Competence Centers for Excellent Technologies

Combinatorial Methods for Modelling Composed Software Systems

Ludwig Kampel, Bernhard Garn and Dimitris E. Simos

SBA Research, Austria

6th International Workshop on
Combinatorial Testing (IWCT 2017)
Waseda University, Nishiwaseda Campus, Tokyo, Japan
March 13, 2017

Outline of the Talk

Composed SUTs
- Motivation for Combinatorial Methods
- Nested Test Suites from Nested Models

SBA Research

Outline of the Talk

Composed SUTs
- Motivation for Combinatorial Methods
- Nested Test Suites from Nested Models

Modelling Composed SUTs
- The Case of Automotive Industry
- The Case of Linux Kernel API

SBA Research

Outline of the Talk

Composed SUTs
- Motivation for Combinatorial Methods
- Nested Test Suites from Nested Models

Modelling Composed SUTs
- The Case of Automotive Industry
- The Case of Linux Kernel API

Extending Bounds of Usability of CT
- Nested CA Construction
- Construction of Large CAs

SBA Research

Composed SUT

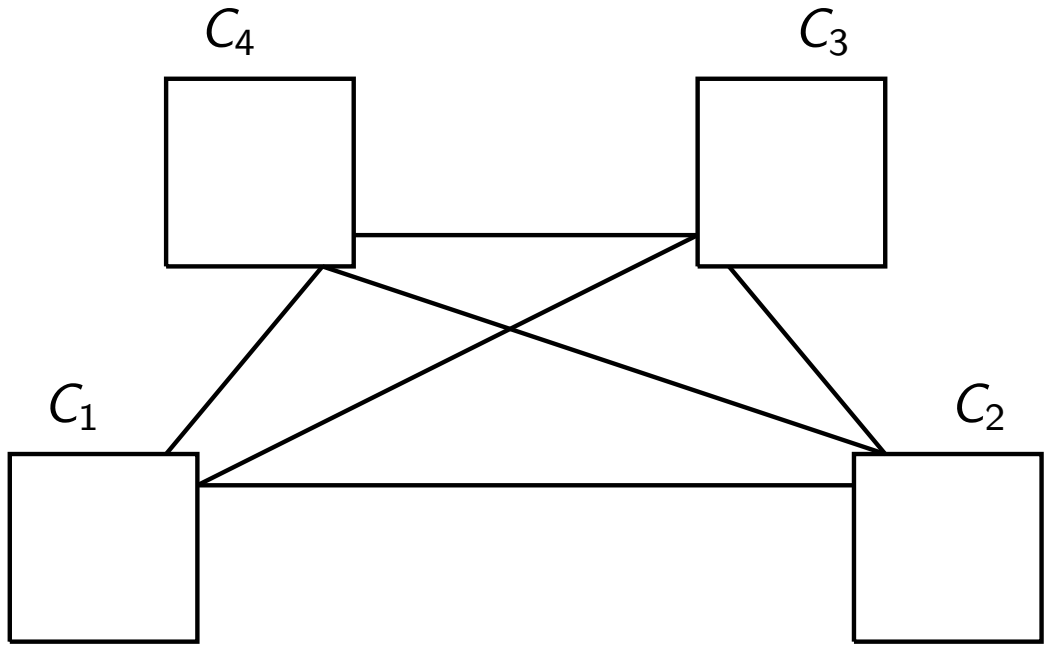


Figure: Composed SUT, with components C_1, C_2, C_3, C_4

$\mathcal{C}_1$	$\mathcal{C}_2$	$\mathcal{C}_3$	$\mathcal{C}_4$

Table: Corresponding IPM: 4 parameters

Composed SUT

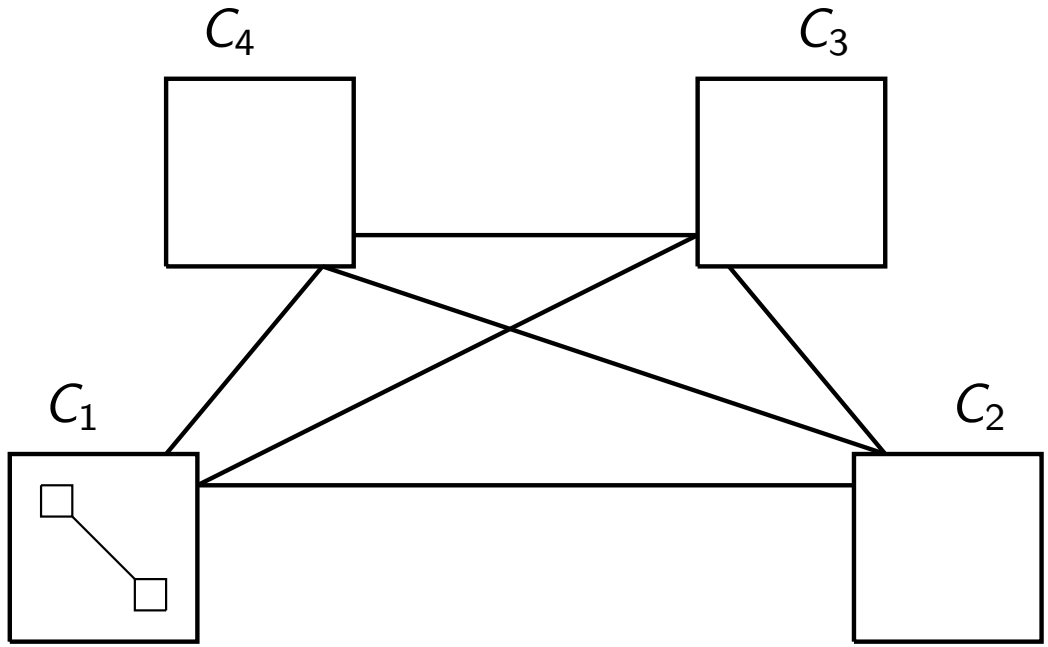


Figure: Composed SUT, with components C_1, C_2, C_3, C_4

$\mathcal{C}_1$	$\mathcal{C}_2$	$\mathcal{C}_3$	$\mathcal{C}_4$

Table: Corresponding IPM: 4 parameters

Composed SUT

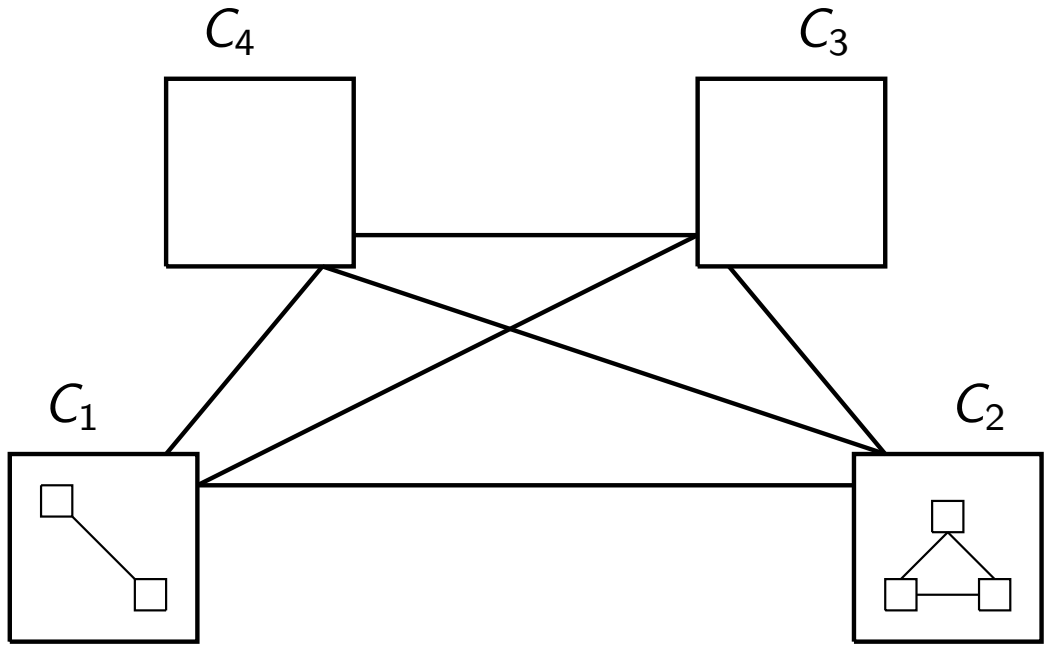


Figure: Composed SUT, with components C_1, C_2, C_3, C_4

$\mathcal{C}_1$	$\mathcal{C}_2$	$\mathcal{C}_3$	$\mathcal{C}_4$

Table: Corresponding IPM: 4 parameters

Composed SUT

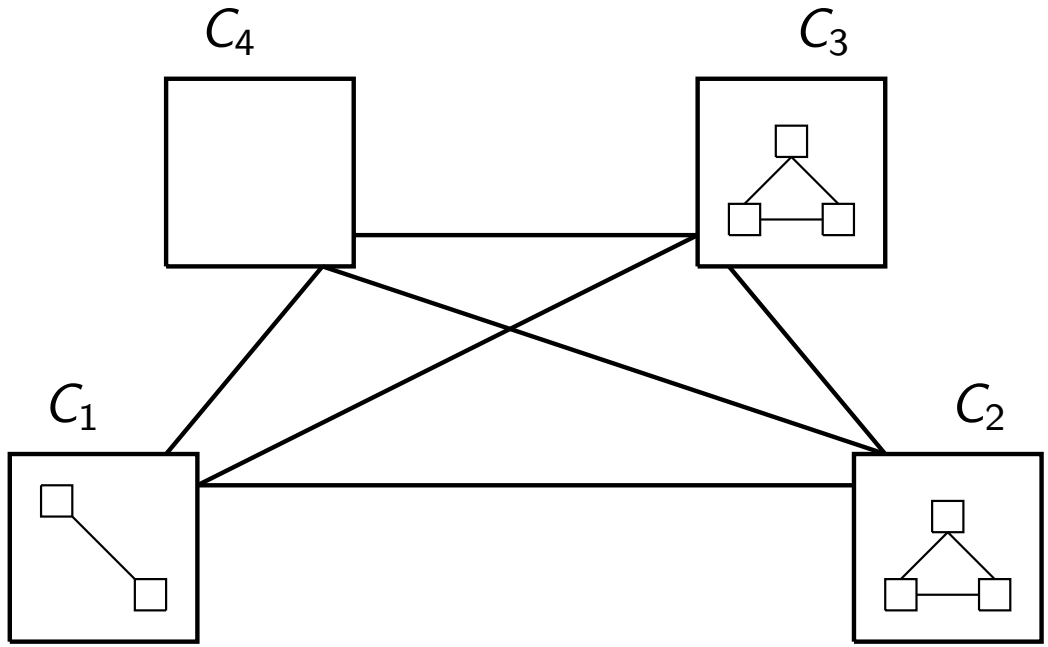


Figure: Composed SUT, with components C_1, C_2, C_3, C_4

$\mathcal{C}_1$	$\mathcal{C}_2$	$\mathcal{C}_3$	$\mathcal{C}_4$

Table: Corresponding IPM: 4 parameters

Composed SUT

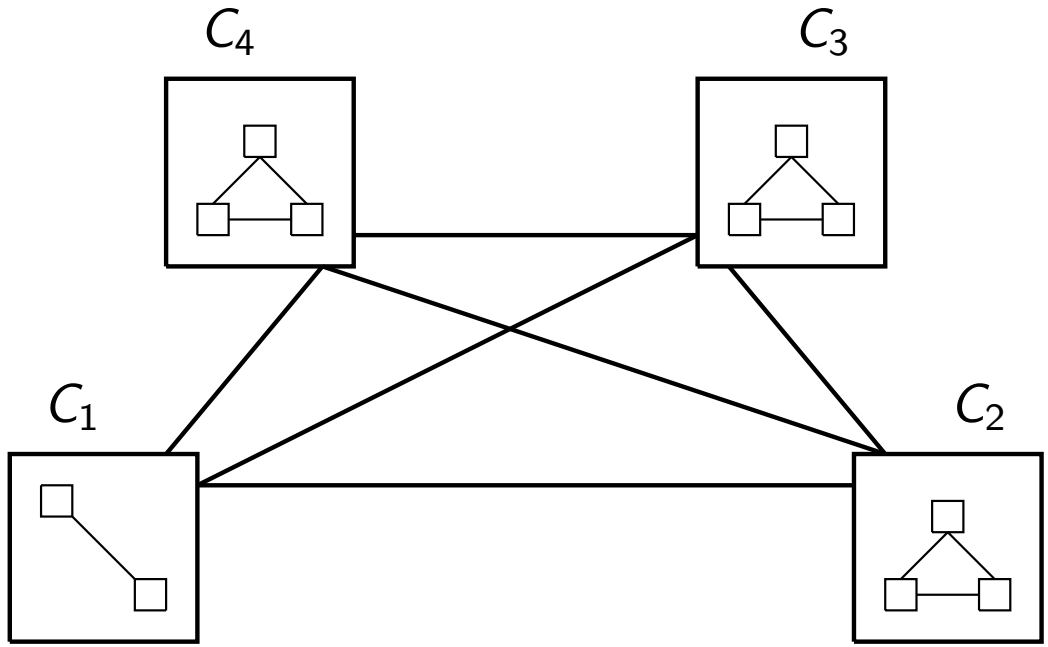


Figure: Composed SUT, with components C_1, C_2, C_3, C_4

$\mathcal{C}_1$	$\mathcal{C}_2$	$\mathcal{C}_3$	$\mathcal{C}_4$

Table: Corresponding IPM: 4 parameters

Composed SUT

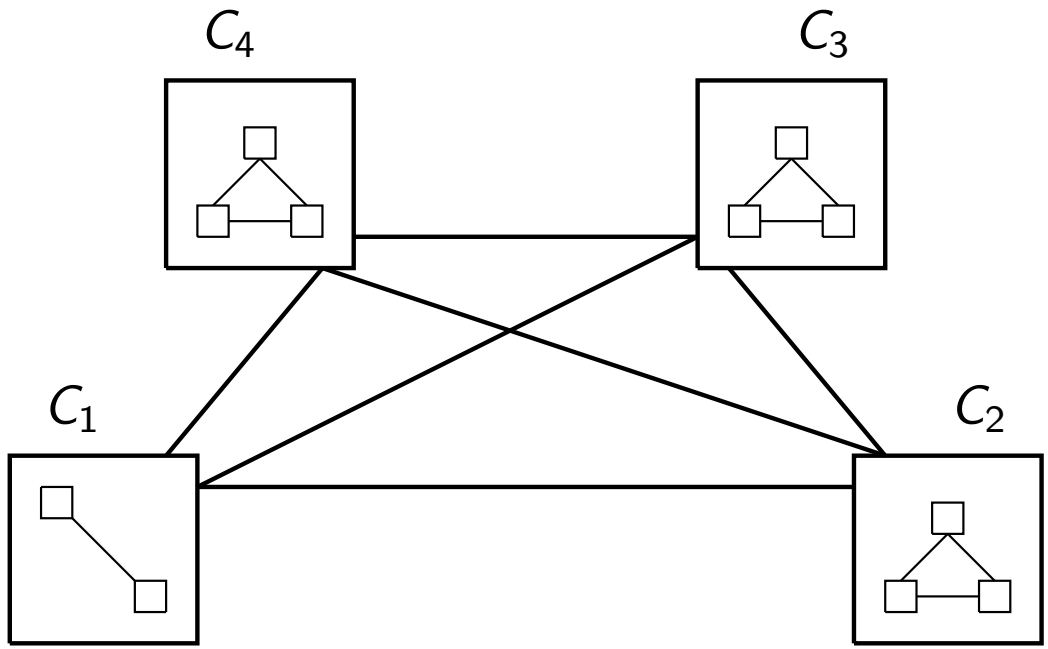


