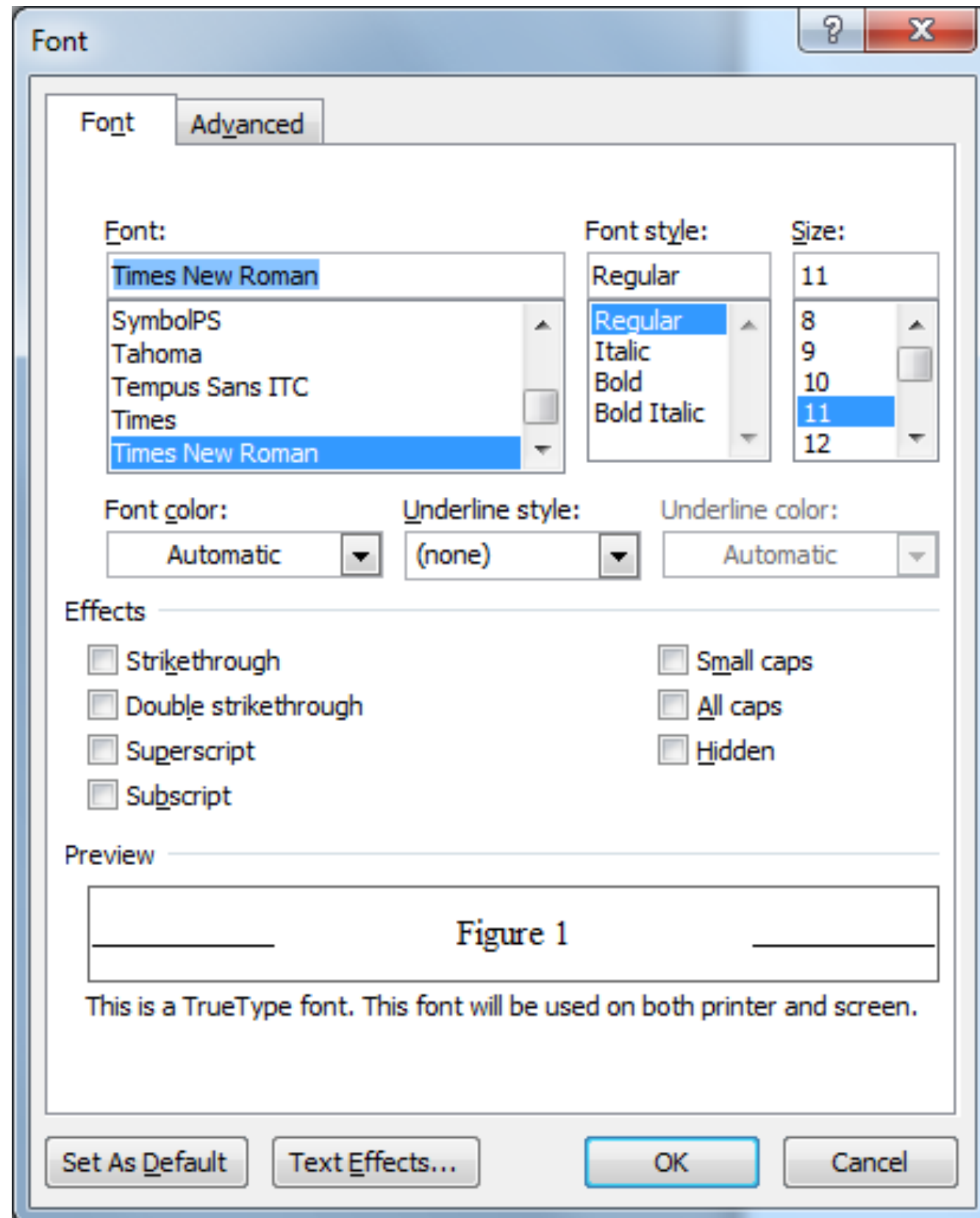


组合测试

Combinatorial Testing (CT)



40 Fonts

×

4 Styles

×

30 Sizes

×

10 Colours

×

16 Underlines

×

2⁷ Effects

组合测试

🤔 使用所有可能的输入进行测试是不可行的

🤔 并不是所有的输入参数都和故障相关

```
def function(a1, a2, a3, a4, ..., a10)
    if a5 < 10:
        # do something
        if a7 > 300:
            # faulty code
            # triggered by interactions of a5 and a7
        else:
            # do something else
```

s_client seems not using client certificate when connecting via proxy #24697

Closed

ftasnetamot opened on Jun 21, 2024

Tried with current debian based system OpenSSL 3.0.11 19 Sep 2023 (Library: OpenSSL 3.0.11 19 Sep 2023)

Connecting to FINAL-DESTINATION over proxy works perfect, as long, as the connection requires no client certificate.

`openssl s_client -proxy PROXY:3128 -connect FINAL-DESTINATION:3128` results in no problem at all

However, using client certificate, the server claims a missing cert

```
openssl s_client -proxy PROXY:3128 -cert public.cert -key private.pem -connect FINAL-DESTINATION:3128
CONNECTED(00000003)
s_client: HTTP CONNECT failed, reason= 403 Forbidden
---
no peer certificate available
---
No client certificate CA names sent
---
SSL handshake has read 1448 bytes
Verification: OK
---
New, (NONE), Cipher is (NONE)
```

So I see:

- -proxy statement works
- -client certificate work
- the combination of both does not work

Have I overseen something, or is this a bug?



moby / moby ▾

🔍 Type to search

<> Code **Issues 3.3k** 🔗 Pull requests 619 🤖 Agents 💬 Discussions ▶ Actions 📁 Projects 📖 Wiki 🛡️ Security and quality 2

28.0.1: network unavailable with iptables 1.8.11 and userland proxy disabled #49667

🔒 Closed

Bug



leo9800 opened on Mar 21, 2025

Last edited by leo9800 ▾ ⋮

Description

If all the 3 conditions below are satisfied, docker container would have no access to the network.

1. iptables 1.8.11
2. docker daemon's userland proxy is disabled
3. multiple bridge networks are created (incl. those created by compose stacks)
4. multiple containers span accross multiple bridge networks

Reproduce

1. build/install docker 28.0.1
2. install
3. ensure `"userland-proxy": false` added to `/etc/docker/daemon.json`
4. run docker
5. `docker network create network-0`
6. `docker network create network-1`

Assignees

No one as

Labels

area/net

status/O

Type

Bug

Projects

Show 1 cl

Milestone

No milesto

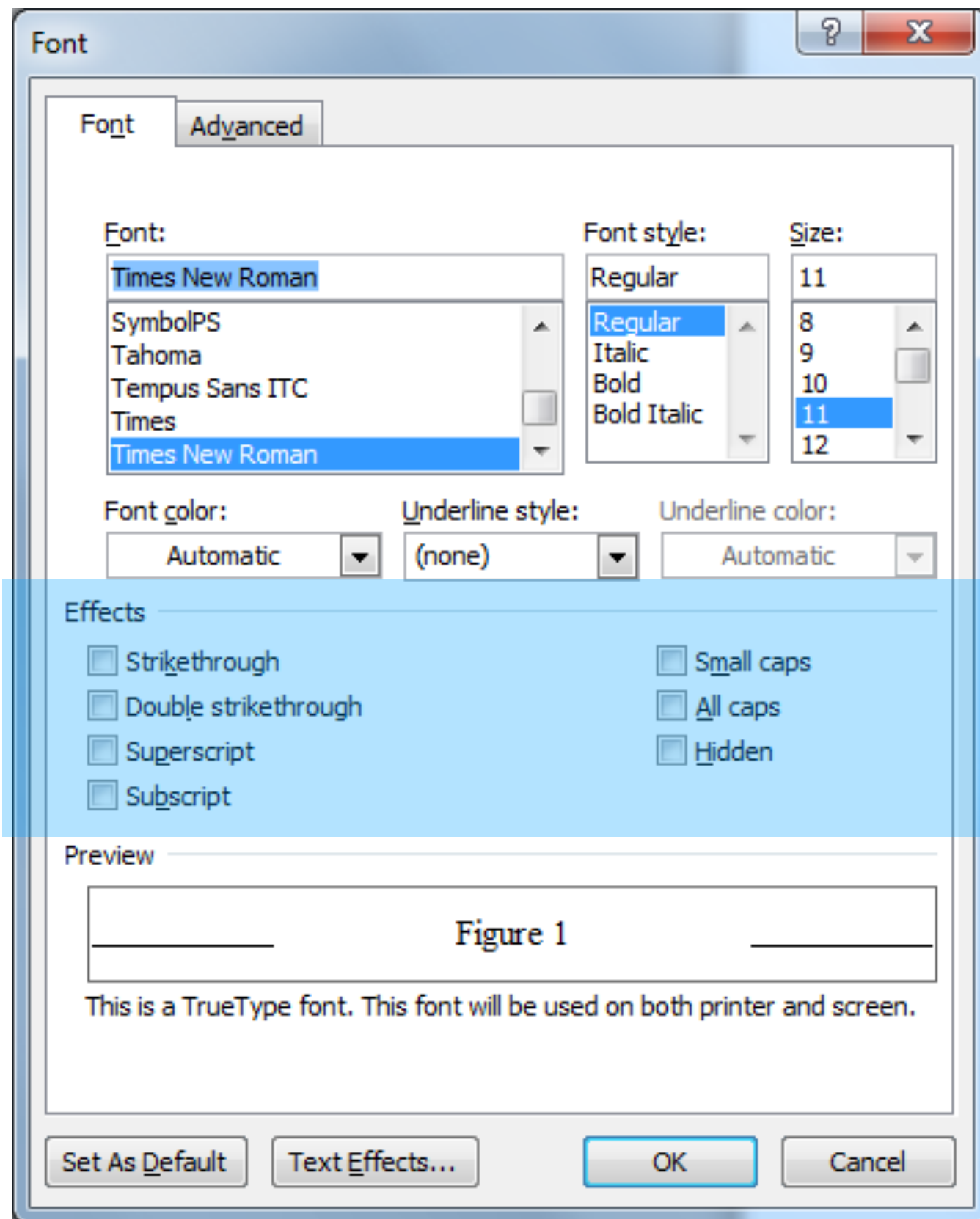
Relationsl

None vet

<https://github.com/moby/moby/issues/49667>

组合测试

- 🤔 使用所有可能的输入进行测试是不可行的
- 🤔 并不是所有的输入参数都和故障相关
- 💡 对输入参数间的交互作用进行测试 (test combinations)



