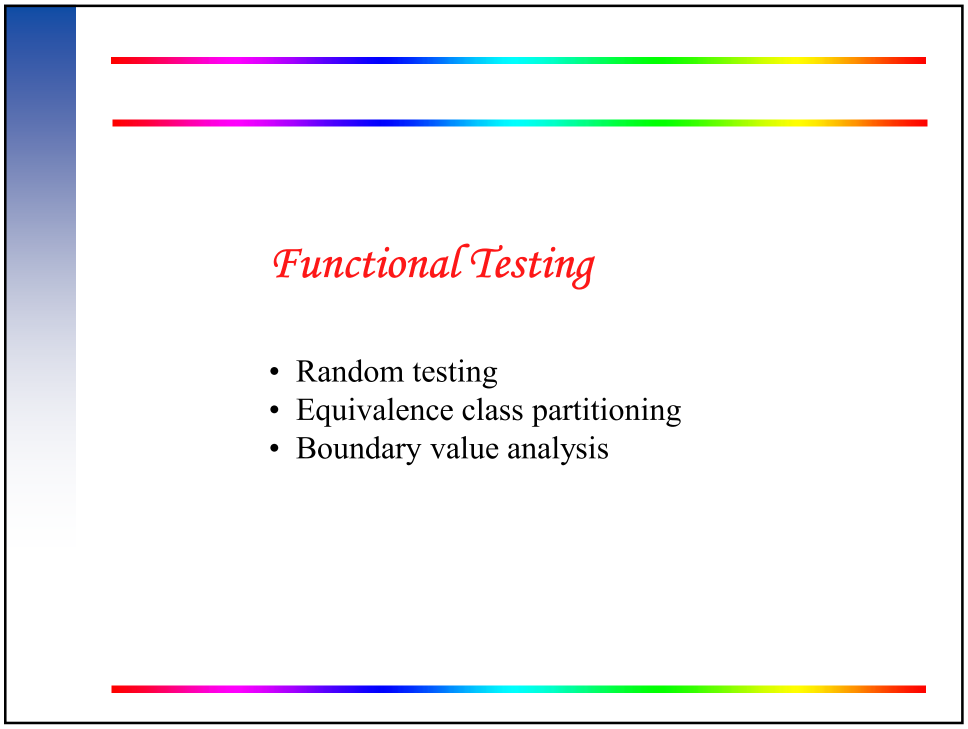
Do You Know . . .

- How many failure causing inputs?
- Which one is a failure causing input?

What should we do?

5

5



Functional Testing

- Random testing
- Equivalence class partitioning
- Boundary value analysis

6

Combinatorial Testing

7

Challenges to be Overcome

- Testing individual component is not enough
- We should also focus on interactions on different components

Many system components are tested individually, but often unexpected interactions between components cause failures.

How to do it

Combinatorial Testing

8

8

Combinatorial Testing – What Is It ?

- CT is an effective test generation technique focuses on testing various combinations among different components of a system
  - Based on a classic mathematics topic – *Combinatorics* (concerning the study of finite or countable discrete structures)
    - N. L. Biggs, "The Root of Combinatorics," *Historia Mathematica*, 6(2): 109-136, May 1979
    - J. Riordan, "Introduction to Combinatorial Analysis," *Dover Publications*, September 1980

A Formal Engineering Method

Rather Than an Ad-Hoc Approach

Combinatorial Testing: Example I (1)

Input Parameters	Values			
Parameter 1	800	900		
Parameter 2	start	stop		
Parameter 3	on	off	idle	
Parameter 4	up	down	forward	backward

Number of exhaustive combinations among all parameters= 2×2×3×4 = 48 ← Too many

- Focus on all the combinations among some (e.g., 2) parameters

	Parameter 1	Parameter 2	Parameter 3	Parameter 4
t_1	800	Start	?	?
t_2	800	Stop	?	?
t_3	900	Start	?	?
t_4	900	Stop	?	?
t_5	?	?	on	Up
t_6	?	?	on	Down
t_7	?	?	on	Forward
t_8	?	?	on	Backward
⋮	⋮	⋮	⋮	⋮

We can reduce the number of test cases by choosing parameter values intelligently.