Figure: Composed SUT, with components C_1, C_2, C_3, C_4

C_1		C_2			C_3			C_4		
a_1	a_2	b_1	b_2	b_3	c_1	c_2	c_3	d_1	d_2	d_3

Table: Corresponding IPM: 4 vs. 11 parameters

Motivation for Combinatorial Methods

Modular Software Everywhere!

Modern software design relies on modular software architecture

- Software is structured in **components**
- These components interact with each other
- Unit testing: Components are tested individually
- Integration testing: Targets interactions between different components

Our Contribution

- Combinatorial modelling methodology that adapts to this modular approach of software design
- Combinatorial constructions to merge test suites of the components to a test suite for the whole SUT
- Formalized the concept of t-way coverage inheritance

SBA Research

Nested Test Suites from Nested Models

Methodology

1. Given a composed SUT with components $C_1, \ldots, C_k$
2. IPM of composed SUT is comprised of many different **sub-IPMs**
3. sub-IPMs have test suites $S_1, \ldots, S_k$
4. Test suites may or may not form a t-way CA
5. Combine the **test suites** to obtain a nested test suite for the SUT

t-way Coverage Inheritance Property

If the test suites are CAs of strength t, then the resulting nested test suite $\mathcal{R}$ is again a CA of strength t

- Formal statement (**proof**) and more general properties (see paper)

SBA Research

Nested Test Suites for Composed SUTs

Algorithmic Procedure

1. Select test suites $S_1, \ldots, S_k$ (seeds) and desired interaction strength t between the component $C_1, \ldots, C_k$
2. Compute meta array $\mathcal{M} := MCA(N; t, k, (|S_1|, \ldots, |S_k|))$ [a]
3. For each test suite S_i identify the tests with the values occurring in column i of $\mathcal{M}$, via a **bijective mapping**
4. Replace each value in $\mathcal{M}$ according to the mappings defined in Step 3

[a] $|S_i|$ denotes the number of test cases in the test suite $|S_i|$

Other Approaches

- Concept of subrelations (Cohen et al, 1997)
- Submodels and (informal) coverage preservation (Czerwonka, 2006)
- Concept Sub-attributes (Krishnan et al, 2007)