假设我们对 Font Effects
进行测试

构建测试输入空间模型

Test Model

strikethrough	double strikethrough	super script	sub script	small caps	all caps	hidden
0	0	0	0	0	0	0
1	1	1	1	1	1	1

Parameters

Value Domains

- 识别影响待测软件行为的参数及其取值
(e.g., input parameters, events, configurations)
- 通常采用 Category Partition 方法
 - 结合等价类划分和边界值分析等

测试用例设计

Test Generation

strikethrough	double strikethrough	super script	sub script	small caps	all caps	hidden
0	0	0	0	0	0	0
1	1	1	1	1	1	1

- 穷尽测试需要 $2^7 = 128$ 个测试用例 (覆盖所有 7 个参数间的所有取值组合)
- 可以采用 RT 或 ART 来生成测试用例
 - 如何度量测试用例间的“距离”?

测试用例设计

Test Generation

strikethrough	double strikethrough	super script	sub script	small caps	all caps	hidden
0	0	0	0	0	0	0
1	1	1	1	1	1	1

- 穷尽测试需要 $2^7 = 128$ 个测试用例 (覆盖所有 7 个参数间的所有取值组合)
- 如果故障是由一个 2-way combination 触发
 - 例如 (*strikethrough = 1 & hidden = 1*)
 - 则需要让测试用例覆盖该组合, 例如 (1, 0, 0, 0, 0, 0, 1)
- 如何在不知道是哪两个参数、以及这两个参数的哪个取值组合的情况下系统地 (systematically) 检测这样的故障?

覆盖表

Covering Array

一个 ***t*-way 覆盖表** 是一个 $N \times k$ 矩阵 (t 的值通常称为覆盖强度), 其满足如下性质:

- 每一列 i ($1 \leq i \leq k$) 的元素都来自第 i 个参数的值域
(即每一行是一个测试数据)
- 任意 t 个参数间的所有 t -way combinations 都至少在矩阵中出现一次

覆盖表

Covering Array

strikethrough	double strikethrough	super script	sub script	small caps	all caps	hidden
0	0	1	0	0	0	1
0	1	0	1	1	1	0
1	0	0	0	1	1	1
1	1	1	1	1	0	0
1	0	0	1	0	0	0
0	1	1	1	0	1	1
1	1	1	0	0	1	0

**2-way
covering
array**

使用 7 个测试用例覆盖 7 个参数中的所有
2-way combinations (pairwise testing)

覆盖表

Covering Array

strikethrough	double strikethrough	super script	sub script	small caps	all caps	hidden
0	0	1	0	0	0	1
0	1	0	1	1	1	0
1	0	0	0	1	1	1
1	1	1	1	1	0	0
1	0	0	1	0	0	0
0	1	1	1	0	1	1
1	1	1	0	0	1	0

2-way
covering
array

覆盖 strikethrough 和 double strikethrough 间的
所有 $2 \times 2 = 4$ 种取值组合

覆盖表

Covering Array

strikethrough	double strikethrough	super script	sub script	small caps	all caps	hidden
0	0	1	0	0	0	1
0	1	0	1	1	1	0
1	0	0	0	1	1	1
1	1	1	1	1	0	0
1	0	0	1	0	0	0
0	1	1	1	0	1	1
1	1	1	0	0	1	0

2-way
covering
array

覆盖 strikethrough 和 super script 间的
所有 $2 \times 2 = 4$ 种取值组合

覆盖表

Covering Array

strikethrough	double strikethrough	super script	sub script	small caps	all caps	hidden
0	0	1	0	0	0	1
0	1	0	1	1	1	0
1	0	0	0	1	1	1
1	1	1	1	1	0	0
1	0	0	1	0	0	0
0	1	1	1	0	1	1
1	1	1	0	0	1	0

2-way
covering
array

覆盖 super script 和 all caps 间的
所有 $2 \times 2 = 4$ 种取值组合

覆盖表

Covering Array

strikethrough	double strikethrough	super script	sub script	small caps	all caps	hidden
0	0	1	0	0	0	1
0	1	0	1	1	1	0
1	0	0	0	1	1	1
1	1	1	1	1	0	0
1	0	0	1	0	0	0
0	1	1	1	0	1	1
1	1	1	0	0	1	0

2-way
covering
array

此时一定存在一条测试用例覆盖
(*strikethrough = 1 & hidden = 1*)

覆盖表

Covering Array

使用 t -way 覆盖表可以极大降低测试用例集规模 (especially for $t = 2$)

Covering array size increases logarithmically with the number of parameters k , but exponentially with the coverage strength t .



34 switches

$2^{34} = 1.7 \times 10^{10}$ inputs

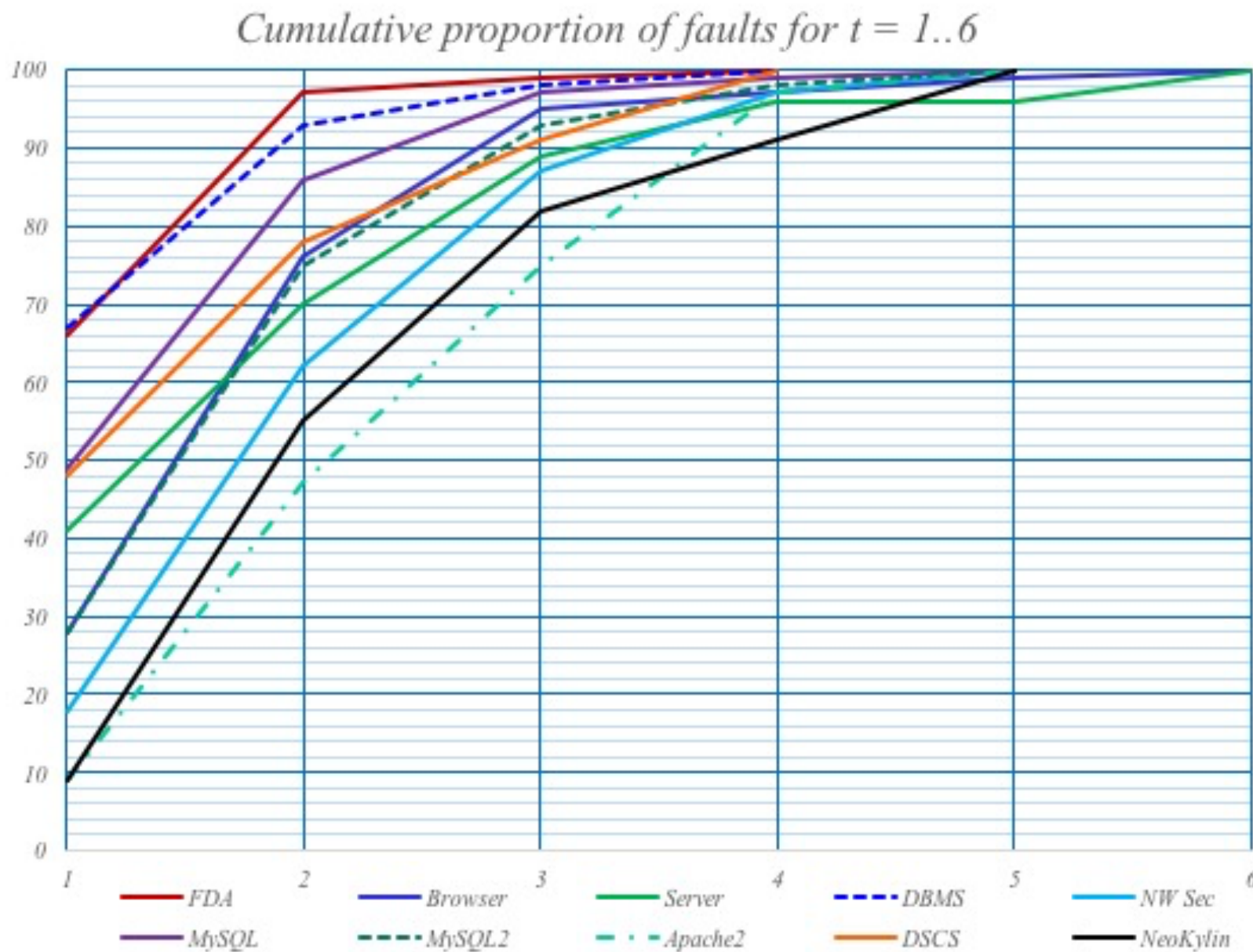
组合测试的有效性

组合测试使用 t -way 覆盖表来作为测试用例集

- 确保覆盖任意 $k \leq t$ 个参数间的所有可能取值组合
 - 满足 t -way 覆盖一定满足 $(t-1)$ -way 覆盖
- 如果和故障相关的参数个数不超过 t ，则组合测试的故障检测能力和穷尽测试 (exhaustive testing) 等价
- 真实软件故障对应的 t 的值是多少？