Combinatorial Testing: Example I (2)

- Number of pairwise combinations
  (all combinations between 2 parameters)
  - Parameter 1 and Parameter 2: 4 (2 × 2)
  - Parameter 1 and Parameter 3: 6 (2 × 3)
  - Parameter 1 and Parameter 4: 8 (2 × 4)
  - Parameter 2 and Parameter 3: 6 (2 × 3)
  - Parameter 2 and Parameter 4: 8 (2 × 4)
  - Parameter 3 and Parameter 4: 12 (3 × 4)

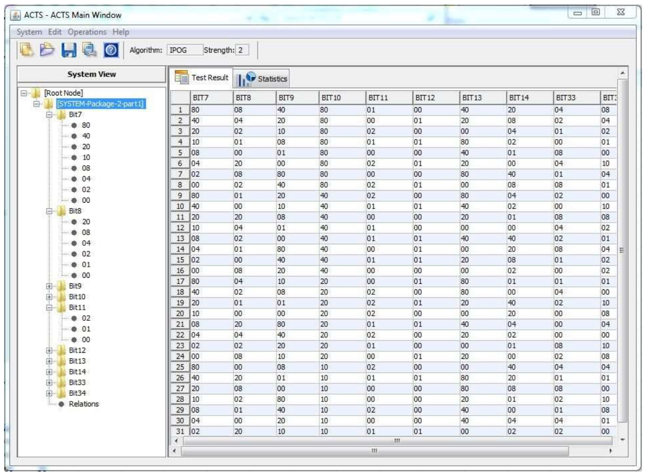
  - Do we really need 44 (4 + 6 + 8 + 6 + 8 +12) test cases ?

11

11

Do We Have Any Tool Support ?

- ACTS is one of the tools which support CT



Algorithm: IPOG Strength: 2

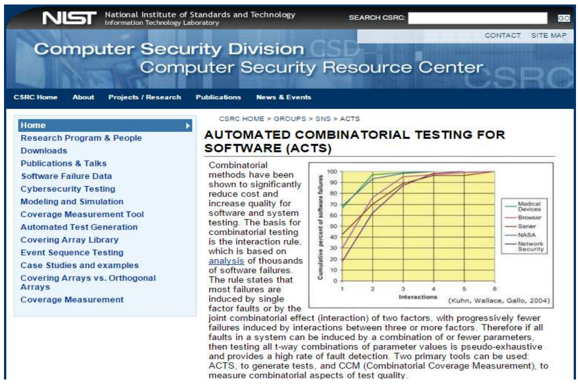
	BIT7	BIT8	BIT9	BIT10	BIT11	BIT12	BIT13	BIT14	BIT33	BIT
1	80	08	40	80	01	00	40	20	04	08
2	40	04	20	80	00	01	20	08	02	04
3	20	02	10	80	02	00	00	04	01	02
4	10	01	08	80	01	01	80	02	00	01
5	08	00	01	80	00	00	40	01	08	00
6	04	20	00	80	02	01	20	00	04	10
7	02	08	80	80	00	00	80	40	01	04
8	00	02	40	80	02	01	00	08	08	01
9	80	01	20	40	02	00	80	04	02	00
10	40	00	10	40	01	01	40	02	00	10
11	20	20	08	40	00	00	20	01	08	08
12	10	04	01	40	01	00	00	00	04	02
13	08	02	00	40	01	01	40	40	02	01
14	04	01	80	40	00	01	00	20	08	04
15	02	00	40	40	01	01	20	08	01	02
16	00	08	20	40	00	00	00	02	00	02
17	80	04	10	20	00	01	80	01	01	01
18	40	02	08	20	02	00	80	00	04	00
19	20	01	01	20	02	01	20	40	02	10
20	10	00	00	20	02	00	00	20	00	08
21	08	20	80	20	01	01	40	04	00	04
22	04	04	40	20	02	00	20	02	00	00
23	02	02	20	20	01	00	00	01	08	10
24	00	08	10	20	00	01	20	00	02	08
25	80	00	08	10	02	00	00	40	04	04
26	40	20	01	10	01	01	80	20	01	01
27	20	08	00	10	00	00	80	08	08	00
28	10	02	80	10	00	00	20	01	02	10
29	08	01	40	10	02	00	40	00	01	08
30	04	00	20	10	00	00	40	04	04	01
31	02	20	10	10	01	01	00	02	02	00

12

12

Combinatorial Testing at NIST

- NIST (National Institute of Standards and Technology)


NIST National Institute of Standards and Technology
Information Technology Laboratory
SEARCH CSRC:
GO
CONTACT
SITE MAP
Computer Security Division CSD
Computer Security Resource Center CSRC
CSRC Home
About
Projects / Research
Publications
News & Events
Home
Research Program & People
Downloads
Publications & Talks
Software Failure Data
Cybersecurity Testing
Modeling and Simulation
Coverage Measurement Tool
Automated Test Generation
Covering Array Library
Event Sequence Testing
Case Studies and examples
Covering Arrays vs. Orthogonal Arrays
Coverage Measurement
CSRC HOME > GROUPS > SNS > ACTS
Cumulative percent of software failures
Interactions
Medical Devices
Browser
Server
NASA
Network Security
(Kuhn, Wallace, Gallo, 2004)

AUTOMATED COMBINATORIAL TESTING FOR SOFTWARE (ACTS)

Combinatorial methods have been shown to significantly reduce cost and increase quality for software and system testing. The basis for combinatorial testing is the interaction rule, which is based on analysis of thousands of software failures. The rule states that most failures are induced by single factor faults or by the joint combinatorial effect (interaction) of two factors, with progressively fewer failures induced by interactions between three or more factors. Therefore if all faults in a system can be induced by a combination of or fewer parameters, then testing all t-way combinations of parameter values is pseudo-exhaustive and provides a high rate of fault detection. Two primary tools can be used: ACTS, to generate tests, and CCM (Combinatorial Coverage Measurement), to measure combinatorial aspects of test quality.

