Functional Testing

Mohammad Mousavi

Department of Informatics, King's College London

Software Testing and Measurement

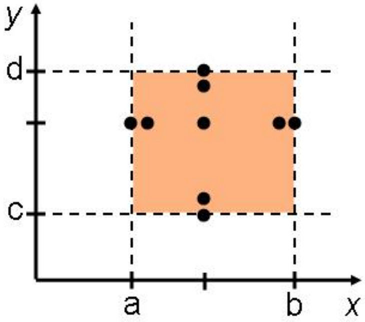
Outline

Functional Testing

- functional testing:
  program is an input from a certain domain to a certain range
- impossible to check all input/output combinations:
  defining a coverage criterion to choose some some

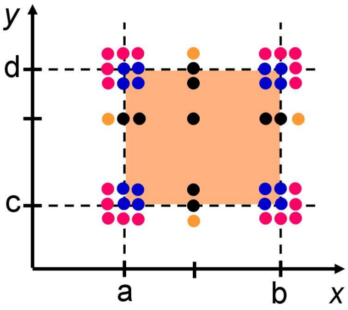
Boundary Value Testing

- boundary value testing: a test case for each combination of extreme (normal, out of bound) values



Boundary Value Testing: Pros and Cons

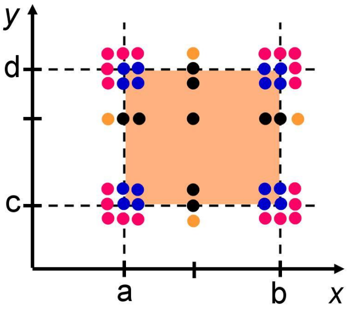
- + straightforward test-case generation
- - no sense of covering the input domain
- - awkward for logical vars.
- - only independent input domains
- - not using white-box information



Boundary Value Testing: Pros and Cons

+ straightforward test-case generation
- no sense of covering the input domain *
- awkward for logical vars. *
- only independent input domains *
- not using white-box information

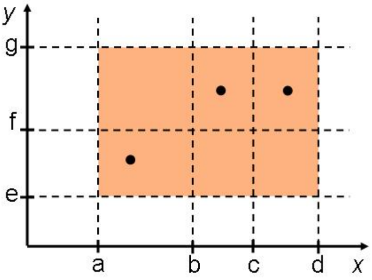
*: See: decision tables and classification trees.



Outline

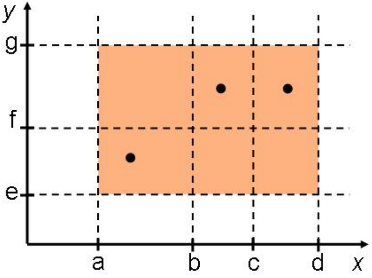
Weak Normal EC: Idea

- Define equivalence classes on the domain (range) of input (output) for each variable: (independent input)
- cover equivalence classes for the domain of each variable: single fault assumption
- how many test-cases are needed?
- also called: (equivalence, category) partition method



Little Puzzle

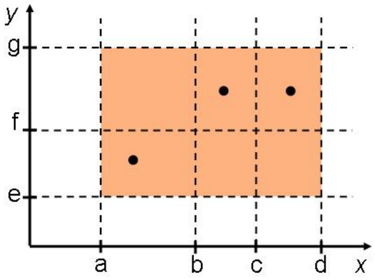
What is the minimal number of tokens that are needed to be put in an $m \times n$ grid such that each row and column contains at least one token?



Little Puzzle

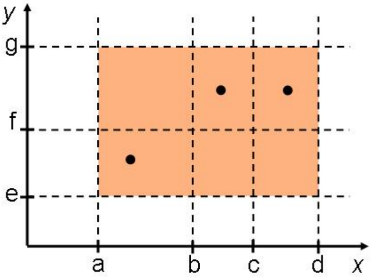
What is the minimal number of tokens that are needed to be put in an $m \times n$ grid such that each row and column contains at least one token?

max(m,n):
Put token number i at $(min(i, m), min(i, n))$.



Weak Normal EC: Idea

- Define equivalence classes on the domain (range) of input (output) for each variable:
  (independent input)
- cover equivalence classes for the domain of each variable:
  single fault assumption
- how many test-cases are needed?
  $\max_x \mid S_x \mid$.



Mortgage Example

Spec. Write a program that takes
three inputs: full-time (boolean), age([18-55]), salary ([0-10000])
and output the total mortgage for one person

Mortgage = salary * factor,
where factor is given by the following table.

Category	**full-time = true**	**false**
Young	(18-35 years) 75	(18-30 years) 70
Middle	(36-45 years) 55	(31-40 years) 50
Old	(46-55 years) 30	(41-50 years) 35

Weak Normal EC Testing

Category	full-time = true	false
Young	(18-35 years) 75	(18-30 years) 70
Middle	(36-45 years) 55	(31-40 years) 50
Old	(46-55 years) 30	(41-50 years) 35

- age: difficult!
- salary: [0-10000]
- full-time: as strange as boundary value!