组合测试的有效性

Empirical Evidence of the *Interaction Rule*



来自 National Institute of Standards and Technology (NIST) 的经验研究

[1999~2004]

- 绝大多数 (70%~80%) 的软件故障都是由 1~2 个参数间的相互作用引起的
- 和软件故障相关的参数个数通常不超过 6
- 使用 $t = 2$ 是一个非常 **cost-effectiveness** 的选择

PROJECTS

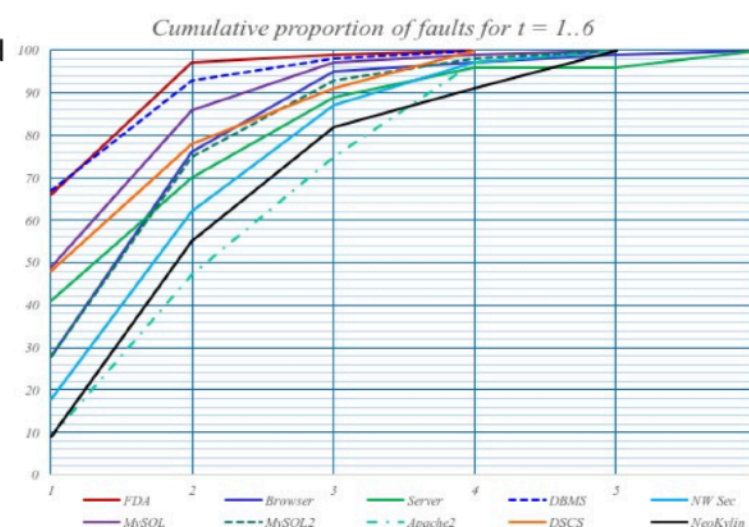
Combinatorial Methods for Trust and Assurance ACTS



Overview

Combinatorial methods reduce costs for testing, and *have important applications in software engineering:*

- **Combinatorial or t-way testing** is a proven method for better testing at lower cost. The key insight underlying its effectiveness resulted from a series of studies by NIST from 1999 to 2004. NIST research showed that most software bugs and failures are caused by one or two parameters, with progressively fewer by three or more, which means that combinatorial testing can provide more efficient fault detection than conventional methods. [Multiple studies](#) have shown fault detection equal to exhaustive testing with a 20X to 700X reduction in test set size. New algorithms compressing combinations into a small number of tests have made this method practical for industrial use, providing better [high assurance software](#)



ification: Input space

d in [assurance and](#)

s, because current methods for assurance of safety critical systems rely on measures of
pply to many autonomous systems. Combinatorial methods, including a theorem relating

<https://csrc.nist.gov/projects/automated-combinatorial-testing-for-software>

D. Richard Kuhn
(now at Virginia Tech)

PROJECT LINKS

Overview

FAQs

Publications

ADDITIONAL PAGES

Quick start

Downloadable Tools

Fundamental background papers

Tutorials and Documentation

Combinatorial Methods in Testing

Why do Combinatorial Testing?

Event Sequence Testing

Case Studies

Assured autonomy - AI Safety and Security

AI-Enabled Systems



广泛的应用

Configuration Testing

Input Parameter Testing

Event Driven Software (GUI)

Software Product Line

Web Service Testing

Security Testing

App Testing

Big Data Application Testing

AI Testing

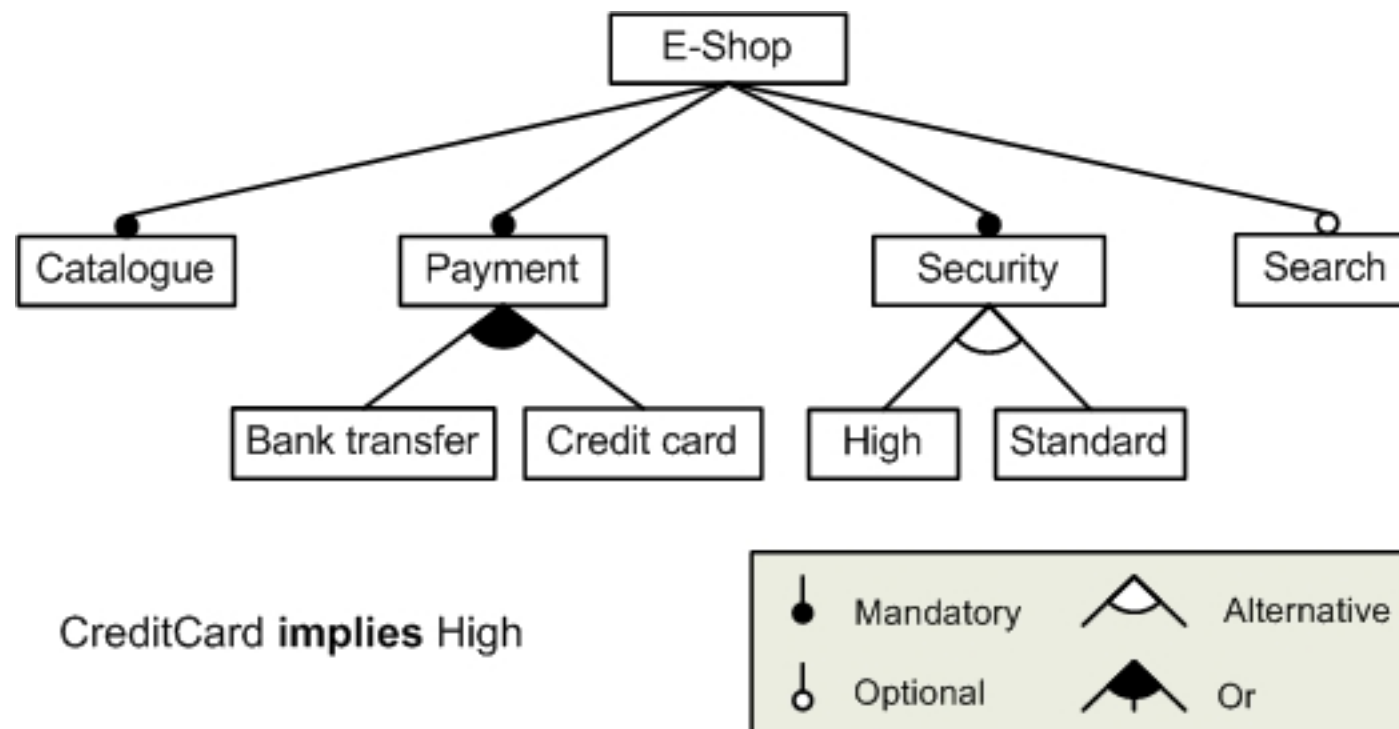
...



Software Product Line (SPL)

基于软件产品线 Feature Model 的测试

- 如何从大量合法产品中选择少量代表性配置进行测试？



- Mandatory – child feature must be selected.
- Optional – child feature can be selected or not selected.
- Or – at least one of the sub-features must be selected.
- Alternative (xor) – exactly one of the sub-features must be selected.

Problem: unknown factors causing failures of F-16 ventral fin



Figure 1. LANTIRN pod carriage on the F-16.