- D. R. Kuhn, D. R. Wallace, A. M. Gallo, "Software Fault Interactions and Implications for Software Testing," *IEEE Transactions on Software Engineering*, 30(6):418-421, June 2004
- J. D. Hagar, T. L. Wissink, and D. R. Kuhn, "Introducing Combinatorial Testing in a Large Organization," IEEE Computer, 48(4): 64-72, April 2015

13

13

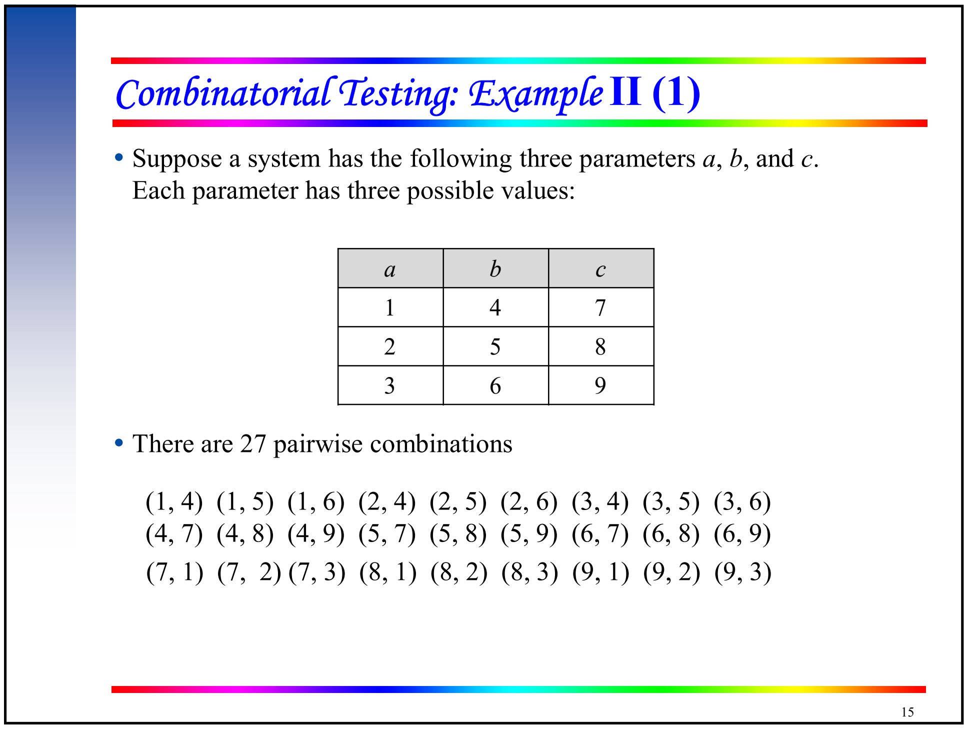
Combinatorial Testing: Example I (3)

- Using ACTS, we only need 12 test cases to cover all pairwise combinations

Test Case	p1	p2	p3	p4
1	900	stop	on	up
2	800	start	off	up
3	900	start	idle	up
4	800	stop	on	down
5	900	start	off	down
6	800	stop	idle	down
7	900	start	on	forward
8	800	stop	off	forward
9	900	stop	idle	forward
10	900	start	on	backward
11	800	stop	off	backward
12	800	stop	idle	backward

14

14



Combinatorial Testing: Example II (1)

- Suppose a system has the following three parameters *a*, *b*, and *c*. Each parameter has three possible values:

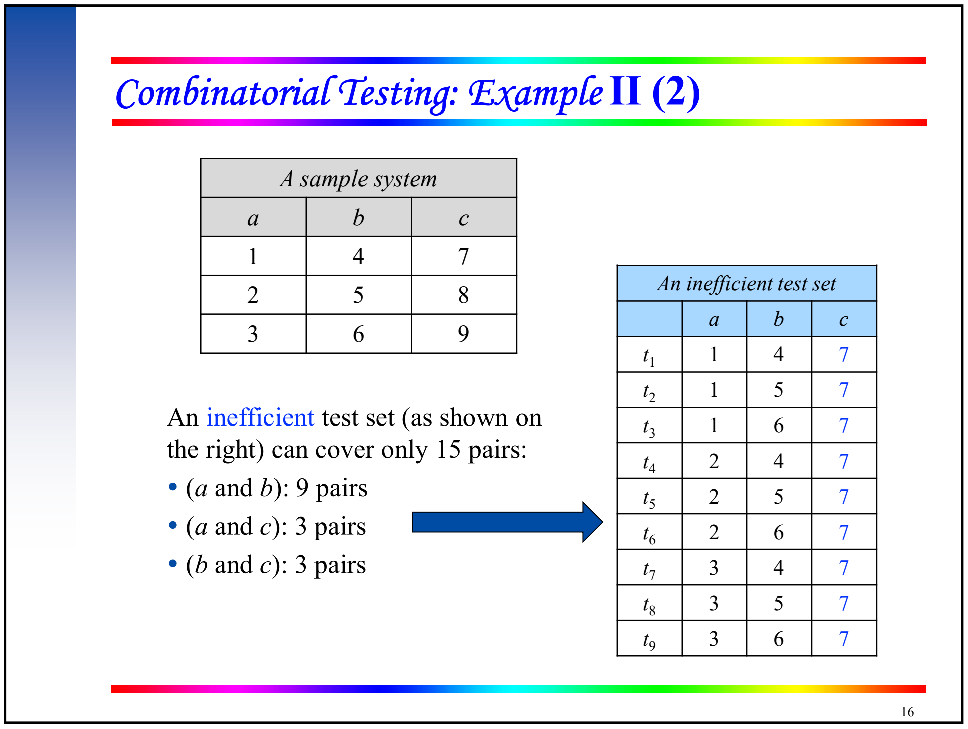
a	*b*	*c*
1	4	7
2	5	8
3	6	9

- There are 27 pairwise combinations

(1, 4) (1, 5) (1, 6) (2, 4) (2, 5) (2, 6) (3, 4) (3, 5) (3, 6)
(4, 7) (4, 8) (4, 9) (5, 7) (5, 8) (5, 9) (6, 7) (6, 8) (6, 9)
(7, 1) (7, 2) (7, 3) (8, 1) (8, 2) (8, 3) (9, 1) (9, 2) (9, 3)

15

15



Combinatorial Testing: Example II (2)

A sample system		
a	*b*	*c*
1	4	7
2	5	8
3	6	9

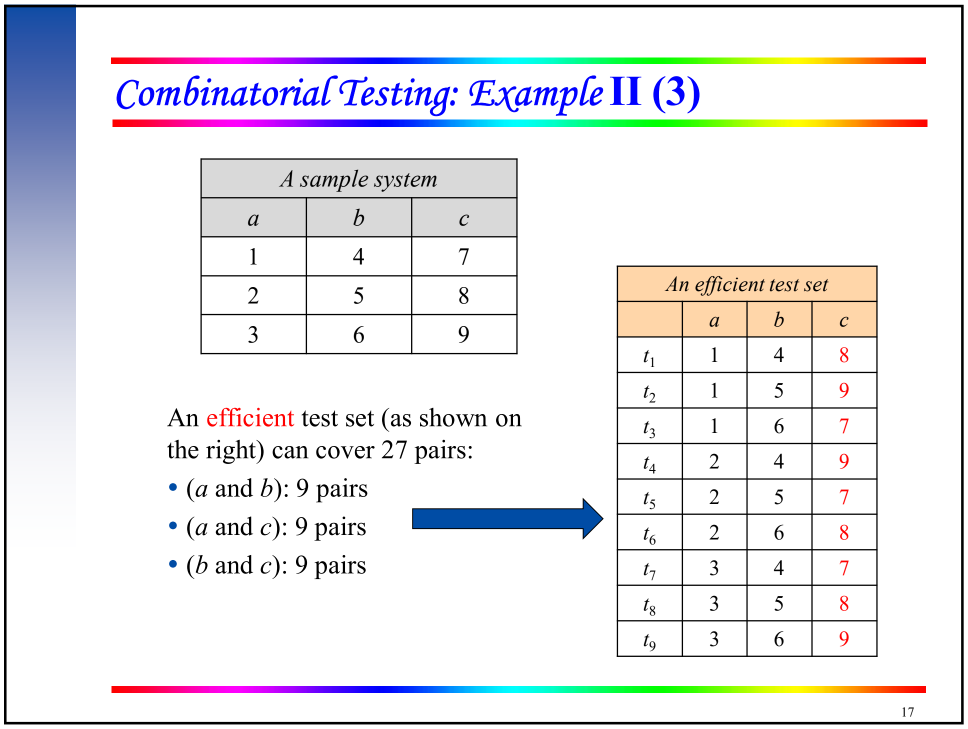
An inefficient test set (as shown on the right) can cover only 15 pairs:

- (*a* and *b*): 9 pairs
- (*a* and *c*): 3 pairs
- (*b* and *c*): 3 pairs

An inefficient test set			
	a	*b*	*c*
t_1	1	4	7
t_2	1	5	7
t_3	1	6	7
t_4	2	4	7
t_5	2	5	7
t_6	2	6	7
t_7	3	4	7
t_8	3	5	7
t_9	3	6	7

16

16



Combinatorial Testing: Example II (3)

A sample system		
a	*b*	*c*
1	4	7
2	5	8
3	6	9

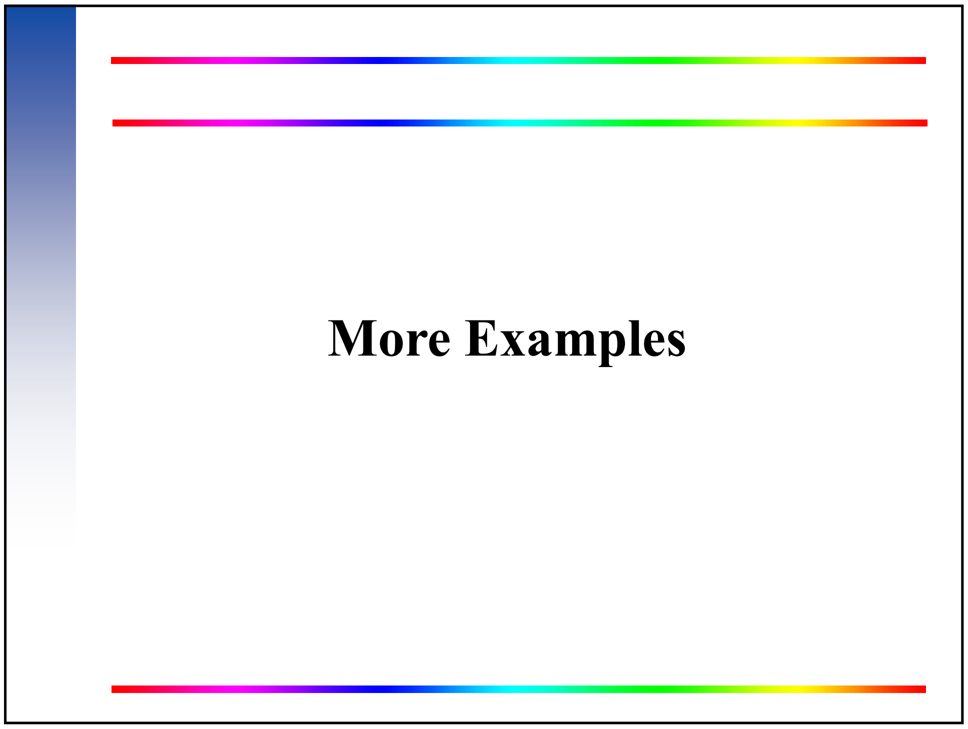
An efficient test set (as shown on the right) can cover 27 pairs:

- (*a* and *b*): 9 pairs
- (*a* and *c*): 9 pairs
- (*b* and *c*): 9 pairs

An efficient test set			
	a	*b*	*c*
t_1	1	4	8
t_2	1	5	9
t_3	1	6	7
t_4	2	4	9
t_5	2	5	7
t_6	2	6	8
t_7	3	4	7
t_8	3	5	8
t_9	3	6	9