Composed SUT: Modern Car

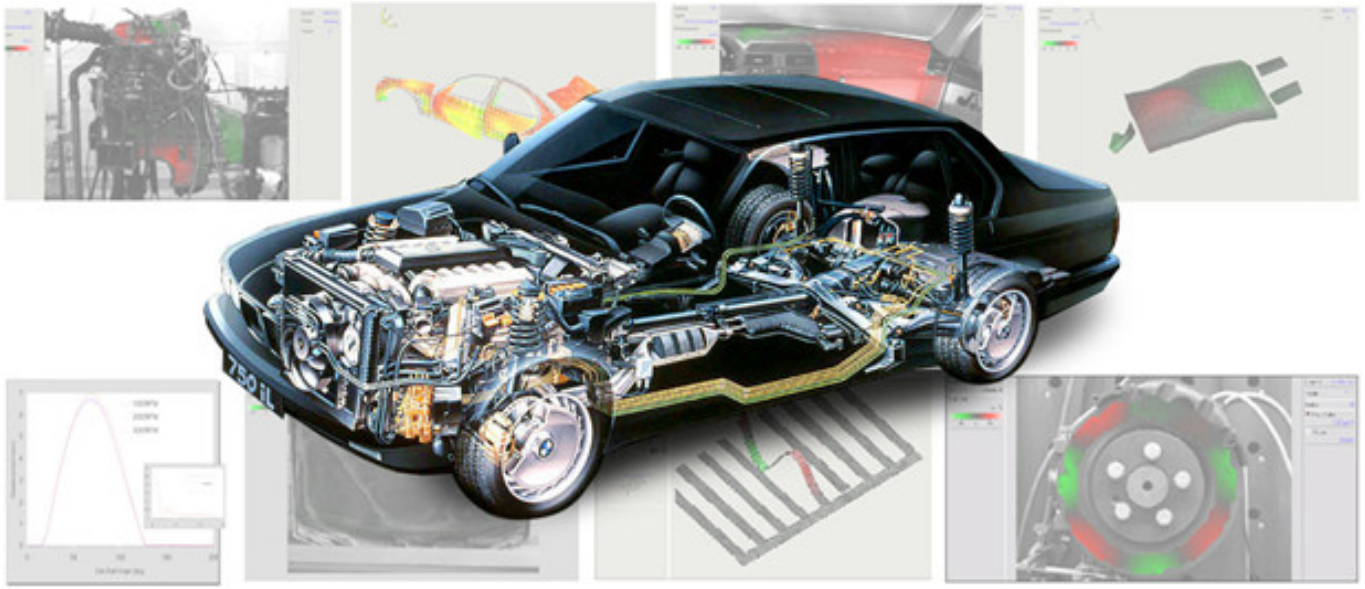
Components of the Car

C_1: Wheels modelled with two parameters

C_2: Engine modelled with three parameters

C_3: Infotainment modelled with three parameters

C_4: Communication modelled with three parameters



IPM of a component: Wheels

```
Name: IPM-Wheels

producer (enum): Giti (0), Mitas (1), Pirelli (2)
type  (boolean): winter (0), summer (1)
```

A	
a_1	a_2
0	0
0	1
1	0
1	1
2	0
2	1

Wheels	
producer	type
Giti	winter
Giti	summer
Mitas	winter
Mitas	summer
Pirelli	winter
Pirelli	summer

IPM of a component: Engine

```
Name: IPM-Engine

fuel       (boolean): diesel (0), gas (1)
drive_mode (boolean): eco (0), sport (1)
filter     (boolean): present (0), missing (1)
```

B		
b_1	b_2	b_3
0	0	1
0	1	0
1	0	0
1	1	1

Engine		
fuel	drive_mode	filter
diesel	eco	missing
diesel	sport	present
gas	eco	present
gas	sport	missing

IPM of a component: Infotainment system

```
Name: IPM-Infotainment

streaming      (boolean): true (0), false (1)
audio          (boolean): speaker (0), bluetooth (1)
remote_control (boolean): connected (0), disconnected (1)
```

C		
c_1	c_2	c_3
0	0	1
1	1	1
1	0	0
0	1	0

Infotainment		
streaming	audio	remote
true	speaker	disconnected
false	bluetooth	disconnected
false	speaker	connected
true	bluetooth	connected

IPM of a component: Communication

```
Name: IPM-Communication

ip_version (boolean): ipv4 (0), ipv6 (1)
connection (boolean): lte (0), wifi (1)
speed      (boolean): good (0), bad (1)
```

D		
d_1	d_2	d_3
0	0	1
0	1	0
1	0	0
1	1	1

Communication		
ip_version	connection	speed
ipv4	lte	bad
ipv4	wifi	good
ipv6	lte	good
ipv6	wifi	bad

Plug-In of Seeds into Meta Array $\mathcal{M}$

A	
a_1	a_2
0	0
0	1
1	0
1	1
2	0
2	1

B		
b_1	b_2	b_3
0	0	1
0	1	0
1	0	0
1	1	1

C		
c_1	c_2	c_3
0	0	1
1	1	1
1	0	0
0	1	0

D		
d_1	d_2	d_3
0	0	1
0	1	0
1	0	0
1	1	1

	$\mathcal{M}$			
#	M_1	M_2	M_3	M_4
1	0	0	1	1
2	0	1	2	2
3	0	2	3	3
4	0	3	0	0
5	1	0	2	3
6	1	1	3	0
7	1	2	0	1
8	1	3	1	2
9	2	0	3	2
10	2	1	0	3
11	2	2	1	0
12	2	3	2	1
13	3	0	0	2
14	3	1	1	3
15	3	2	2	0
16	3	3	3	1
17	4	0	0	0
18	4	1	1	1
19	4	2	2	2
20	4	3	3	3
21	5	0	0	0
22	5	1	1	1
23	5	2	2	2
24	5	3	3	3

Plug-In of Seeds into Meta Array $\mathcal{M}$

A	
a_1	a_2
0	0
0	1
1	0
1	1
2	0
2	1

B		
b_1	b_2	b_3
0	0	1
0	1	0
1	0	0
1	1	1

C		
c_1	c_2	c_3
0	0	1
1	1	1
1	0	0
0	1	0

D		
d_1	d_2	d_3
0	0	1
0	1	0
1	0	0
1	1	1

	$\mathcal{M}$			
#	M_1	M_2	M_3	M_4
1	0	0	1	1
2	0	1	2	2
3	0	2	3	3
4	0	3	0	0
5	1	0	2	3
6	1	1	3	0
7	1	2	0	1
8	1	3	1	2
9	2	0	3	2
10	2	1	0	3
11	2	2	1	0
12	2	3	2	1
13	3	0	0	2
14	3	1	1	3
15	3	2	2	0
16	3	3	3	1
17	4	0	0	0
18	4	1	1	1
19	4	2	2	2
20	4	3	3	3
21	5	0	0	0
22	5	1	1	1
23	5	2	2	2
24	5	3	3	3

Plug-In of Seeds into Meta Array $\mathcal{M}$

A	
a_1	a_2
0	0
0	1
1	0
1	1
2	0
2	1

B		
b_1	b_2	b_3
0	0	1
0	1	0
1	0	0
1	1	1

C		
c_1	c_2	c_3
0	0	1
1	1	1
1	0	0
0	1	0

D		
d_1	d_2	d_3
0	0	1
0	1	0
1	0	0
1	1	1

	$\mathcal{M}$			
#	M_1	M_2	M_3	M_4
1	0	0	1	1
2	0	1	2	2
3	0	2	3	3
4	0	3	0	0
5	1	0	2	3
6	1	1	3	0
7	1	2	0	1
8	1	3	1	2
9	2	0	3	2
10	2	1	0	3
11	2	2	1	0
12	2	3	2	1
13	3	0	0	2
14	3	1	1	3
15	3	2	2	0
16	3	3	3	1
17	4	0	0	0
18	4	1	1	1
19	4	2	2	2
20	4	3	3	3
21	5	0	0	0
22	5	1	1	1
23	5	2	2	2
24	5	3	3	3

Plug-In of Seeds into Meta Array $\mathcal{M}$

A	
a_1	a_2
0	0
0	1
1	0
1	1
2	0
2	1

B		
b_1	b_2	b_3
0	0	1
0	1	0
1	0	0
1	1	1

C		
c_1	c_2	c_3
0	0	1
1	1	1
1	0	0
0	1	0

D		
d_1	d_2	d_3
0	0	1
0	1	0
1	0	0
1	1	1

	$\mathcal{M}$			
#	M_1	M_2	M_3	M_4
1	0	0	1	1
2	0	1	2	2
3	0	2	3	3
4	0	3	0	0
5	1	0	2	3
6	1	1	3	0
7	1	2	0	1
8	1	3	1	2
9	2	0	3	2
10	2	1	0	3
11	2	2	1	0
12	2	3	2	1
13	3	0	0	2
14	3	1	1	3
15	3	2	2	0
16	3	3	3	1
17	4	0	0	0
18	4	1	1	1
19	4	2	2	2
20	4	3	3	3
21	5	0	0	0
22	5	1	1	1
23	5	2	2	2
24	5	3	3	3