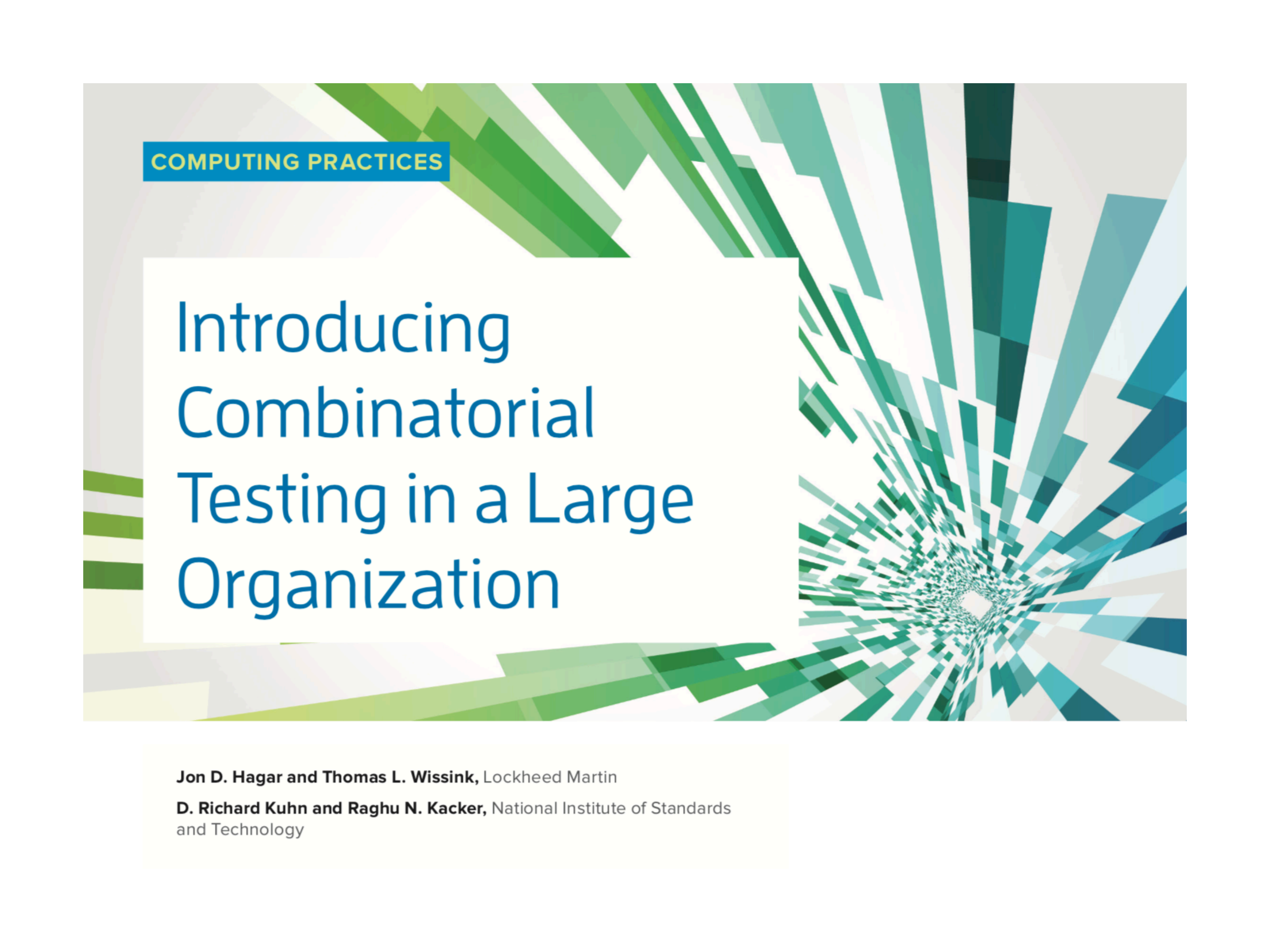
Figure 2. F-16 ventral fin damage on flight with LANTIRN

Can the problem factors be found efficiently?

Original solution: Lockheed Martin engineers spent many months with wind tunnel tests and expert analysis to consider interactions that could cause the problem

Combinatorial testing solution: modeling and simulation using ACTS

Parameter	Values
Aircraft	15, 40
Altitude	5k, 10k, 15k, 20k, 30k, 40k, 50k
Maneuver	hi-speed throttle, slow accel/dwell, L/R 5 deg side slip, L/R 360 roll, R/L 5 deg side slip, Med accel/dwell, R-L-R-L banking, Hi-speed to Low, 360 nose roll
Mach (100 th)	40, 50, 60, 70, 80, 90, 100, 110, 120



COMPUTING PRACTICES

Introducing Combinatorial Testing in a Large Organization

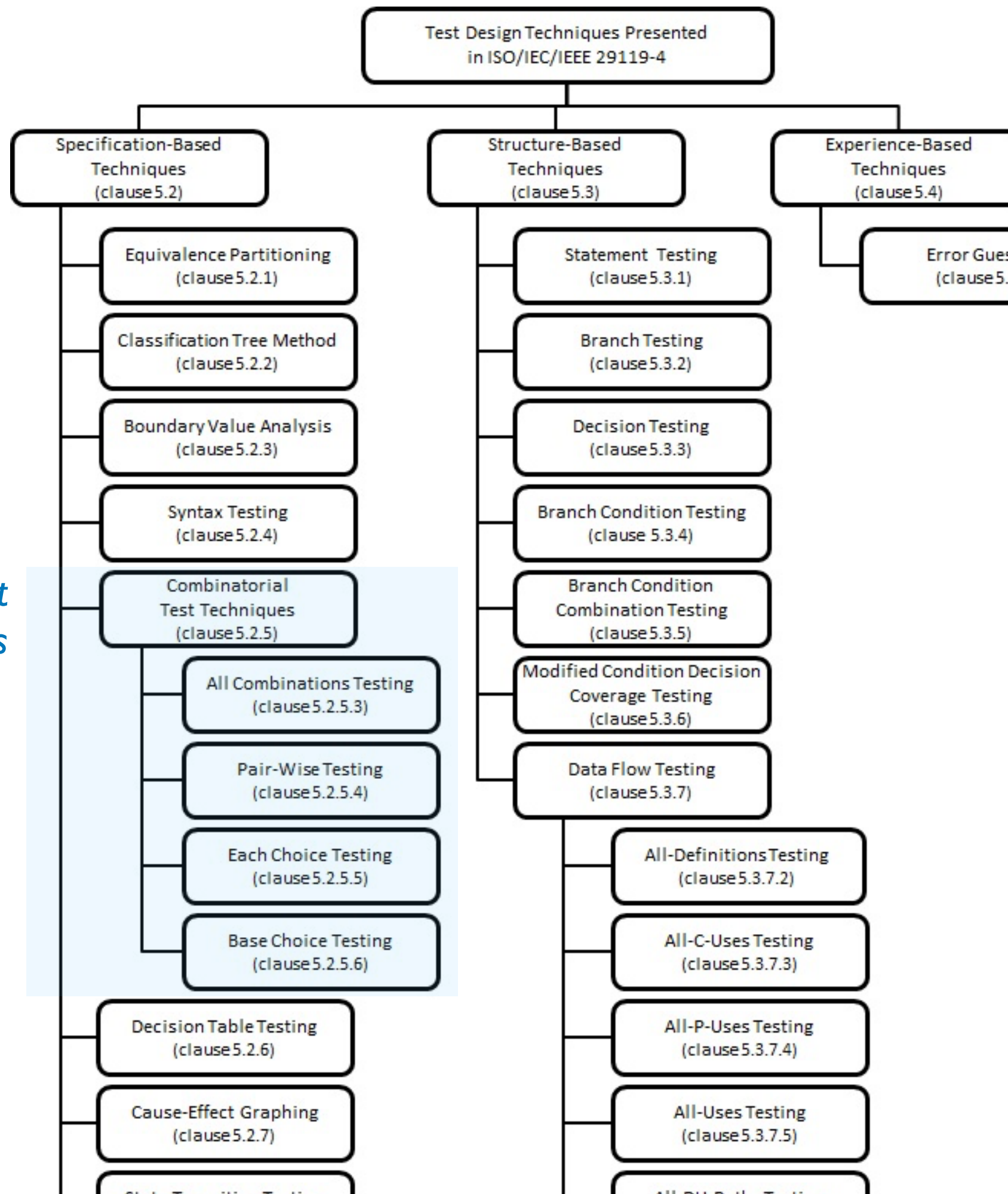
Jon D. Hagar and Thomas L. Wissink, Lockheed Martin