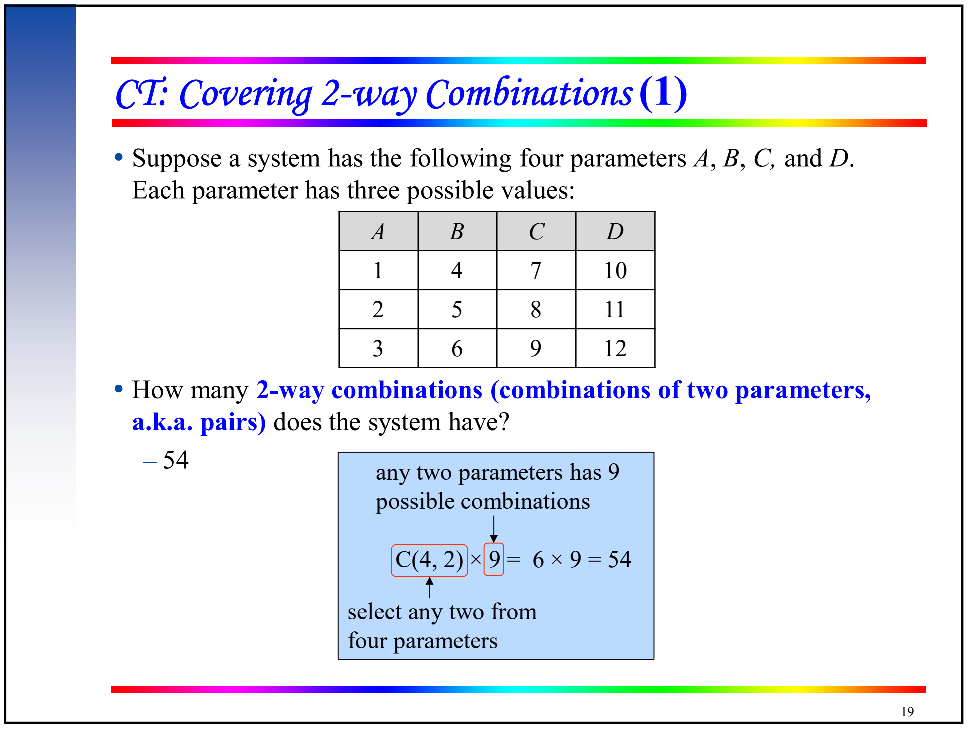
17

17



More Examples

18



CT: Covering 2-way Combinations (1)

- Suppose a system has the following four parameters *A*, *B*, *C,* and *D*. Each parameter has three possible values:

A	*B*	*C*	*D*
1	4	7	10
2	5	8	11
3	6	9	12

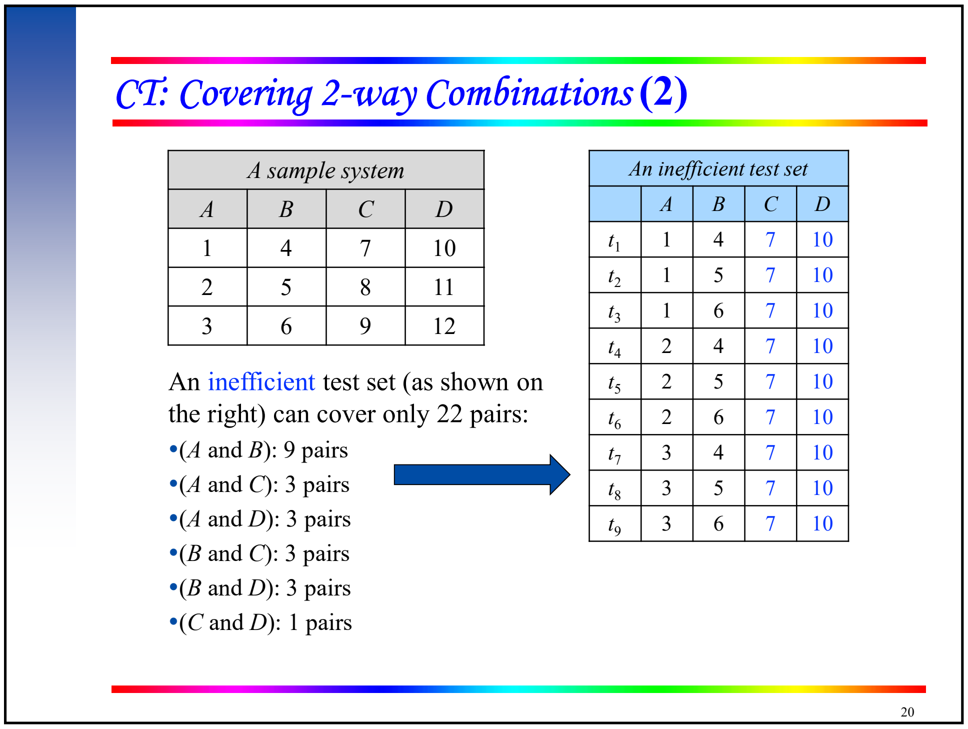
- How many **2-way combinations (combinations of two parameters, a.k.a. pairs)** does the system have?
  - 54

any two parameters has 9 possible combinations

$$\boxed{C(4,2)} \times \boxed{9} = 6 \times 9 = 54$$

select any two from four parameters

19



CT: Covering 2-way Combinations (2)

A sample system			
A	*B*	*C*	*D*
1	4	7	10
2	5	8	11
3	6	9	12

An inefficient test set (as shown on the right) can cover only 22 pairs:

- (*A* and *B*): 9 pairs
- (*A* and *C*): 3 pairs
- (*A* and *D*): 3 pairs
- (*B* and *C*): 3 pairs
- (*B* and *D*): 3 pairs
- (*C* and *D*): 1 pairs

An inefficient test set				
	A	*B*	*C*	*D*
t_1	1	4	7	10
t_2	1	5	7	10
t_3	1	6	7	10
t_4	2	4	7	10
t_5	2	5	7	10
t_6	2	6	7	10
t_7	3	4	7	10
t_8	3	5	7	10
t_9	3	6	7	10

20

CT: Covering 2-way Combinations (3)

A sample system			
A	*B*	*C*	*D*
1	4	7	10
2	5	8	11
3	6	9	12

An efficient test set				
	A	*B*	*C*	*D*
t_1	1	4	8	11
t_2	1	5	9	12
t_3	1	6	7	10
t_4	2	4	9	10
t_5	2	5	7	11
t_6	2	6	8	12
t_7	3	4	7	12
t_8	3	5	8	10
t_9	3	6	9	11

An efficient test set generated by ACTS (as shown on the right) can cover all 54 pairs!