Plug-In of Seeds into Meta Array $\mathcal{M}$

A	
a_1	a_2
0	0
0	1
1	0
1	1
2	0
2	1

B		
b_1	b_2	b_3
0	0	1
0	1	0
1	0	0
1	1	1

C		
c_1	c_2	c_3
0	0	1
1	1	1
1	0	0
0	1	0

D		
d_1	d_2	d_3
0	0	1
0	1	0
1	0	0
1	1	1

	$\mathcal{M}$			
#	M_1	M_2	M_3	M_4
1	0	0	1	1
2	0	1	2	2
3	0	2	3	3
4	0	3	0	0
5	1	0	2	3
6	1	1	3	0
7	1	2	0	1
8	1	3	1	2
9	2	0	3	2
10	2	1	0	3
11	2	2	1	0
12	2	3	2	1
13	3	0	0	2
14	3	1	1	3
15	3	2	2	0
16	3	3	3	1
17	4	0	0	0
18	4	1	1	1
19	4	2	2	2
20	4	3	3	3
21	5	0	0	0
22	5	1	1	1
23	5	2	2	2
24	5	3	3	3

Plug-In of Seeds into Meta Array $\mathcal{M}$

A	
a_1	a_2
0	0
0	1
1	0
1	1
2	0
2	1

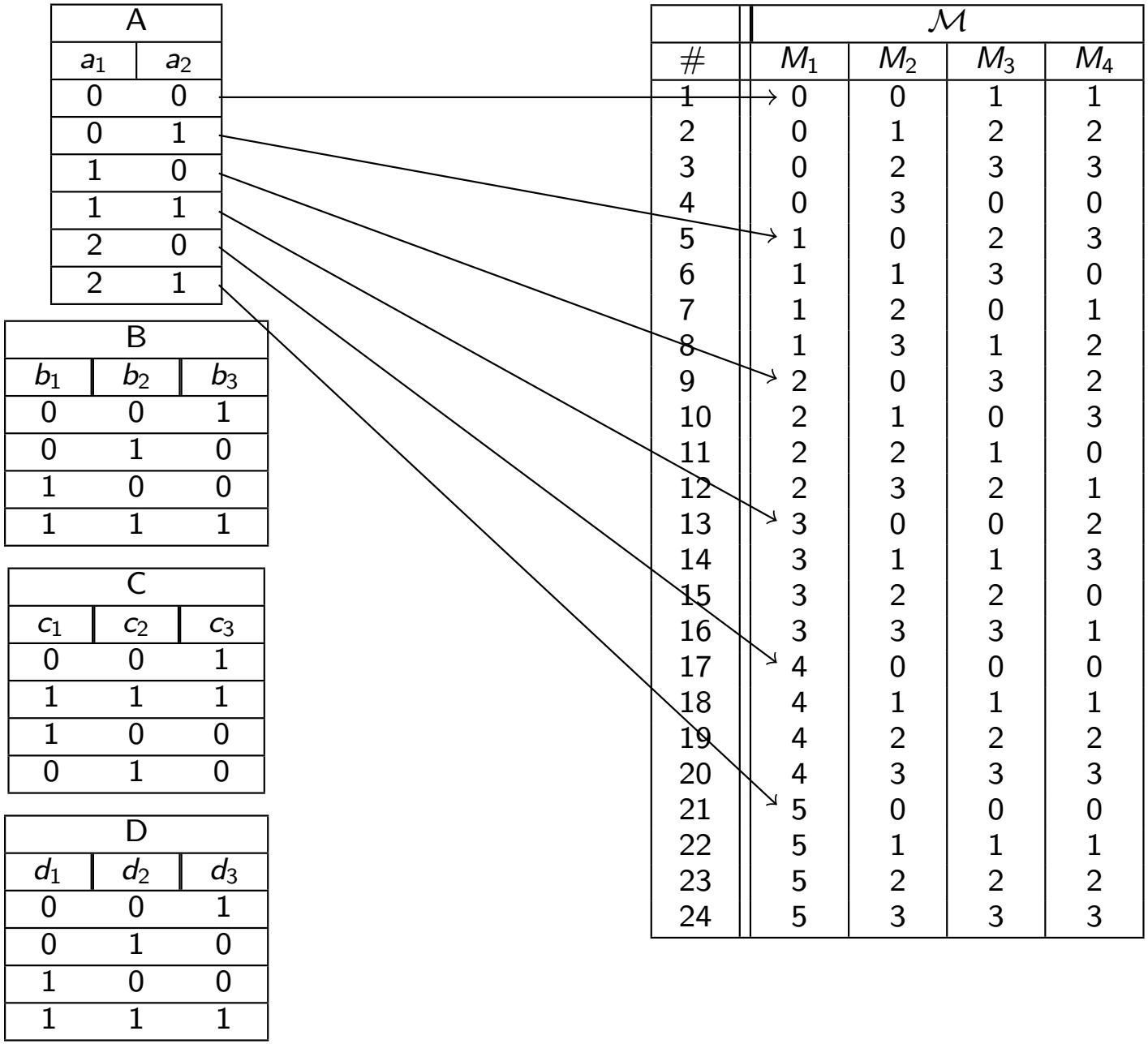
B		
b_1	b_2	b_3
0	0	1
0	1	0
1	0	0
1	1	1

C		
c_1	c_2	c_3
0	0	1
1	1	1
1	0	0
0	1	0

D		
d_1	d_2	d_3
0	0	1
0	1	0
1	0	0
1	1	1

	$\mathcal{M}$			
#	M_1	M_2	M_3	M_4
1	0	0	1	1
2	0	1	2	2
3	0	2	3	3
4	0	3	0	0
5	1	0	2	3
6	1	1	3	0
7	1	2	0	1
8	1	3	1	2
9	2	0	3	2
10	2	1	0	3
11	2	2	1	0
12	2	3	2	1
13	3	0	0	2
14	3	1	1	3
15	3	2	2	0
16	3	3	3	1
17	4	0	0	0
18	4	1	1	1
19	4	2	2	2
20	4	3	3	3
21	5	0	0	0
22	5	1	1	1
23	5	2	2	2
24	5	3	3	3

Plug-In of Seeds into Meta Array $\mathcal{M}$



A	
a_1	a_2
0	0
0	1
1	0
1	1
2	0
2	1

B		
b_1	b_2	b_3
0	0	1
0	1	0
1	0	0
1	1	1

C		
c_1	c_2	c_3
0	0	1
1	1	1
1	0	0
0	1	0

D		
d_1	d_2	d_3
0	0	1
0	1	0
1	0	0
1	1	1

	$\mathcal{M}$			
#	M_1	M_2	M_3	M_4
1	0	0	1	1
2	0	1	2	2
3	0	2	3	3
4	0	3	0	0
5	1	0	2	3
6	1	1	3	0
7	1	2	0	1
8	1	3	1	2
9	2	0	3	2
10	2	1	0	3
11	2	2	1	0
12	2	3	2	1
13	3	0	0	2
14	3	1	1	3
15	3	2	2	0
16	3	3	3	1
17	4	0	0	0
18	4	1	1	1
19	4	2	2	2
20	4	3	3	3
21	5	0	0	0
22	5	1	1	1
23	5	2	2	2
24	5	3	3	3

Plug-In of Seeds into Meta Array $\mathcal{M}$

A	
a_1	a_2
0	0
0	1
1	0
1	1
2	0
2	1

B		
b_1	b_2	b_3
0	0	1
0	1	0
1	0	0
1	1	1

C		
c_1	c_2	c_3
0	0	1
1	1	1
1	0	0
0	1	0

D		
d_1	d_2	d_3
0	0	1
0	1	0
1	0	0
1	1	1

	$\mathcal{M}$			
#	M_1	M_2	M_3	M_4
1	0	0	1	1
2	0	1	2	2
3	0	2	3	3
4	0	3	0	0
5	1	0	2	3
6	1	1	3	0
7	1	2	0	1
8	1	3	1	2
9	2	0	3	2
10	2	1	0	3
11	2	2	1	0
12	2	3	2	1
13	3	0	0	2
14	3	1	1	3
15	3	2	2	0
16	3	3	3	1
17	4	0	0	0
18	4	1	1	1
19	4	2	2	2
20	4	3	3	3
21	5	0	0	0
22	5	1	1	1
23	5	2	2	2
24	5	3	3	3