D. Richard Kuhn and Raghu N. Kacker, National Institute of Standards
and Technology

测试标准

ISO/IEC/IEEE 29119

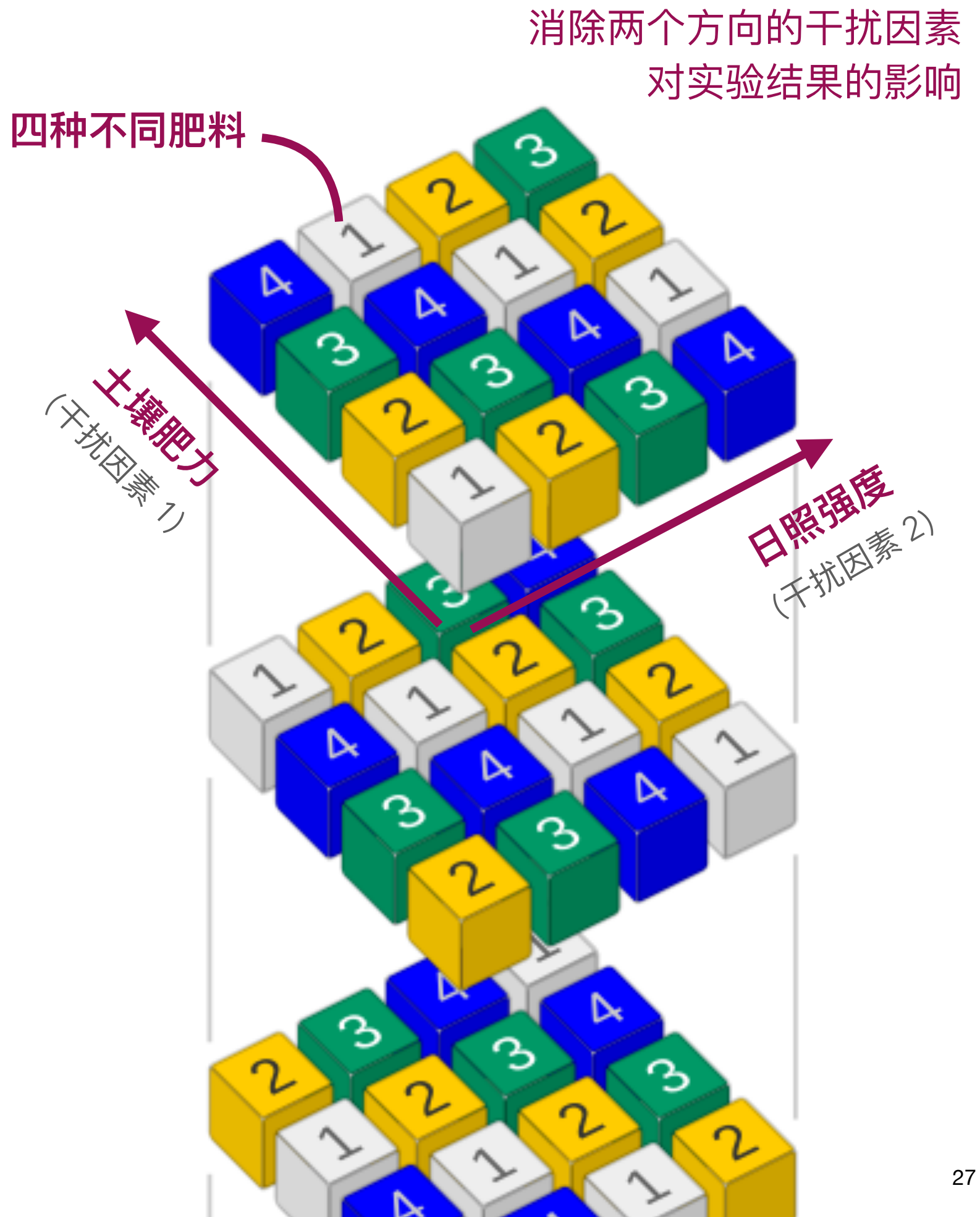
Combinatorial Test Techniques



拉丁方

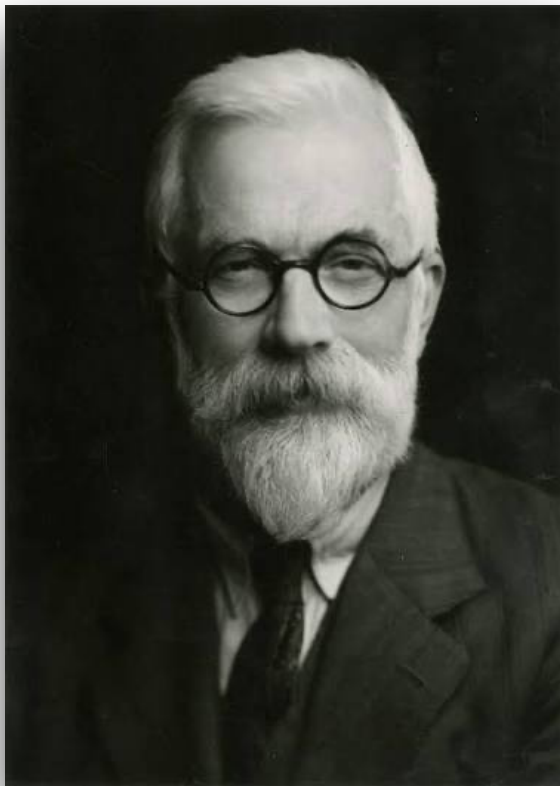
Latin Square

拉丁方是一种 $n \times n$ 方阵，其包含 n 种不同元素，且每种元素在同一行和同一列中都只出现一次



实验设计

Design of Experiment (DOE)



Ronald Fisher

允许多因素同时实验，
发现因素之间的交互作用
[1920s, 英国农业试验]



Genichi Taguchi
田口玄一

基于正交表 (Orthogonal Array)
来选择稳定的参数配置
[1950s~1980s, 日本和美国
工业质量改进]

正交表

Orthogonal Array

正交表是一个大小为 $N \times k$ 的矩阵，记为 $L_N(v^k)$ ，其满足如下性质：

- 每一列 i ($1 \leq i \leq k$) 的元素都来自一个包含 v 个 Level 的有限集合
- 对于矩阵中的任意 $t = 2$ 列，其所有可能的 Level 组合都必须在矩阵中出现，且出现的次数完全相同

$L_9(3^4)$

试验号	因素 A	因素 B	因素 C	因素 D
1	1	1	1	1
2	1	2	2	2
3	1	3	3	3
4	2	1	2	3
5	2	2	3	1
6	2	3	1	2
7	3	1	3	2
8	3	2	1	3
9	3	3	2	1

覆盖表

Covering Array

相比于正交表严格要求每个组合出现相同的次数 (且对特定参数场景并不存在正交表), 覆盖表仅要求每个组合至少出现一次

- 面向软件测试而非实验设计的需求
- 不同参数的值域大小可以不一样 (Mixed Covering Array)

Test Model

P1	P2	P3	P4
0	0	0	0
1	1	1	1
		2	2

2-way Covering Array

P1	P2	P3	P4
0	0	0	0
1	1	0	1
0	1	0	2
1	0	1	0
0	1	1	1
1	0	1	2
0	1	2	0
1	0	2	1
1	0	2	2

History of Combinatorial Testing

1950 1960 1970 1980 '85 1990 '95 2000 '05 2010

(1957: Decision table)

~1967: Equivalence partitioning,
Boundary value analysis

1970: Cause Effect Graph
[Elmendorf]

Combinatorial techniques

Input condition
analysis techniques

(1976: Test Factor Analysis
Method) [Fujitsu]

1988: Category-partition method
[Ostrand, Balcer]

1993: Classification-tree method
[Grochtmann, Grimm]

(1983~)1984: Orthogonal array
/ Combination Table
[Sato, Shimokawa]

Fujitsu

1987: Test Case Design Support
System [Tatsumi]

(1983~)1985: Orthogonal Latin Squares
[R. Mandl]

AT&T, Bellcore

(198x~)1992: OATS
[Brownlie, Prowse, Phadke]

(1990~)1994: CATS
[Sherwood]

(1992~)1994: AETG
[Cohen, et. al]

(199x~)2004: Orthogonal Array
(HAYST method)
[Akiyama(Fuji Xerox)]

1998: IPO
[Lei, Tai]

2007: FireEye 2009: ACTS
(IPOG) [Lei, Kuhn]

2000: CTE XL
[Daimler Chrystler]

2000: Covering arrays
[Williams]

(2000~)2004: PICT
[Microsoft]

(C) K. Tatsumi 2012

Keizo Tatsumi



可变力度覆盖表

Variable Strength Covering Array

基于组合覆盖的思想，我们可以为不同场景设计各式各样的“覆盖表”

- 例如，对不同的参数集合应用不同的覆盖强度

2-way Coverage

P1	P2	P3	P4	P5	P6	P7
0	1	1	0	1	1	0
1	0	0	0	0	0	1
1	1	0	1	1	0	0
0	0	0	1	0	1	1
1	1	1	0	0	1	1
0	0	1	0	0	0	0
0	0	0	1	1	1	1
0	1	0	0	1	1	0
1	0	1	0	0	0	1

3-way Coverage

序列覆盖表

Sequence Covering Array

基于组合覆盖的思想，我们可以为不同场景设计各式各样的“覆盖表”

- 例如，假设测试用例由 $\{0, 1, 2, 3, 4\}$ 这 5 个事件的一种排列方式构成 (permutation of events)
- 覆盖不同的 t -way sequence
 $\langle 0, 1 \rangle, \langle 1, 0 \rangle, \langle 0, 2 \rangle \dots$
 $\langle 0, 1, 2 \rangle, \langle 0, 2, 1 \rangle \dots$

3-way Sequence Covering Array

Event 1	Event 2	Event 3	Event 4	Event 5
1	3	4	2	0
0	2	4	3	1
4	0	1	2	3
3	2	1	0	4
1	0	4	3	2
2	3	4	0	1
4	2	1	0	3
3	0	1	2	4
1	0	3	2	4
4	2	3	0	1

覆盖表生成

给定一个测试输入空间模型和覆盖强度 t ，如何构造得到一个规模尽可能小的 t -way 覆盖表

- 一个非常具有挑战的组合优化问题 (大部分情况下是 NP-Hard)
- 吸引了大量数学家和计算机科学家的关注



Colbourn tables

Ulrike Grömping July 25, 2025

Purpose

This file gives access to a November 2024 status of Covering Array tables maintained by Charlie Colbourn (short: th as those tables are no longer available to the public at their previous location (see references). In a few places (e.g., for strength $t = 3$ with $v = 4$ levels for more than 798 columns or for strength $t = 6$ with $v = 2$ levels for $k = 30, \dots, 72$; not systematically checked for more such instances), this file has more recent content than the latest wayback machine crawl of February 2025 at <https://web.archive.org/web/20250211193915/https://www.public.asu.edu/~ccolbou/src/tabby/catable.html>, as the latest crawl dates for linked individual tables are all earlier than February 2025. The one instance for $t = 2$, $v = 10$ with a crawl date later than November 2024 (crawled in Dec 2024) has unchanged content.

Tables

There is a table for each combination of covering array strength t and number of levels per column (i.e., order) v . The table entries indicate the largest number of columns k for a given run size N or the smallest runs size N required for a given number of columns N based on a known construction, which is hinted at in the column headed "Source". These tables are a replication of those by Colbourn, as they were available recently (download in November 2024), but are not available at present.