- (*A* and *B*): 9 pairs
- (*A* and *C*): 9 pairs
- (*A* and *D*): 9 pairs
- (*B* and *C*): 9 pairs
- (*B* and *D*): 9 pairs
- (*C* and *D*): 9 pairs

CT: Covering 3-way Combinations (1)

- Suppose a system has the following four parameters *A*, *B*, *C,* and *D*. Each parameter has three possible values:

A	*B*	*C*	*D*
1	4	7	10
2	5	8	11
3	6	9	12

- How many **3-way combinations** (combination of any three parameters, a.k.a. triples) does the system have?
  - 108

$$\underbrace{\boxed{C(4, 3)}}_{\text{select any three from four parameters}} \times \overbrace{\boxed{27}}^{\text{any three parameters has 27 possible combinations}} = 4 \times 27 = 108 \text{ triples}$$

CT: Covering 3-way Combinations (2)

108 Triples:

- 27 triples of <A, B, C>:
  <1, 4, 7>, <1, 4, 8>
  <1, 4, 9>, <1, 5, 7>
  …
- 27 triples of <A, B, D>:
  <1, 4, 10>, <1, 4, 11>
  <1, 4, 12>, <1, 5, 10>
  …
- 27 triples of <A, C, D>
  <1, 7, 10>, <1, 7, 11>
  <1, 7, 12>, <1, 8, 10>
  …
- 27 triples of <B, C, D>:
  <4, 7, 10>, <4, 7, 11>
  <4, 7, 12>, <4, 8, 10>
  …

Covered by just **33** test cases:

1,4,7,10	2,4,9,12	3,5,8,12
1,4,8,11	2,5,7,10	3,5,9,11
1,4,9,12	2,5,8,11	3,6,7,10
1,5,7,11	2,5,9,10	3,6,8,12
1,5,8,10	2,6,7,12	3,6,9,12
1,5,9,12	2,6,8,11	3,6,7,11
1,6,7,12	2,6,9,10	2,4,9,11
1,6,8,10	3,4,7,12	1,4,8,12
1,6,9,11	3,4,8,11	3,5,8,10
2,4,7,11	3,4,9,10	1,4,9,10
2,4,8,10	3,5,7,12	2,5,8,12

CT: Covering 4-way Combinations (1)

A sample system			
A	*B*	*C*	*D*
1	4	7	10
2	5	8	11
3	6	9	12

2-way combinations	can be covered by no. of test cases
54	9
3-way combinations	can be covered by no. of test cases
108	33

Observation: *n*-way combinations seems can be covered by ***less*** no. of test cases

Question 1: How many 4-way combinations does this system have?

Answer: 3 × 3 × 3 × 3 = 81

Question 2: How many test cases are required to cover those 4-way combinations?

Answer: 81, which is the entire input domain. A smaller test suite does not exist.

CT: Covering 4-way Combinations (2)

Question 3: Why does 4-way combinations have to be covered using the entire input domain while 2-way and 3-way can be covered using small test suites?

Answer: This is because when the number of parameters equals the total number of parameters, covering those combinations is the same as generating exhaustive test cases, which is the entire input domain.

CT: Covering 4-way Combinations (3)