Plug-In of Seeds into Meta Array $\mathcal{M}$

A	
a_1	a_2
0	0
0	1
1	0
1	1
2	0
2	1

B		
b_1	b_2	b_3
0	0	1
0	1	0
1	0	0
1	1	1

C		
c_1	c_2	c_3
0	0	1
1	1	1
1	0	0
0	1	0

D		
d_1	d_2	d_3
0	0	1
0	1	0
1	0	0
1	1	1

	$\mathcal{M}$			
#	M_1	M_2	M_3	M_4
1	0	0	1	1
2	0	1	2	2
3	0	2	3	3
4	0	3	0	0
5	1	0	2	3
6	1	1	3	0
7	1	2	0	1
8	1	3	1	2
9	2	0	3	2
10	2	1	0	3
11	2	2	1	0
12	2	3	2	1
13	3	0	0	2
14	3	1	1	3
15	3	2	2	0
16	3	3	3	1
17	4	0	0	0
18	4	1	1	1
19	4	2	2	2
20	4	3	3	3
21	5	0	0	0
22	5	1	1	1
23	5	2	2	2
24	5	3	3	3

Plug-In of Seeds into Meta Array $\mathcal{M}$

A	
a_1	a_2
0	0
0	1
1	0
1	1
2	0
2	1

B		
b_1	b_2	b_3
0	0	1
0	1	0
1	0	0
1	1	1

C		
c_1	c_2	c_3
0	0	1
1	1	1
1	0	0
0	1	0

D		
d_1	d_2	d_3
0	0	1
0	1	0
1	0	0
1	1	1

	$\mathcal{M}$			
#	M_1	M_2	M_3	M_4
1	0	0	1	1
2	0	1	2	2
3	0	2	3	3
4	0	3	0	0
5	1	0	2	3
6	1	1	3	0
7	1	2	0	1
8	1	3	1	2
9	2	0	3	2
10	2	1	0	3
11	2	2	1	0
12	2	3	2	1
13	3	0	0	2
14	3	1	1	3
15	3	2	2	0
16	3	3	3	1
17	4	0	0	0
18	4	1	1	1
19	4	2	2	2
20	4	3	3	3
21	5	0	0	0
22	5	1	1	1
23	5	2	2	2
24	5	3	3	3

Plug-In of Seeds into Meta Array $\mathcal{M}$

A	
a_1	a_2
0	0
0	1
1	0
1	1
2	0
2	1

B		
b_1	b_2	b_3
0	0	1
0	1	0
1	0	0
1	1	1

C		
c_1	c_2	c_3
0	0	1
1	1	1
1	0	0
0	1	0

D		
d_1	d_2	d_3
0	0	1
0	1	0
1	0	0
1	1	1

	$\mathcal{M}$			
#	M_1	M_2	M_3	M_4
1	0	0	1	1
2	0	1	2	2
3	0	2	3	3
4	0	3	0	0
5	1	0	2	3
6	1	1	3	0
7	1	2	0	1
8	1	3	1	2
9	2	0	3	2
10	2	1	0	3
11	2	2	1	0
12	2	3	2	1
13	3	0	0	2
14	3	1	1	3
15	3	2	2	0
16	3	3	3	1
17	4	0	0	0
18	4	1	1	1
19	4	2	2	2
20	4	3	3	3
21	5	0	0	0
22	5	1	1	1
23	5	2	2	2
24	5	3	3	3

Plug-In of Seeds into Meta Array $\mathcal{M}$

A	
a_1	a_2
0	0
0	1
1	0
1	1
2	0
2	1

B		
b_1	b_2	b_3
0	0	1
0	1	0
1	0	0
1	1	1

C		
c_1	c_2	c_3
0	0	1
1	1	1
1	0	0
0	1	0

D		
d_1	d_2	d_3
0	0	1
0	1	0
1	0	0
1	1	1

	$\mathcal{M}$			
#	M_1	M_2	M_3	M_4
1	0	0	1	1
2	0	1	2	2
3	0	2	3	3
4	0	3	0	0
5	1	0	2	3
6	1	1	3	0
7	1	2	0	1
8	1	3	1	2
9	2	0	3	2
10	2	1	0	3
11	2	2	1	0
12	2	3	2	1
13	3	0	0	2
14	3	1	1	3
15	3	2	2	0
16	3	3	3	1
17	4	0	0	0
18	4	1	1	1
19	4	2	2	2
20	4	3	3	3
21	5	0	0	0
22	5	1	1	1
23	5	2	2	2
24	5	3	3	3

Plug-In of Seeds into Meta Array $\mathcal{M}$

A	
a_1	a_2
0	0
0	1
1	0
1	1
2	0
2	1

B		
b_1	b_2	b_3
0	0	1
0	1	0
1	0	0
1	1	1

C		
c_1	c_2	c_3
0	0	1
1	1	1
1	0	0
0	1	0

D		
d_1	d_2	d_3
0	0	1
0	1	0
1	0	0
1	1	1

	$\mathcal{M}$			
#	M_1	M_2	M_3	M_4
1	0	0	1	1
2	0	1	2	2
3	0	2	3	3
4	0	3	0	0
5	1	0	2	3
6	1	1	3	0
7	1	2	0	1
8	1	3	1	2
9	2	0	3	2
10	2	1	0	3
11	2	2	1	0
12	2	3	2	1
13	3	0	0	2
14	3	1	1	3
15	3	2	2	0
16	3	3	3	1
17	4	0	0	0
18	4	1	1	1
19	4	2	2	2
20	4	3	3	3
21	5	0	0	0
22	5	1	1	1
23	5	2	2	2
24	5	3	3	3

Plug-In of Seeds into Meta Array $\mathcal{M}$

A	
a_1	a_2
0	0
0	1
1	0
1	1
2	0
2	1

B		
b_1	b_2	b_3
0	0	1
0	1	0
1	0	0
1	1	1

C		
c_1	c_2	c_3
0	0	1
1	1	1
1	0	0
0	1	0

D		
d_1	d_2	d_3
0	0	1
0	1	0
1	0	0
1	1	1

	$\mathcal{M}$			
#	M_1	M_2	M_3	M_4
1	0	0	1	1
2	0	1	2	2
3	0	2	3	3
4	0	3	0	0
5	1	0	2	3
6	1	1	3	0
7	1	2	0	1
8	1	3	1	2
9	2	0	3	2
10	2	1	0	3
11	2	2	1	0
12	2	3	2	1
13	3	0	0	2
14	3	1	1	3
15	3	2	2	0
16	3	3	3	1
17	4	0	0	0
18	4	1	1	1
19	4	2	2	2
20	4	3	3	3
21	5	0	0	0
22	5	1	1	1
23	5	2	2	2
24	5	3	3	3

Plug-In of Seeds into Meta Array $\mathcal{M}$

A	
a_1	a_2
0	0
0	1
1	0
1	1
2	0
2	1

B		
b_1	b_2	b_3
0	0	1
0	1	0
1	0	0
1	1	1

C		
c_1	c_2	c_3
0	0	1
1	1	1
1	0	0
0	1	0

D		
d_1	d_2	d_3
0	0	1
0	1	0
1	0	0
1	1	1

	$\mathcal{M}$			
#	M_1	M_2	M_3	M_4
1	0	0	1	1
2	0	1	2	2
3	0	2	3	3
4	0	3	0	0
5	1	0	2	3
6	1	1	3	0
7	1	2	0	1
8	1	3	1	2
9	2	0	3	2
10	2	1	0	3
11	2	2	1	0
12	2	3	2	1
13	3	0	0	2
14	3	1	1	3
15	3	2	2	0
16	3	3	3	1
17	4	0	0	0
18	4	1	1	1
19	4	2	2	2
20	4	3	3	3
21	5	0	0	0
22	5	1	1	1
23	5	2	2	2
24	5	3	3	3

Plug-In of Seeds into Meta Array $\mathcal{M}$

A	
a_1	a_2
0	0
0	1
1	0
1	1
2	0
2	1

B		
b_1	b_2	b_3
0	0	1
0	1	0
1	0	0
1	1	1

C		
c_1	c_2	c_3
0	0	1
1	1	1
1	0	0
0	1	0

D		
d_1	d_2	d_3
0	0	1
0	1	0
1	0	0
1	1	1

	$\mathcal{M}$			
#	M_1	M_2	M_3	M_4
1	0	0	1	1
2	0	1	2	2
3	0	2	3	3
4	0	3	0	0
5	1	0	2	3
6	1	1	3	0
7	1	2	0	1
8	1	3	1	2
9	2	0	3	2
10	2	1	0	3
11	2	2	1	0
12	2	3	2	1
13	3	0	0	2
14	3	1	1	3
15	3	2	2	0
16	3	3	3	1
17	4	0	0	0
18	4	1	1	1
19	4	2	2	2
20	4	3	3	3
21	5	0	0	0
22	5	1	1	1
23	5	2	2	2
24	5	3	3	3