	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
2	t2v2	t2v3	t2v4	t2v5	t2v6	t2v7	t2v8	t2v9	t2v10	t2v11	t2v12	t2v13	t2v14	t2v15	t2v16
3	t3v2	t3v3	t3v4	t3v5	t3v6	t3v7	t3v8	t3v9	t3v10	t3v11	t3v12	t3v13	t3v14	t3v15	t3v16
4	t4v2	t4v3	t4v4	t4v5	t4v6	t4v7	t4v8	t4v9	t4v10	t4v11	t4v12	t4v13	t4v14	t4v15	t4v16
5	t5v2	t5v3	t5v4	t5v5	t5v6	t5v7	t5v8	t5v9	t5v10	t5v11	t5v12	t5v13	t5v14	t5v15	t5v16

In-Parameter-Order (IPO)

1. Initialise a test set T to cover all combinations of the first t parameters
2. *Horizontally* extend T to include a new parameter (assign parameter values to cover the most uncovered t -way combinations)
3. *Vertically* extend T to cover the remaining t -way combinations
4. Repeat steps 2 and 3, until all t -way combinations are covered

In-Parameter-Order (IPO)

Parameters P_1, P_2, P_3, P_4 with $V_1 = V_2 = \{0, 1\}$, and $V_3 = V_4 = \{0, 1, 2\}$

Suppose that we are going to generate a 2-way covering array

P_1	P_2
0	0
0	1
1	0
1	1

First, initialise T to cover all combinations of the first *two* parameters

In-Parameter-Order (IPO)

P_1	P_2	P_3
0	0	
0	1	
1	0	
1	1	

P_1	P_2	P_3
0		0
0		1
0		2
1		0
1		1
1		2
	0	0
	0	1
	0	2
	1	0
	1	1
	1	2

Next, add P_3 into the table, which includes 12 combinations that need to be covered
(horizontal extension)

In-Parameter-Order (IPO)

P_1	P_2	P_3
0	0	0
0	1	
1	0	
1	1	

P_1	P_2	P_3
0		0
0		1
0		2
1		0
1		1
1		2
	0	0
	0	1
	0	2
	1	0
	1	1
	1	2

Next, add P_3 into the table, which includes 12 combinations that need to be covered
(horizontal extension)

In-Parameter-Order (IPO)

P_1	P_2	P_3
0	0	0
0	1	1
1	0	
1	1	

P_1	P_2	P_3
0		0
0		1
0		2
1		0
1		1
1		2
	0	0
	0	1
	0	2
	1	0
	1	1
	1	2

Next, add P_3 into the table, which includes 12 combinations that need to be covered
(horizontal extension)

In-Parameter-Order (IPO)

P_1	P_2	P_3
0	0	0
0	1	1
1	0	1
1	1	

P_1	P_2	P_3
0		0
0		1
0		2
1		0
1		1
1		2
	0	0
	0	1
	0	2
	1	0
	1	1
	1	2

Next, add P_3 into the table, which includes 12 combinations that need to be covered
(horizontal extension)

In-Parameter-Order (IPO)

P_1	P_2	P_3
0	0	0
0	1	1
1	0	1
1	1	0

P_1	P_2	P_3
0		0
0		1
0		2
1		0
1		1
1		2
	0	0
	0	1
	0	2
	1	0
	1	1
	1	2

Next, add P_3 into the table, which includes 12 combinations that need to be covered
(horizontal extension)

In-Parameter-Order (IPO)

P_1	P_2	P_3
0	0	0
0	1	1
1	0	1
1	1	0

P_1	P_2	P_3
0		0
0		1
0		2
1		0
1		1
1		2
	0	0
	0	1
	0	2
	1	0
	1	1
	1	2

Next, add P_3 into the table, which includes 12 combinations that need to be covered
(horizontal extension)

In-Parameter-Order (IPO)

P_1	P_2	P_3
0	0	0
0	1	1
1	0	1
1	1	0
0	0	2

P_1	P_2	P_3
0		0
0		1
0		2
1		0
1		1
1		2
	0	0
	0	1
	0	2
	1	0
	1	1
	1	2

For the remaining four uncovered combinations, add new test cases to cover them
(vertical extension)

In-Parameter-Order (IPO)

P_1	P_2	P_3
0	0	0
0	1	1
1	0	1
1	1	0
0	0	2
1	1	2

P_1	P_2	P_3
0		0
0		1
0		2
1		0
1		1
1		2
	0	0
	0	1
	0	2
	1	0
	1	1
	1	2

For the remaining four uncovered combinations, add new test cases to cover them
(vertical extension)

In-Parameter-Order (IPO)

P_1	P_2	P_3	P_4
0	0	0	
0	1	1	
1	0	1	
1	1	0	
0	0	2	
1	1	2	

Repeat the **horizontal** and **vertical extensions** for each remaining parameter, until a 2-way covering array is produced

In-Parameter-Order (IPO)

P_1	P_2	P_3	P_4
0	0	0	
0	1	1	
1	0	1	
1	1	0	
0	0	2	
1	1	2	

Repeat the **horizontal** and **vertical extensions** for each remaining parameter, until a 2-way covering array is produced

ACTS - ACTS Main Window

System Edit Operations Help

Algorithm: IPOG Strength: 2 Status : A new test set is built.

System View

[Root Node]

SYSTEM-TCAS_WithInv

Cur_Vertical_Sep

299

300

601

-1

-2

High_Confidence

Two_of_Three_Rep

Own_Tracked_Alt

Other_Tracked_Alt

Own_Tracked_Alt_R

Alt_Layer_Value

Up_Separation

Down_Separation

Other_RAC

Other_Capability

Climb_Inhibit

Relations

Constraints

Test Result

	Cur_Vertical_Sep	High_Confidence	Two_of_Three_Reports_Valid	Own_Tracked_Alt	Other_Tracked_Alt	Own_Tracked_Alt
1	300	FALSE	FALSE	2	2	601
2	601	TRUE	TRUE	1	1	600
3	299	FALSE	TRUE	2	1	601
4	300	TRUE	FALSE	1	2	600
5	601	TRUE	FALSE	2	1	600
6	299	TRUE	FALSE	1	2	601
7	300	FALSE	TRUE	1	2	600
8	601	FALSE	TRUE	2	1	601
9	299	TRUE	TRUE	1	2	600
10	300	FALSE	FALSE	2	1	601
11	601	TRUE	TRUE	1	2	600
12	299	FALSE	FALSE	2	2	601
13	300	TRUE	FALSE	1	1	600
14	601					
15	299					
16	300					
17	601					
18	299					
19	300					
20	601					
21	299					
22	300					
23	601					
24	299					
25	300					
26	601					
27	299					
28	300					
29	601					
30	299					
31	300					

Most Influential Paper Award (MIP)
for the ACTS paper published in 2013



Sun 16 - Thu 20 April 2023 Dublin, Ireland

Attending ▾

Sponsorship Opportunities

Program ▾

Tracks ▾

Organization ▾

Search

Series ▾

Sign in

Sign up

[ICST 2023 \(series\)](#) / [MIPs](#) /

ACTS: A Combinatorial Test Generation Tool

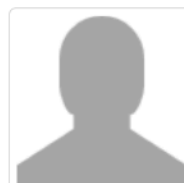
Who Linbin Yu, Yu Lei, Raghu Kacker, Richard Kuhn

Track [ICST 2023 MIPs](#)

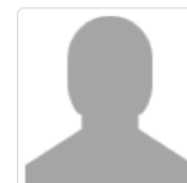
When Tue 18 Apr 2023 16:45 - 17:30 at [Grand canal](#) - [MIPs](#) Chair(s): [Mike Papadakis](#)

Abstract In this paper, we introduce a combinatorial test generation research tool called Advanced Combinatorial Testing System (or ACTS). ACTS supports t-way combinatorial test generation with several advanced features such as mixed-strength test generation and constraint handling. To facilitate its use and integration with other tools, ACTS provides three types of external interface, including a graphic user interface, a command line interface, and an application programming interface. ACTS is a freely distributed research tool and has been downloaded by more than 1200 companies and organizations.

DOI <https://doi.org/10.1109/ICST.2013.52>



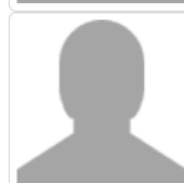
Linbin Yu



Yu Lei



Raghu Kacker
National Institute of
Standards and Technology
United States



Richard Kuhn
National Institute of
Standards and Technology
United States

覆盖表生成

- 基于数学的方法 能构造理论最优的覆盖表，但受限于问题实例
- 基于计算的方法
 - 贪心算法 (Greedy) 速度快，工业界广泛应用
 - One-Test-At-a-Time (OTAT)
 - In-Parameter-Order (IPO)
 - 搜索算法 (Search-Based) 耗时长，很多 Best-Known 的构造方法
(将在 "基于搜索的测试" 部分介绍)

约束

Constraints

strikethrough	double strikethrough	superscript	subscript	small caps	all caps	hidden
0	0	0	0	0	0	0
1	1	1	1	1	1	1

superscript and *subscript* cannot both be enabled

Test Input Space Model

Parameters

Value Domains

Constraints

在构建测试输入空间时，除了识别参数及其取值外，还需要考虑参数取值之间的依赖关系

- 违反约束的测试用例将被视为 Invalid
e.g., (0, 0, 1, 1, 0, 0, 0)

约束覆盖表

Constrained Covering Array

strikethrough	double strikethrough	super script	sub script	small caps	all caps	hidden
0	1	1	0	1	1	0
1	0	0	0	0	0	1
1	1	0	1	1	0	0
0	0	0	1	0	1	1
1	1	1	0	0	1	1
0	0	1	0	0	0	0
0	0	0	1	1	1	1

2-way
constrained
covering
array

每个测试用例都满足约束，且所有满足约束的
t-way combinations 都至少被覆盖一次

生成约束覆盖表

约束覆盖表的构造通常比普通覆盖表要困难

- 如何表示约束?
- 如何知道有多少满足约束的 *t*-way combination?
(覆盖表生成算法的停止条件)
- 如何判断一个测试用例 (或部分测试用例) 是否满足约束?