A sample system			
A	*B*	*C*	*D*
1	4	7	10
2	5	8	11
3	6	9	12

Total number of parameters = 4

+

Covering 4-way combinations

↓

Needs the exhaustive test suite

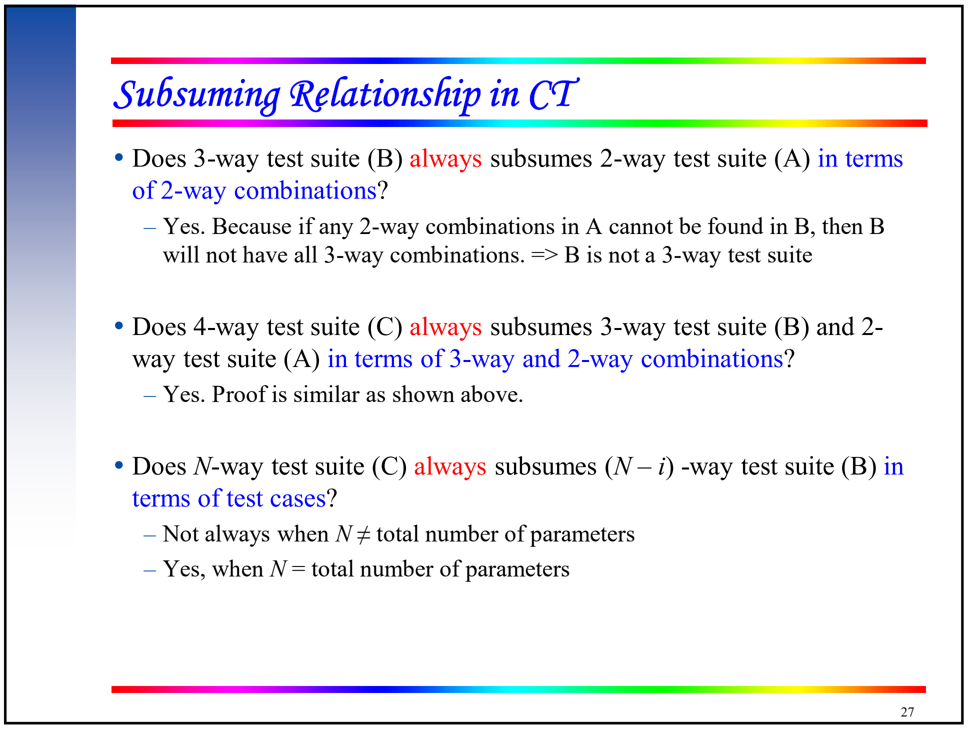
Total number of parameters = 4

+

Covering 3-way or 2-way combinations

↓

Can be covered by a small test suite

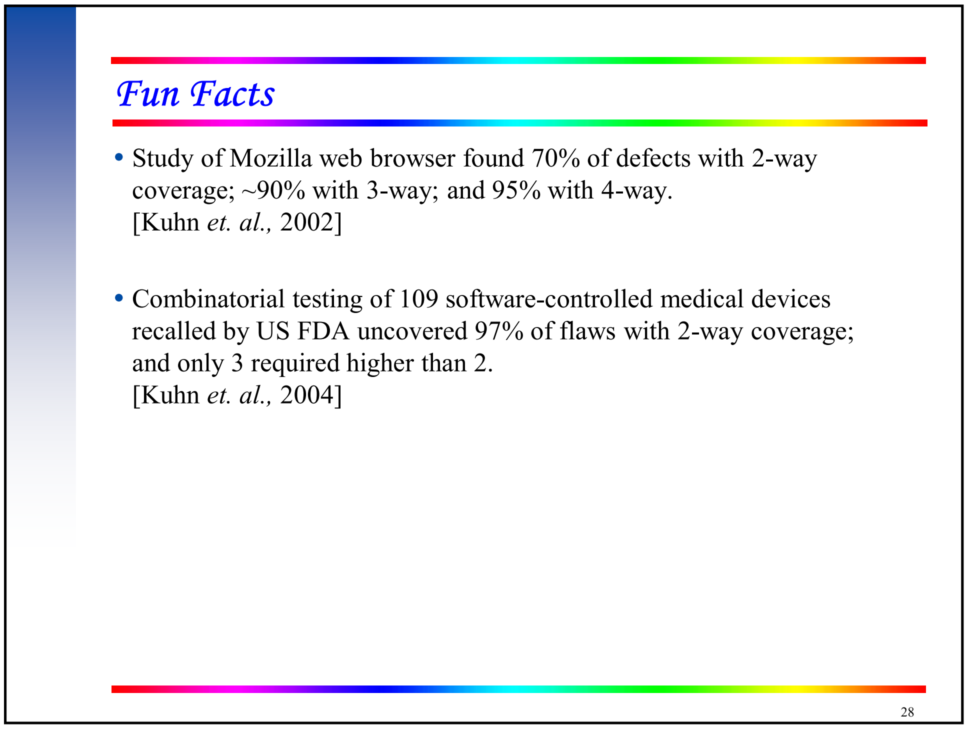


Subsuming Relationship in CT

- Does 3-way test suite (B) always subsumes 2-way test suite (A) in terms of 2-way combinations?
  - Yes. Because if any 2-way combinations in A cannot be found in B, then B will not have all 3-way combinations. => B is not a 3-way test suite
- Does 4-way test suite (C) always subsumes 3-way test suite (B) and 2-way test suite (A) in terms of 3-way and 2-way combinations?
  - Yes. Proof is similar as shown above.
- Does N-way test suite (C) always subsumes $(N - i)$ -way test suite (B) in terms of test cases?
  - Not always when $N \neq$ total number of parameters
  - Yes, when $N =$ total number of parameters

27

27



Fun Facts

- Study of Mozilla web browser found 70% of defects with 2-way coverage; ~90% with 3-way; and 95% with 4-way. [Kuhn *et. al.*, 2002]
- Combinatorial testing of 109 software-controlled medical devices recalled by US FDA uncovered 97% of flaws with 2-way coverage; and only 3 required higher than 2. [Kuhn *et. al.*, 2004]

28

28

In-Class Exercise

29