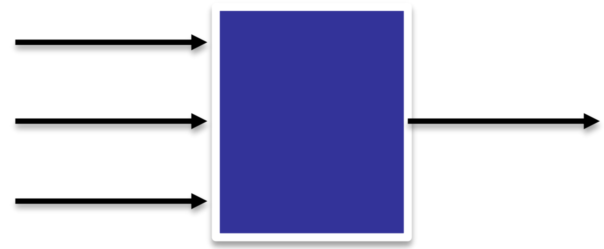
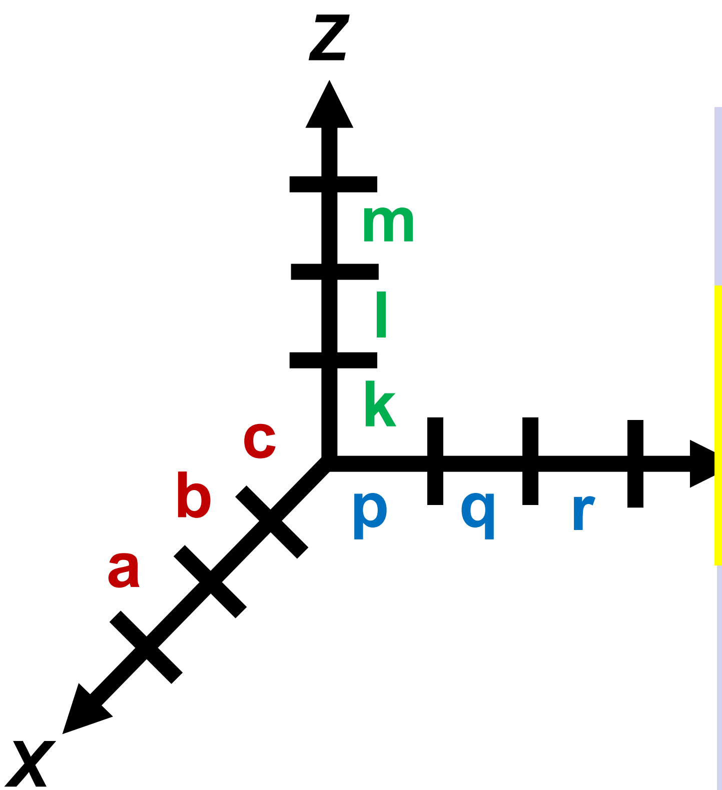
Combinatorial testing

Multiple Variables : Combinatorial Testing

X : { a,b,c }
Y : { k,l,m }
Z : { p,q,r }





Equivalence Partitioning = 1-wise:

3 test cases. e.g.: (a,k,p)

In between:

2-wise (pairwise)

9 test cases

All combinations

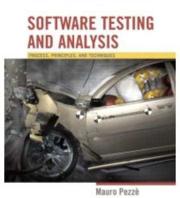
= Cartesian product = 3-wise:

3 x 3 x 3 = 27 test cases

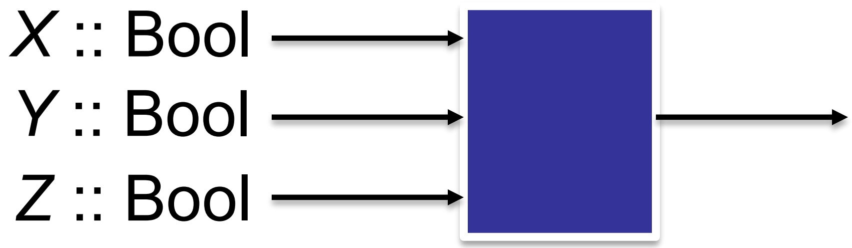
Pairwise combinatorial testing

- Pairwise combination (instead of exhaustive)
  - Generate combinations that efficiently cover all pairs (triples,…) of classes
  - Rationale: most failures are triggered by single values or combinations of a few values. Covering pairs (triples,…) reduces the number of test cases, but reveals most faults