基于禁止元组的方法

Forbidden Tuple Based

使用一个不允许出现的 t -way combination (即禁止元组) 的集合来表示约束

(superscript = 1 & subscript = 1)

- 如果一个 (partial) test case 包含任何一个禁止元组，则其违反约束
- 如果其不包含禁止元组，则一定满足约束 ?

基于禁止元组的方法

Forbidden Tuple Based

使用一个不允许出现的 t -way combination (即禁止元组) 的集合来表示约束

(superscript = 1 & subscript = 1)
(subscript = 0 & small cpas = 0) $\rightarrow$ (superscript = 1 & small cpas = 0)

- 基于给定的禁止元组集合可推出更多的隐含约束 (implicit constraints)
- 计算最小禁止元组 (Minimum Forbidden Tuples) 集合
 - 反复 Derive 和 Simplify 禁止元组，直到没有新的 MFT 为止
 - 然后在生成过程中就可以和 MFT 集合中的元组进行比较

基于禁止元组的方法

Forbidden Tuple Based

使用一个不允许出现的 t -way combination (即禁止元组) 的集合来表示约束

- 有些约束形式会导致非常庞大的禁止元组集合
 - 假设有 2 个参数, 其中 $V_1 = \{1, 3, 5, 7\}$ 和 $V_2 = \{2, 4, 6, 8\}$
约束为 $P_1 + P_2 < 9$
 - 假设有 10 个参数, 每个参数的值域为 $V_i = \{0, 1\}$
约束为 $P_1 + P_2 + \dots + P_{10} < 5$

基于约束求解器的方法

Constraints Solver Based

使用某种约束编码形式来表示约束

IF superscript = 1 *THEN* subscript = 0

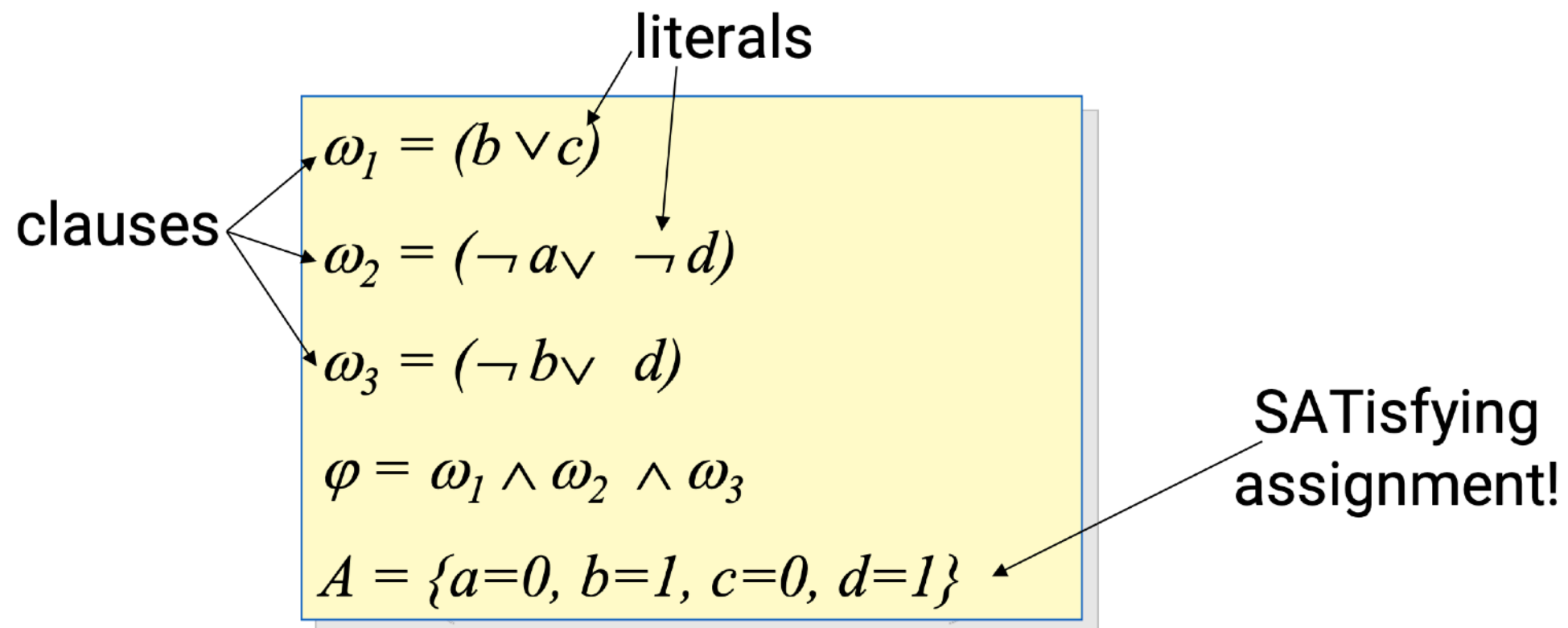
- 将判断 (partial) test case 是否满足约束转化为一个约束可满足性问题 (constraint satisfaction problem)
- 借助先进的 Constraint Solver 来实现

基于约束求解器的方法

Constraints Solver Based

例如使用 SAT Solver (Boolean Satisfiability Problem)

- 给定一个 CNF 形式的命题公式，判断是否存在一组布尔变量赋值，以使该公式为真 (Truth Assignment)
- 第一个被证明为 NP-Complete 的问题 [Cook–Levin theorem, 1971]



基于约束求解器的方法

Constraints Solver Based

Encode each parameter value
as a boolean variable

Represent constraints
as CNF formulas

P1	P2	P3	P4
0	0	0	0
1	1	1	1
		2	2

Forbidden Tuple:
(P2 = 1 & P3 = 1)

$$P1 = 0 \leftrightarrow x_1$$

$$P1 = 1 \leftrightarrow x_2$$

$$P2 = 0 \leftrightarrow x_3$$

$$P2 = 1 \leftrightarrow x_4$$

$$P3 = 0 \leftrightarrow x_5$$

$$P3 = 1 \leftrightarrow x_6$$

$$P3 = 2 \leftrightarrow x_7$$

$$P4 = 0 \leftrightarrow x_8$$

$$P4 = 1 \leftrightarrow x_9$$

$$P4 = 2 \leftrightarrow x_{10}$$

forbidden tuple

$$(\neg x_4 \vee \neg x_6)$$

at-most

$$(\neg x_1 \vee \neg x_2)$$

$$(\neg x_3 \vee \neg x_4)$$

$$(\neg x_5 \vee \neg x_6) \wedge (\neg x_5 \vee \neg x_7) \wedge (\neg x_6 \vee \neg x_7)$$

$$(\neg x_8 \vee \neg x_9) \wedge (\neg x_8 \vee \neg x_{10}) \wedge (\neg x_9 \vee \neg x_{10})$$

at-least

$$(x_1 \vee x_2)$$

$$(x_3 \vee x_4)$$

$$(x_5 \vee x_6 \vee x_7)$$

$$(x_8 \vee x_9 \vee x_{10})$$

基于约束求解器的方法

Constraints Solver Based

P1	P2	P3	P4
0	0	0	0
1	1	1	1
		2	2

Forbidden Tuple:
(P2 = 1 & P3 = 1)



$$C = C_{consistency} \wedge C_{forbidden} \wedge C_{solution}$$

e.g., $(x_1 \wedge x_3)$

$(x_2 \wedge x_4 \wedge x_6)$

If the boolean formula C is satisfiable,
then the solution is considered as *valid*

(that is, we can find a truth
assignment for variables in C)

基于约束求解器的方法

Constraints Solver Based

还可以把“覆盖表生成问题”编码为约束可满足问题

P1	P2	P3	P4
0	0	0	0
1	1	1	1
		2	2

Forbidden Tuple:

(P2 = 1 & P3 = 1)



$x_{i,j,k} \leftrightarrow$ the j -th parameter of the i -th row takes value k

$c_{i,m} \leftrightarrow$ the t -way combination m is covered in the i -th row

(let M = auxiliary variables for all valid t -way combinations)

at-most & at-least

$$\begin{aligned} &(\neg x_{1,1,0} \vee \neg x_{1,1,1}) \wedge (x_{1,1,0} \vee x_{1,1,1}) \\ &(\neg x_{1,2,0} \vee \neg x_{1,2,1}) \wedge (x_{1,2,0} \vee x_{1,2,1}) \\ &\dots \end{aligned}$$

$$C_1 = \bigwedge_{i=1}^N C_{consistency}^i$$

forbidden tuple

$$\begin{aligned} &(\neg x_{1,2,1} \vee \neg x_{1,3,1}) \\ &(\neg x_{2,2,1} \vee \neg x_{2,3,1}) \\ &\dots \end{aligned}$$

$$C_2 = \bigwedge_{i=1}^N (\neg x_{i,2,1} \vee \neg x_{i,3,1})$$

coverage requirement

$$\begin{aligned} &(c_{1,1} \vee c_{2,1} \vee \dots \vee c_{N,1}) \\ &(c_{1,2} \vee c_{2,2} \vee \dots \vee c_{N,2}) \\ &\dots \end{aligned}$$

$$c_{1,1} \leftrightarrow (x_{1,1,0} \wedge x_{1,2,0})$$

$$\text{i.e., } ((\neg c_{1,1} \vee x_{1,1,0}) \wedge (\neg c_{1,1} \vee x_{1,2,0})) \wedge (\neg x_{1,1,0} \vee \neg x_{1,2,0} \vee c_{1,1})$$

$$c_{2,1} \leftrightarrow (x_{2,1,0} \wedge x_{2,2,0})$$

...

$$C_3 = \bigwedge_{m \in M} \left(\bigvee_{i=1}^N c_{i,m} \right)$$

$$c_{i,m} \leftrightarrow \bigwedge_{(P_j=k) \in m} x_{i,j,k}$$

更多的挑战

更多的参数和更复杂的约束 (highly configurable software)

Table 1: Comparing *SamplingCA* against *AutoCCAG*, *FastCA* and *TCA* on 20 selected instances.