Plug-In of Seeds into Meta Array $\mathcal{M}$

A	
a_1	a_2
0	0
0	1
1	0
1	1
2	0
2	1

B		
b_1	b_2	b_3
0	0	1
0	1	0
1	0	0
1	1	1

C		
c_1	c_2	c_3
0	0	1
1	1	1
1	0	0
0	1	0

D		
d_1	d_2	d_3
0	0	1
0	1	0
1	0	0
1	1	1

	$\mathcal{M}$			
#	M_1	M_2	M_3	M_4
1	0	0	1	1
2	0	1	2	2
3	0	2	3	3
4	0	3	0	0
5	1	0	2	3
6	1	1	3	0
7	1	2	0	1
8	1	3	1	2
9	2	0	3	2
10	2	1	0	3
11	2	2	1	0
12	2	3	2	1
13	3	0	0	2
14	3	1	1	3
15	3	2	2	0
16	3	3	3	1
17	4	0	0	0
18	4	1	1	1
19	4	2	2	2
20	4	3	3	3
21	5	0	0	0
22	5	1	1	1
23	5	2	2	2
24	5	3	3	3

Plug-In of Seeds into Meta Array $\mathcal{M}$

A	
a_1	a_2
0	0
0	1
1	0
1	1
2	0
2	1

B		
b_1	b_2	b_3
0	0	1
0	1	0
1	0	0
1	1	1

C		
c_1	c_2	c_3
0	0	1
1	1	1
1	0	0
0	1	0

D		
d_1	d_2	d_3
0	0	1
0	1	0
1	0	0
1	1	1

	$\mathcal{M}$			
#	M_1	M_2	M_3	M_4
1	0	0	1	1
2	0	1	2	2
3	0	2	3	3
4	0	3	0	0
5	1	0	2	3
6	1	1	3	0
7	1	2	0	1
8	1	3	1	2
9	2	0	3	2
10	2	1	0	3
11	2	2	1	0
12	2	3	2	1
13	3	0	0	2
14	3	1	1	3
15	3	2	2	0
16	3	3	3	1
17	4	0	0	0
18	4	1	1	1
19	4	2	2	2
20	4	3	3	3
21	5	0	0	0
22	5	1	1	1
23	5	2	2	2
24	5	3	3	3

Plug-In of Seeds into Meta Array $\mathcal{M}$

A	
a_1	a_2
0	0
0	1
1	0
1	1
2	0
2	1

B		
b_1	b_2	b_3
0	0	1
0	1	0
1	0	0
1	1	1

C		
c_1	c_2	c_3
0	0	1
1	1	1
1	0	0
0	1	0

D		
d_1	d_2	d_3
0	0	1
0	1	0
1	0	0
1	1	1

	$\mathcal{M}$			
#	M_1	M_2	M_3	M_4
1	0	0	1	1
2	0	1	2	2
3	0	2	3	3
4	0	3	0	0
5	1	0	2	3
6	1	1	3	0
7	1	2	0	1
8	1	3	1	2
9	2	0	3	2
10	2	1	0	3
11	2	2	1	0
12	2	3	2	1
13	3	0	0	2
14	3	1	1	3
15	3	2	2	0
16	3	3	3	1
17	4	0	0	0
18	4	1	1	1
19	4	2	2	2
20	4	3	3	3
21	5	0	0	0
22	5	1	1	1
23	5	2	2	2
24	5	3	3	3

Plug-In of Seeds into Meta Array $\mathcal{M}$

A	
a_1	a_2
0	0
0	1
1	0
1	1
2	0
2	1

B		
b_1	b_2	b_3
0	0	1
0	1	0
1	0	0
1	1	1

C		
c_1	c_2	c_3
0	0	1
1	1	1
1	0	0
0	1	0

D		
d_1	d_2	d_3
0	0	1
0	1	0
1	0	0
1	1	1

	$\mathcal{M}$			
#	M_1	M_2	M_3	M_4
1	0	0	1	1
2	0	1	2	2
3	0	2	3	3
4	0	3	0	0
5	1	0	2	3
6	1	1	3	0
7	1	2	0	1
8	1	3	1	2
9	2	0	3	2
10	2	1	0	3
11	2	2	1	0
12	2	3	2	1
13	3	0	0	2
14	3	1	1	3
15	3	2	2	0
16	3	3	3	1
17	4	0	0	0
18	4	1	1	1
19	4	2	2	2
20	4	3	3	3
21	5	0	0	0
22	5	1	1	1
23	5	2	2	2
24	5	3	3	3

Plug-In of Seeds into Meta Array $\mathcal{M}$

A	
a_1	a_2
0	0
0	1
1	0
1	1
2	0
2	1

B		
b_1	b_2	b_3
0	0	1
0	1	0
1	0	0
1	1	1

C		
c_1	c_2	c_3
0	0	1
1	1	1
1	0	0
0	1	0

D		
d_1	d_2	d_3
0	0	1
0	1	0
1	0	0
1	1	1

	$\mathcal{M}$			
#	M_1	M_2	M_3	M_4
1	0	0	1	1
2	0	1	2	2
3	0	2	3	3
4	0	3	0	0
5	1	0	2	3
6	1	1	3	0
7	1	2	0	1
8	1	3	1	2
9	2	0	3	2
10	2	1	0	3
11	2	2	1	0
12	2	3	2	1
13	3	0	0	2
14	3	1	1	3
15	3	2	2	0
16	3	3	3	1
17	4	0	0	0
18	4	1	1	1
19	4	2	2	2
20	4	3	3	3
21	5	0	0	0
22	5	1	1	1
23	5	2	2	2
24	5	3	3	3

Plug-In of Seeds into Meta Array $\mathcal{M}$

A	
a_1	a_2
0	0
0	1
1	0
1	1
2	0
2	1

B		
b_1	b_2	b_3
0	0	1
0	1	0
1	0	0
1	1	1

C		
c_1	c_2	c_3
0	0	1
1	1	1
1	0	0
0	1	0

D		
d_1	d_2	d_3
0	0	1
0	1	0
1	0	0
1	1	1

	$\mathcal{M}$			
#	M_1	M_2	M_3	M_4
1	0	0	1	1
2	0	1	2	2
3	0	2	3	3
4	0	3	0	0
5	1	0	2	3
6	1	1	3	0
7	1	2	0	1
8	1	3	1	2
9	2	0	3	2
10	2	1	0	3
11	2	2	1	0
12	2	3	2	1
13	3	0	0	2
14	3	1	1	3
15	3	2	2	0
16	3	3	3	1
17	4	0	0	0
18	4	1	1	1
19	4	2	2	2
20	4	3	3	3
21	5	0	0	0
22	5	1	1	1
23	5	2	2	2
24	5	3	3	3

Resulting Array $\mathcal{R}$ where $|\mathcal{R}| = |\mathcal{M}|$

$\mathcal{R}$										
a_1	a_2	b_1	b_2	b_3	c_1	c_2	c_3	d_1	d_2	d_3
0	0	0	0	1	1	1	1	0	1	0
0	0	0	1	0	1	0	0	1	0	0
0	0	1	0	0	0	1	0	1	1	1
0	0	1	1	1	0	0	1	0	0	1
0	1	0	0	1	1	0	0	1	1	1
0	1	0	1	0	0	1	0	0	0	1
0	1	1	0	0	0	0	1	0	1	0
0	1	1	1	1	1	1	1	1	0	0
1	0	0	0	1	0	1	0	1	0	0
1	0	0	1	0	0	0	1	1	1	1
1	0	1	0	0	1	1	1	0	0	1
1	0	1	1	1	1	0	0	0	1	0
1	1	0	0	1	0	0	1	1	0	0
1	1	0	1	0	1	1	1	1	1	1
1	1	1	0	0	1	0	0	0	0	1
1	1	1	1	1	0	1	0	0	1	0
2	0	0	0	1	0	0	1	0	0	1
2	0	0	1	0	1	1	1	0	1	0
2	0	1	0	0	1	0	0	1	0	0
2	0	1	1	1	0	1	0	1	1	1
2	1	0	0	1	0	0	1	0	0	1
2	1	0	1	0	1	1	1	0	1	0
2	1	1	0	0	1	0	0	1	0	0
2	1	1	1	1	0	1	0	1	1	1

An $MCA(24; 2, 11, (3, 2^{10}))$

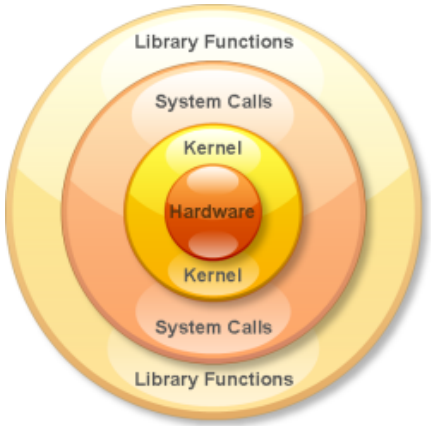
IPMs for Linux Kernel Systems Calls

SUTs: System Call API of Linux kernel

- Is the interface between the kernel and user space
- Is mostly used via wrappers in the standard library of the programming languages
- Is used to request, monitor and control system resources

Goal

Reliability and quality assurance of kernel software



SBA Research

Linux Kernel System Call API: chmod System Call

Example: chmod(pathname, mode)

- The `chmod()` system call changes the permissions of a file
- It has two arguments:
  1. **pathname**: Path to a file called
  2. **mode**: New file permissions specified in a bitmask
- Mode is created by **ORing** together zero or more of the following flags: `S_ISUID`, `S_ISGID`, `S_ISVTX`, `S_IRUSR`, `S_IWUSR`, `S_IXUSR`, `S_IRGRP`, `S_IWGRP`, `S_IXGRP`, `S_IROTH`, `S_IWOTH`, `S_IXOTH`
- Flags can be modelled as boolean parameters (TRUE / FALSE)

Towards Nested Models for the System Call chmod

Incorporating semantics – new Nested IPM

- The mode argument can be regarded as a component, having its own internal structure
- Model mode with four sub-attributes:

P_{user}: Captures semantically all permission masks for 'user': `S_IRUSR, S_IWUSR, S_IXUSR`

P_{grp}: Captures semantically all permission masks for 'group': `S_IRGRP, S_IWGRP, S_IXGRP`

P_{other}: Captures semantically all permission masks for 'other': `S_IROTH, S_IWOTH, S_IXOTH`

P_{sys}: Captures semantically the masks `S_ISUID, S_ISGID, S_ISVTX`

Nested Models for the System Call API chmod

How to choose tests for P_{user}, P_{grp}, P_{other}, P_{sys}?