- Generalized: *t*- wise testing
  - 2- wise = pairwise



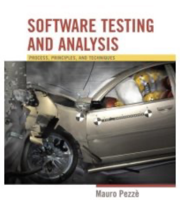
Combinatorial Testing



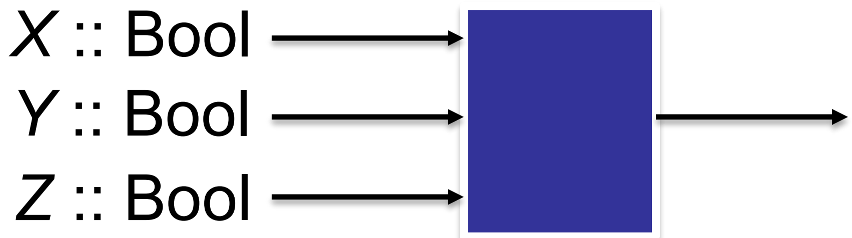
- ***1-wise testing*:**

  each variable X,Y,Z has each possible value at least *once*

	X	Y	Z
test case 1	0	0	0
test case 2	1	1	1



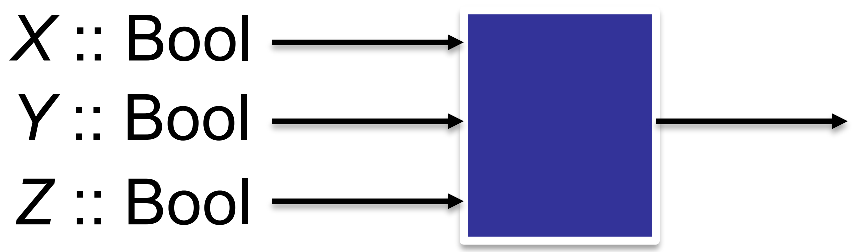
Combinatorial Testing



- ***2-wise testing = pairwise:***

  each pair of variables *(X,Y)*, (X,*Z)*, *(Y,Z)*
  has each possible pair of values at least *once*

Combinatorial Testing



- ***3-wise testing:***

  each triple of possible values occurs at least once

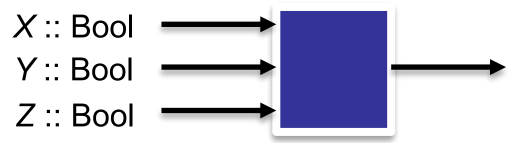
	X	*Y*	*Z*
tc 1	0	0	0
tc 2	0	0	1
tc 3	0	1	0
tc 4	0	1	1
tc 5	1	0	0
tc 6	1	0	1
tc 7	1	1	0
tc 8	1	1	1

Combinatorial Testing

2-wise testing
= pairwise:

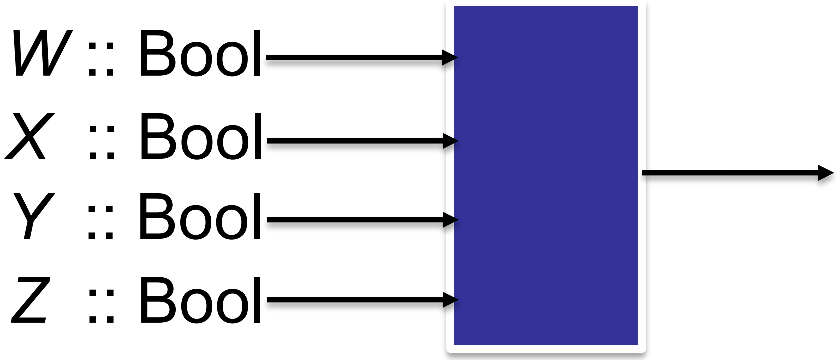
each pair of variables (*X*,*Y*), (X,*Z*), (*Y*,*Z*) *has* each possible pair of values at least *once*

X	*Y*	*Z*
0	0	
0	1	
1	0	
1	1	
0		0
0		1
1		0
1		1
	0	0
	0	1
	1	0
	1	1



	X	*Y*	*Z*
tc 1	0	0	0
tc 2	0	0	1
tc 3	0	1	0
tc 4	0	1	1
tc 5	1	0	0
tc 6	1	0	1
tc 7	1	1	0
tc 8	1	1	1

Combinatorial Testing



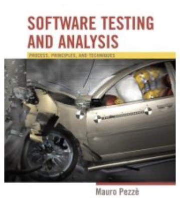
- ***2-wise testing = pairwise:***
  each pair of variables
  (W,X), (W,Y), W,Z), (X,Y), (X,Z), (Y,Z)
  has each possible pair of values at least *once*

[Tools: e.g., http://www.pairwise.org/](http://www.pairwise.org/)

Example: Display Control

No constraints reduce the total number of combinations
432 (3x4x3x4x3) test cases
if we consider all combinations

Display Mode	Language	Fonts	Color	Screen size
full-graphics	English	Minimal	Monochrome	Hand-held
text-only	French	Standard	Color-map	Laptop
limited-bandwidth	Spanish	Document-loaded	16-bit	Full-size
	Portuguese		True-color	