Instances	#Options	#Constraints	SamplingCA		AutoCCAG		FastCA		TCA	
			min (avg)	time	min (avg)	size	time	min (avg)	time	min (avg)
adderII	1276	3206	101 (109.16)	33.37	166 (183.67)	1605.72	164 (183.82)	1266.56	203 (219.82)	948.99
at91sam7sek	1296	3921	98 (104.10)	32.96	154 (170.23)	1484.51	156 (170.25)	1482.87	187 (204.13)	958.29
busybox_1_28_0	998	962	54 (58.12)	11.57	95 (111.18)	611.19	95 (111.37)	567.30	125 (137.77)	512.78
dreamcast	1252	3168	108 (113.52)	33.41	172 (188.47)	1589.67	172 (188.75)	1195.18	199 (226.71)	918.31
e7t	1266	3816	103 (110.17)	32.87	163 (177.60)	1732.23	163 (177.60)	1543.14	200 (218.47)	937.28
ecos-icse11	1244	3146	103 (108.22)	31.04	260 (286.70)	1662.22	260 (287.23)	1962.32	328 (366.92)	927.75
embtoolkit	2128	15483	1107 (1119.48)	891.94	– (–)	–	– (–)	–	– (–)	–
financial	771	7241	4421 (4433.91)	544.31	5292 (5351.01)	2951.96	5292 (5351.04)	2904.89	5293 (5352.04)	2743.31
freebsd-icse11	1396	62183	116 (121.33)	43.94	164 (184.02)	3478.97	173 (188.62)	3552.38	211 (232.44)	2821.50
h8max	1202	3072	102 (108.07)	29.84	163 (182.41)	1496.94	164 (182.80)	1178.40	192 (219.10)	801.22
innovator	1256	50452	134 (140.95)	41.17	204 (225.73)	2454.67	206 (225.92)	2988.04	255 (270.43)	1556.81
integrator_arm9	1267	3939	116 (122.19)	36.94	197 (213.56)	2445.35	196 (213.70)	3143.44	233 (256.05)	1682.01
mpc50	1213	3728	96 (100.97)	28.20	141 (160.01)	1260.33	145 (160.15)	1220.16	177 (193.85)	819.19
ocelot	1266	3141	103 (107.63)	32.19	163 (181.75)	1618.54	165 (181.87)	1274.91	198 (221.31)	926.08
pc_i82544	1259	3179	106 (111.18)	33.15	169 (188.16)	1561.15	168 (188.65)	1189.46	204 (225.70)	940.95
ref4955	1218	3099	95 (102.02)	28.91	154 (168.99)	1548.75	155 (169.17)	1211.88	185 (202.64)	827.63
refidt334	1263	3140	108 (114.60)	34.47	173 (192.91)	1540.53	173 (193.23)	1324.82	217 (234.44)	985.29
se7751	1295	3254	113 (118.08)	36.94	182 (198.69)	1704.43	181 (198.72)	1396.61	213 (238.97)	1028.23
uClinux-config	11254	31637	63 (66.45)	1498.42	– (–)	–	– (–)	–	– (–)	–
XSEngine	1260	3803	102 (107.95)	32.50	159 (177.14)	1537.39	161 (177.30)	1389.57	191 (214.61)	926.49

更多的挑战

更高的覆盖强度

TABLE II

RESULTS OF *HSCA* AND ITS COMPETITORS FOR 4-WISE AND 5-WISE CCAG ON ALL 5 REAL-WORLD INSTANCES AND 5 SYNTHETIC INSTANCES.

Instance	<i>HSCA</i>		<i>AutoCCAG</i>		<i>CAmpactor</i>		<i>SamplingCA</i>		<i>PRBOT-its</i>	
	min. (avg.)	time	min. (avg.)	time	min. (avg.)	time	min. (avg.)	time	min. (avg.)	time
$t = 4$										
apache	734 (736.5)	971.7	-- (- -)	--	-- (- -)	--	-- (- -)	--	-- (- -)	--
bugzilla	166 (166.3)	444.6	167 (167.6)	390.3	171 (172.9)	867.6	272 (278.2)	50.4	195 (199.6)	601.3
gcc	340 (344.3)	969.5	-- (- -)	--	-- (- -)	--	-- (- -)	--	-- (- -)	--
spins	308 (308.0)	105.7	308 (310.0)	81.2	344 (356.9)	354.0	454 (462.2)	7.2	390 (399.2)	963.6
spinv	1,091 (1,094.2)	863.2	1,107 (1,111.4)	760.0	-- (- -)	--	-- (- -)	--	-- (- -)	--
Syn_2	624 (628.2)	920.9	676 (680.7)	950.7	-- (- -)	--	-- (- -)	--	-- (- -)	--
Syn_4	275 (276.6)	654.8	298 (299.3)	637.0	399 (405.6)	991.6	477 (482.6)	217.4	342 (348.3)	785.8
Syn_9	200 (200.0)	635.9	200 (200.0)	222.3	209 (211.5)	977.6	329 (334.9)	142.7	237 (242.0)	879.5
Syn_12	1,257 (1,262.5)	967.6	-- (- -)	--	-- (- -)	--	-- (- -)	--	-- (- -)	--
Syn_18	1,840 (1,845.5)	992.7	-- (- -)	--	-- (- -)	--	-- (- -)	--	-- (- -)	--
$t = 5$										
apache	3,643 (3,702.8)	9,996.6	-- (- -)	--	-- (- -)	--	-- (- -)	--	-- (- -)	--
bugzilla	555 (558.6)	6,747.6	559 (559.6)	7,905.7	741 (750.2)	9,947.6	856 (860.6)	2,467.5	660 (673.7)	9,452.1
gcc	1,403 (1,427.4)	9,983.7	-- (- -)	--	-- (- -)	--	-- (- -)	--	-- (- -)	--
spins	1,174 (1,174.0)	184.1	1,174 (1,174.0)	487.9	1,223 (1,232.3)	9,831.9	1,519 (1,533.4)	198.4	1,294 (1,306.7)	9,710.6
spinv	5,577 (5,586.2)	9,691.7	5,796 (5,847.0)	9,974.5	-- (- -)	--	-- (- -)	--	-- (- -)	--
Syn_2	2,810 (2,817.7)	9,530.3	-- (- -)	--	-- (- -)	--	-- (- -)	--	-- (- -)	--
Syn_4	984 (990.2)	9,765.2	1,022 (1,025.7)	8,576.4	-- (- -)	--	-- (- -)	--	-- (- -)	--
Syn_9	662 (667.0)	8,556.7	665 (667.4)	7,720.6	-- (- -)	--	-- (- -)	--	-- (- -)	--
Syn_12	6,332 (6,360.6)	9,988.4	-- (- -)	--	-- (- -)	--	-- (- -)	--	-- (- -)	--
Syn_18	10,821 (11,041.0)	9,998.3	-- (- -)	--	-- (- -)	--	-- (- -)	--	-- (- -)	--

重新思考组合测试的有效性

组合测试的核心在于对 t -way combination 的系统覆盖，但其有效性取决于

- 测试输入空间的建模方式 ?
 - 所有黑盒测试方法面临的共性问题
 - LLM 可以辅助，但高质量测试建模仍是一个挑战
- 覆盖强度的选择 ?
 - 虽然 2-way 是一个常见的选择，但未必适用所有待测软件
 - 一个 t -way 覆盖表 vs. 随机生成同样规模的一个测试用例集
 - 在 t -way 覆盖上的差异?
 - 对 $(t + k)$ -way 覆盖上的差异?

重新思考组合测试的有效性

组合测试的核心在于对 t -way combination 的系统覆盖，但其有效性取决于

- 测试预期输出 ?
 - 所有自动化测试方法的一个限制
 - 在某种程度上，“最小化 t -way 覆盖表规模”实际上就是在最小化人工判定测试结果的花费
- 测试生成和执行成本 ?
 - 如果按照测试成本 (e.g., time budget) 来作为停止条件
 - 把 CPU 时间花在“生成一个更小的覆盖表”上，还是花在“执行更多的 (随机) 测试输入”