Let P be a sub-attribute of chmod-IPM-NEST, having three binary parameters:

- Use full Cartesian product of the values of three binary parameters to produce all possible triples to form a seed array, leading to eight test cases
- Use a 2-way test suite for three binary parameters as a seed array, leading to four test cases
- Use a non-empty proper subset of the Cartesian product as legacy test suite; Reasons for making this choice include the reuse of an existing test suite

Depending on the testing requirements, different choices can be made P_{user}, P_{grp}, P_{other}, P_{sys}

Nested Models for the System Call API chmod

```
Name: ipm-P_sys

is_uid (boolean): true (0), false (1)
is_gid (boolean): true (0), false (1)
is_vtx (boolean): true (0), false (1)
```

A_1		
a_1	a_2	a_3
0	0	0
1	0	1
0	1	1

Figure: Tests selected by expert knowledge.

Nested Models for the System Call API chmod

```
Name: ipm-P_usr

u_r (boolean): true (0), false (1)
u_w (boolean): true (0), false (1)
u_x (boolean): true (0), false (1)
```

B_1		
b_1	b_2	b_3
0	0	0
1	1	1

Figure: Tests selected by expert knowledge.

Nested Models for the System Call API chmod

```
Name: ipm-P_grp

g_r (boolean): true (0), false (1)
g_w (boolean): true (0), false (1)
g_x (boolean): true (0), false (1)
```

C_1		
c_1	c_2	c_3
0	0	0
1	1	1

Figure: Tests selected by expert knowledge.

Nested Models for the System Call API chmod

```
Name: ipm-P_oth

o_r (boolean): true (0), false (1)
o_w (boolean): true (0), false (1)
o_x (boolean): true (0), false (1)
```

D_1		
d_1	d_2	d_3
0	0	0
1	1	1

Figure: Tests selected by expert knowledge.

Plug-In of Seeds into Meta Array

A_1		
a_1	a_2	a_3
0	0	0
1	0	1
0	1	1

B_1		
b_1	b_2	b_3
0	0	0
1	1	1

C_1		
c_1	c_2	c_3
0	0	0
1	1	1

D_1		
d_1	d_2	d_3
0	0	0
1	1	1

	$\mathcal{M}$			
#	M_1	M_2	M_3	M_4
1	0	0	1	1
2	0	1	0	0
3	1	0	0	1
4	1	1	1	0
5	2	0	0	0
6	2	1	1	1

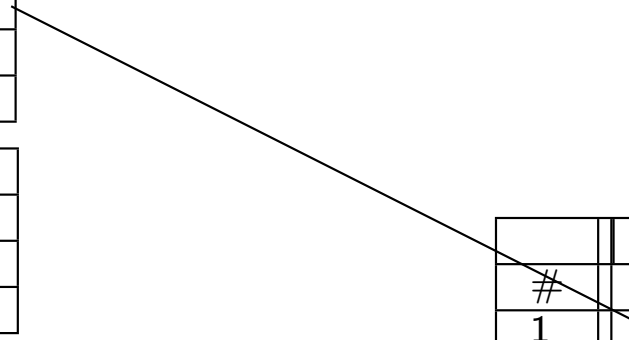
Plug-In of Seeds into Meta Array

A_1		
a_1	a_2	a_3
0	0	0
1	0	1
0	1	1

B_1		
b_1	b_2	b_3
0	0	0
1	1	1

C_1		
c_1	c_2	c_3
0	0	0
1	1	1

D_1		
d_1	d_2	d_3
0	0	0
1	1	1



	$\mathcal{M}$			
#	M_1	M_2	M_3	M_4
1	0	0	1	1
2	0	1	0	0
3	1	0	0	1
4	1	1	1	0
5	2	0	0	0
6	2	1	1	1

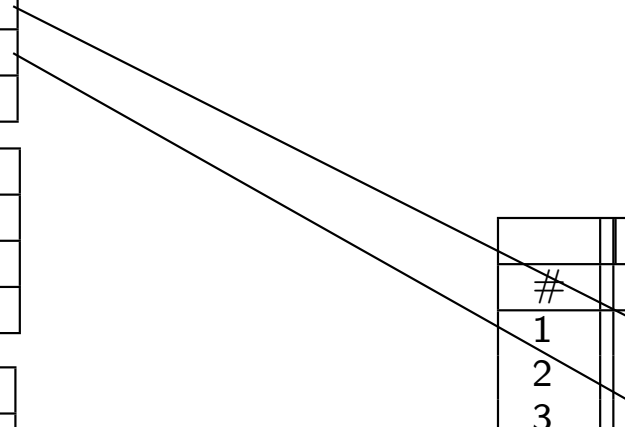
Plug-In of Seeds into Meta Array

A_1		
a_1	a_2	a_3
0	0	0
1	0	1
0	1	1

B_1		
b_1	b_2	b_3
0	0	0
1	1	1

C_1		
c_1	c_2	c_3
0	0	0
1	1	1

D_1		
d_1	d_2	d_3
0	0	0
1	1	1



	$\mathcal{M}$			
#	M_1	M_2	M_3	M_4
1	0	0	1	1
2	0	1	0	0
3	1	0	0	1
4	1	1	1	0
5	2	0	0	0
6	2	1	1	1

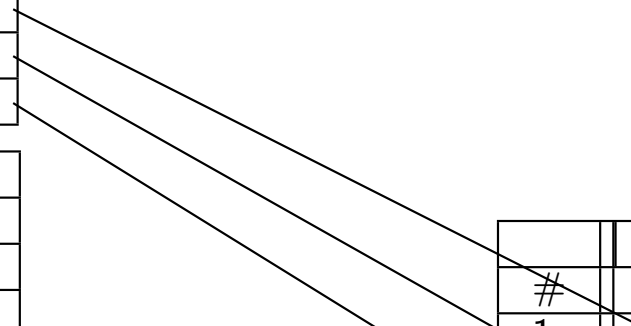
Plug-In of Seeds into Meta Array

A_1		
a_1	a_2	a_3
0	0	0
1	0	1
0	1	1

B_1		
b_1	b_2	b_3
0	0	0
1	1	1

C_1		
c_1	c_2	c_3
0	0	0
1	1	1

D_1		
d_1	d_2	d_3
0	0	0
1	1	1



	$\mathcal{M}$			
#	M_1	M_2	M_3	M_4
1	0	0	1	1
2	0	1	0	0
3	1	0	0	1
4	1	1	1	0
5	2	0	0	0
6	2	1	1	1

Plug-In of Seeds into Meta Array

A_1		
a_1	a_2	a_3
0	0	0
1	0	1
0	1	1

B_1		
b_1	b_2	b_3
0	0	0
1	1	1

C_1		
c_1	c_2	c_3
0	0	0
1	1	1

D_1		
d_1	d_2	d_3
0	0	0
1	1	1

	$\mathcal{M}$			
#	M_1	M_2	M_3	M_4
1	0	0	1	1
2	0	1	0	0
3	1	0	0	1
4	1	1	1	0
5	2	0	0	0
6	2	1	1	1

Plug-In of Seeds into Meta Array

A_1		
a_1	a_2	a_3
0	0	0
1	0	1
0	1	1

B_1		
b_1	b_2	b_3
0	0	0
1	1	1

C_1		
c_1	c_2	c_3
0	0	0
1	1	1

D_1		
d_1	d_2	d_3
0	0	0
1	1	1

	$\mathcal{M}$			
#	M_1	M_2	M_3	M_4
1	0	0	1	1
2	0	1	0	0
3	1	0	0	1
4	1	1	1	0
5	2	0	0	0
6	2	1	1	1