Pairwise combinations: 17 test cases

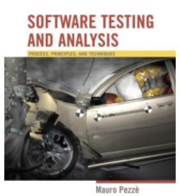
Language	Color	Display Mode	Fonts	Screen Size
English	Monochrome	Full-graphics	Minimal	Hand-held
English	Color-map	Text-only	Standard	Full-size
English	16-bit	Limited-bandwidth	-	Full-size
English	True-color	Text-only	Document-loaded	Laptop
French	Monochrome	Limited-bandwidth	Standard	Laptop
French	Color-map	Full-graphics	Document-loaded	Full-size
French	16-bit	Text-only	Minimal	-
French	True-color	-	-	Hand-held
Spanish	Monochrome	-	Document-loaded	Full-size
Spanish	Color-map	Limited-bandwidth	Minimal	Hand-held
Spanish	16-bit	Full-graphics	Standard	Laptop
Spanish	True-color	Text-only	-	Hand-held
Portuguese	-	-	Monochrome	Text-only
Portuguese	Color-map	-	Minimal	Laptop
Portuguese	16-bit	Limited-bandwidth	Document-loaded	Hand-held
Portuguese	True-color	Full-graphics	Minimal	Full-size
Portuguese	True-color	Limited-bandwidth	Standard	Hand-held

Adding constraints

- Simple constraints

  example: color monochrome not compatible with screen laptop and full size

  can be handled by considering the case in separate tables

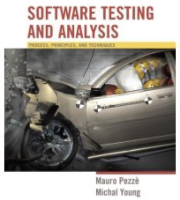


Example: Monochrome only with hand-held

Display Mode	Language	Fonts	Color	Screen size
full-graphics	English	Minimal	Monochrome	Hand-held
text-only	French	Standard	Color-map	
limited-bandwidth	Spanish	Document-loaded	16-bit	
	Portuguese		True-color	

Display Mode	Language	Fonts	Color	Screen size
full-graphics	English	Minimal		
text-only	French	Standard	Color-map	Laptop
limited-bandwidth	Spanish	Document-loaded	16-bit	Full-size
	Portuguese		True-color	

MC/DC Coverage

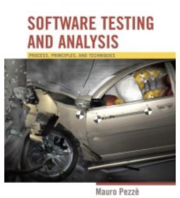


Compound conditions: Exponential complexity

`(((a || b) && c) || d) && e`

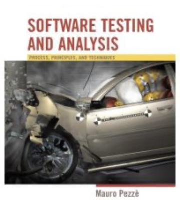
Test Case	a	b	c	d	e
(1)	T	—	T	—	T
(2)	F	T	T	—	T
(3)	T	—	F	T	T
(4)	F	T	F	T	T
(5)	F	F	—	T	T
(6)	T	—	T	—	F
(7)	F	T	T	—	F
(8)	T	—	F	T	F
(9)	F	T	F	T	F
(10)	F	F	—	T	F
(11)	T	—	F	F	—
(12)	F	T	F	F	—
(13)	F	F	—	F	—

- short-circuit evaluation often reduces this to a more manageable number, but not always



Modified condition/decision (MC/DC)

- Motivation: Effectively test important combinations of conditions, without exponential blowup in test suite size
  - "Important" combinations means: Each basic condition shown to independently affect the outcome of each decision
- Requires:
  - For each basic condition C, two test cases,
  - values of all *evaluated* conditions except C are the same
  - compound condition as a whole evaluates to *true* for one and *false* for the other



MC/DC Coverage

$$(x \wedge y) \vee z$$

X	Y	Z	$(X \wedge Y) \vee Z$	
0	0	0	0	
0	0	1	1	
0	1	0	0	X
0	1	1	1	
1	0	0	0	Y Z
1	0	1	1	Z
1	1	0	1	X Y
1	1	1	1	

MC/DC: linear complexity

- N+1 test cases for N basic conditions

```
(((a || b) && c) || d) && e
```

Test Case	a	b	c	d	e	outcome
(1)	true	false	true	false	true	*true*
(2)	false	true	true	false	true	*true*
(3)	true	false	false	true	true	*true*
(6)	true	false	true	false	false	*false*
(11)	true	false	false	false	true	*false*
(13)	false	false	true	false	true	*false*

- Red values independently affect the output of the decision
- Green values are don’t care for the output
- Required by the RTCA/DO-178B standard