Plug-In of Seeds into Meta Array

A_1		
a_1	a_2	a_3
0	0	0
1	0	1
0	1	1

B_1		
b_1	b_2	b_3
0	0	0
1	1	1

C_1		
c_1	c_2	c_3
0	0	0
1	1	1

D_1		
d_1	d_2	d_3
0	0	0
1	1	1

	$\mathcal{M}$			
#	M_1	M_2	M_3	M_4
1	0	0	1	1
2	0	1	0	0
3	1	0	0	1
4	1	1	1	0
5	2	0	0	0
6	2	1	1	1

Plug-In of Seeds into Meta Array

A_1		
a_1	a_2	a_3
0	0	0
1	0	1
0	1	1

B_1		
b_1	b_2	b_3
0	0	0
1	1	1

C_1		
c_1	c_2	c_3
0	0	0
1	1	1

D_1		
d_1	d_2	d_3
0	0	0
1	1	1

	$\mathcal{M}$			
#	M_1	M_2	M_3	M_4
1	0	0	1	1
2	0	1	0	0
3	1	0	0	1
4	1	1	1	0
5	2	0	0	0
6	2	1	1	1

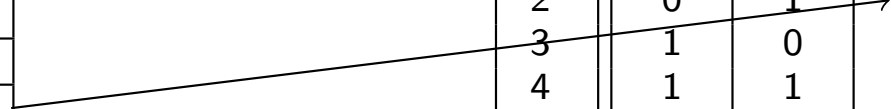
Plug-In of Seeds into Meta Array

A_1		
a_1	a_2	a_3
0	0	0
1	0	1
0	1	1

B_1		
b_1	b_2	b_3
0	0	0
1	1	1

C_1		
c_1	c_2	c_3
0	0	0
1	1	1

D_1		
d_1	d_2	d_3
0	0	0
1	1	1



	$\mathcal{M}$			
#	M_1	M_2	M_3	M_4
1	0	0	1	1
2	0	1	0	0
3	1	0	0	1
4	1	1	1	0
5	2	0	0	0
6	2	1	1	1

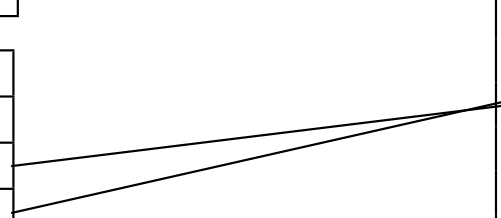
Plug-In of Seeds into Meta Array

A_1		
a_1	a_2	a_3
0	0	0
1	0	1
0	1	1

B_1		
b_1	b_2	b_3
0	0	0
1	1	1

C_1		
c_1	c_2	c_3
0	0	0
1	1	1

D_1		
d_1	d_2	d_3
0	0	0
1	1	1



	$\mathcal{M}$			
#	M_1	M_2	M_3	M_4
1	0	0	1	1
2	0	1	0	0
3	1	0	0	1
4	1	1	1	0
5	2	0	0	0
6	2	1	1	1

Plug-In of Seeds into Meta Array

A_1		
a_1	a_2	a_3
0	0	0
1	0	1
0	1	1

B_1		
b_1	b_2	b_3
0	0	0
1	1	1

C_1		
c_1	c_2	c_3
0	0	0
1	1	1

D_1		
d_1	d_2	d_3
0	0	0
1	1	1

	$\mathcal{M}$			
#	M_1	M_2	M_3	M_4
1	0	0	1	1
2	0	1	0	0
3	1	0	0	1
4	1	1	1	0
5	2	0	0	0
6	2	1	1	1

Plug-In of Seeds into Meta Array

A_1		
a_1	a_2	a_3
0	0	0
1	0	1
0	1	1

B_1		
b_1	b_2	b_3
0	0	0
1	1	1

C_1		
c_1	c_2	c_3
0	0	0
1	1	1

D_1		
d_1	d_2	d_3
0	0	0
1	1	1

	$\mathcal{M}$			
#	M_1	M_2	M_3	M_4
1	0	0	1	1
2	0	1	0	0
3	1	0	0	1
4	1	1	1	0
5	2	0	0	0
6	2	1	1	1

Plug-In of Seeds into Meta Array

A_1		
a_1	a_2	a_3
0	0	0
1	0	1
0	1	1

B_1		
b_1	b_2	b_3
0	0	0
1	1	1

C_1		
c_1	c_2	c_3
0	0	0
1	1	1

D_1		
d_1	d_2	d_3
0	0	0
1	1	1

	$\mathcal{M}$			
#	M_1	M_2	M_3	M_4
1	0	0	1	1
2	0	1	0	0
3	1	0	0	1
4	1	1	1	0
5	2	0	0	0
6	2	1	1	1

Result of Plug In

	$\mathcal{H}$											
#	a_1	a_2	a_3	b_1	b_2	b_3	c_1	c_2	c_3	d_1	d_2	d_3
1	0	0	0	0	0	0	1	1	1	1	1	1
2	0	0	0	1	1	1	0	0	0	0	0	0
3	1	0	1	0	0	0	0	0	0	1	1	1
4	1	0	1	1	1	1	1	1	1	0	0	0
5	0	1	1	0	0	0	0	0	0	0	0	0
6	0	1	1	1	1	1	1	1	1	1	1	1

Extending Bounds of Usability of CT

Observation

Analysis of CA generation tools shows that (most) of them rely on tuple counting

- May cause interruption of computation of large CAs

Possible Solution

- Use *'smaller'* CAs to build *'bigger'* CAs leveraging coverage inheritance
- In case an MCA is not directly constructible due to resource consumption and (constrained) computing environment:
  - employ a "Nested CA construction"

SBA Research

Nested CA Construction

Construct an $MCA(N; t, g, (v_1, \ldots, v_g))$

1. Partition parameter configuration into classes $V_1, \ldots, V_k$
2. For each $i = 1, \ldots, k$ construct seed arrays $S_i = MCA(N_i; t, |V_i|, V_i)$
3. Construct a meta array $\mathcal{M} = MCA(N_m; t, k, (N_1, \ldots, N_k))$
4. Apply a plug-in construction to the **seed arrays** $(S_i)_{i=1}^k$ and the **meta array** $\mathcal{M}$

Example: Construct $CA(N; 3, 500, 2)$

1. Partition parameters into classes of 10 parameters each.
2. Compute a seed array $S = CA(12; 3, 5, 2)$.
3. Compute a meta array $\mathcal{M} = CA(10807; 3, 100, 12)$.
4. Apply a plug-in construction to the seed array S and the meta array $\mathcal{M}$.

The resulting array is a $CA(10807; 3, 500, 2)$

Construction of Large CAs

Remarks

- The constructed arrays are not optimal
- When available infrastructure fails to compute a CA
- We might still be able to construct a CA
  - Computing only the seeds
  - Applying a plug-in construction

Array	Computation Time	Size	Memory Available
$CA(N; 3, 1000, 2)$	o.o.M.	-	5 GB
$S = CA(6; 2, 6, 2)$	0sec	6	5 GB
$T = CA(20; 3, 5, 2)$	0 sec	20	5 GB
$\mathcal{M} = CA(1635; 3, 200, 6)$	$\sim$10 min	1635	5 GB
$\mathcal{R} = CA(1655; 3, 1000, 6)$	0 sec	1655	5 GB
$CA(N; 3, 2000, 3)$	o.o.M. (after $>$7 hrs)	-	10 GB
$S = CA(6; 2, 5, 2)$	0 sec	6	10 GB
$T = CA(20; 3, 5, 2)$	0 sec	20	10 GB
$\mathcal{M} = CA(1930; 3, 400, 6)$	$\sim$2.5 hrs	1930	10 GB
$\mathcal{R} = CA(1950; 3, 2000, 2)$	0 sec	1950	10 GB

"o.o.M.", the computation aborted with an out of memory error.

Summary

Highlights

1. Combinatorial methods for modelling composed software systems
   - automotives
   - Linux kernel system calls
2. Combinatorial constructions for building nested test suites (resp. CAs) from nested IPMs (resp. CAs):
   - use of t-way coverage inheritance
3. Extending bounds of usability of CT

Future Work

- Cyber-Physical Systems (CPS) Modelling
- Internet of Things (IoT)

SBA Research

References

J. Czerwonka, "Pairwise testing in the real world: Practical extensions to test-case scenarios," in Proceedings of 24th Pacific Northwest Software Quality Conference, Citeseer, 2006, pp. 419-430.

D. M. Cohen, S. R. Dalal, M. L. Fredman, and G. C. Patton, "The AETG system: An approach to testing based on combinatorial design," IEEE Transactions on Software Engineering, vol. 23, no. 7, pp. 437-444, 1997.

B. Garn and D. E. Simos, "Eris: A tool for combinatorial testing of the linux system call interface," in Software Testing, Verification and Validation Workshops (ICSTW), 2014 IEEE Seventh International Conference on. IEEE, 2014, pp. 58-67.

R. Krishnan, S. M. Krishna, and P. S. Nandhan, "Combinatorial testing: learnings from our experience," ACM SIGSOFT Software Engineering Notes, vol. 32, no. 3, pp. 1-8, 2007.

SBA Research

Questions - Comments

Thank you for your Attention!

dsimos@sba-research.org