Weak Normal EC Testing

Category	full-time = true	false
Young	(18-35 years) 75	(18-30 years) 70
Middle	(36-45 years) 55	(31-40 years) 50
Old	(46-55 years) 30	(41-50 years) 35

- age: difficult! [18-30], [31-35], [36-40], [41,45], [46-50], [51-55]
- salary: [0-10000]
- full-time: as strange as boundary value! true, false

Weak Normal EC Testing

if (full-time) **then return**

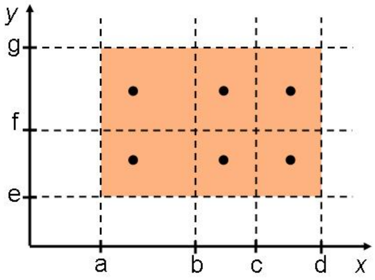
$((18 \leq age < 35)?(75 * salary) : (31 \leq age < 40)?(55 * salary) : (30 * salary))$

else return $((18 \leq age < 30)?(75 * salary) : (31 \leq age < 40)?(50 * salary) : (35 * salary))$

Full-time	Age	Salary	Output	Correct Out.	Pass/Fail
true	20	1000	75*1000	75*1000	P
false	32	1000	50*1000	50*1000	P
true	38	1000	55*1000	50*1000	P
false	42	1000	35*1000	35*1000	P
true	48	1000	30*1000	30*1000	P
false	52	1000	35*5000	too late!	F

Strong Normal EC Testing

- cover the all combinations of equivalence classes for the domain of all variables:
  multiple fault assumption
- number of test-cases? $\prod_x \mid S_x \mid$, where $\prod$ stands for multiplication



Strong Normal EC Testing

Category	**true**	**false**
Young	(18-35 years) 75	(18-30 years) 70
Middle	(36-45 years) 55	(31-40 years) 50
Old	(46-55 years) 30	(41-50 years) 35

- age: [18-30], [31-35], [36-40], [41,45], [46-50], [51-55]
- salary: [0-10000]
- full-time: true, false

Strong Normal EC Testing

if (full-time) **then return**

$((18 \leq age < 35)?(75 * salary) : (31 \leq age < 40)?(55 * salary) : (30 * salary))$

else return $((18 \leq age < 30)?(75 * salary) : (31 \leq age < 40)?(50 * salary) : (35 * salary))$

Full time	Age	Salary	Output	Correct Out.	Pass/Fail
false	20	1000	75*1000	70*1000	F
false	32	1000	50*1000	50*1000	P
false	38	1000	50*1000	50*1000	P
false	42	1000	35*1000	35*1000	P
false	48	1000	35*1000	35*1000	P
false	52	1000	35*5000	too late!	F

Strong Normal EC Testing

if (full-time) **then return**

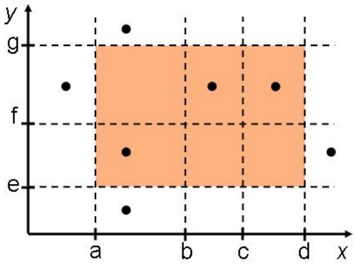
$((18 \leq age < 35)?(75 * salary) : (31 \leq age < 40)?(55 * salary) : (30 * salary))$

else return $((18 \leq age < 30)?(75 * salary) : (31 \leq age < 40)?(50 * salary) : (35 * salary))$

Full time	Age	Salary	Output	Correct Out.	Pass/Fail
true	20	1000	75*1000	75*1000	P
true	32	1000	75*1000	75*1000	P
true	38	1000	55*1000	50*1000	P
true	42	1000	30*1000	55*1000	F
true	48	1000	30*1000	30*1000	P
true	52	1000	30*1000	30*1000	P

Weak Robust EC

- includes weak normal; adds out of range test-cases for each variable
- number of test-cases?
  $(\max_x \mid S_x \mid) + 2 * n$



Weak Robust EC Testing

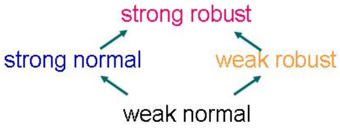
if (full-time) **then return**

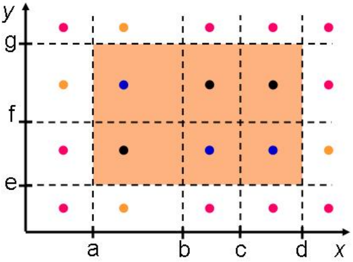
$((18 \leq age < 35)?(75 * salary) : (31 \leq age < 40)?(55 * salary) : (30 * salary))$

else return $((18 \leq age < 30)?(75 * salary) : (31 \leq age < 40)?(50 * salary) : (35 * salary))$

Full time	Age	Salary	Output	Correct Out.	Pass/Fail
true	17	1000	30*1000	too young!	F
false	56	1000	35*1000	too late	F
true	36	-1	55*-1	0	F
false	36	10001	50*10001	50*10000	F

A Brief Comparison

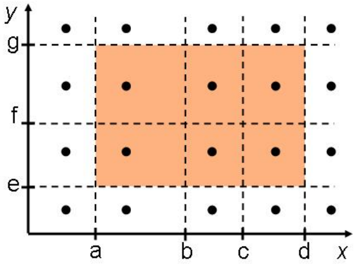




$A \rightarrow B$: Test-cases of A (faults detected by A) is a subset of those of B.

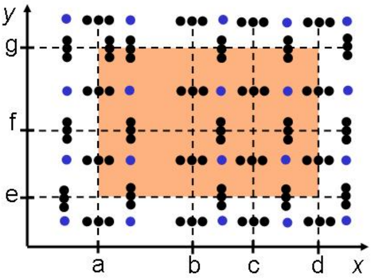
Strong Robust EC

- Same as strong normal but also checks for all out of range combinations
- number of test-cases? $\prod_x(\mid S_x \mid +2)$



Worst-Case: BV + EC

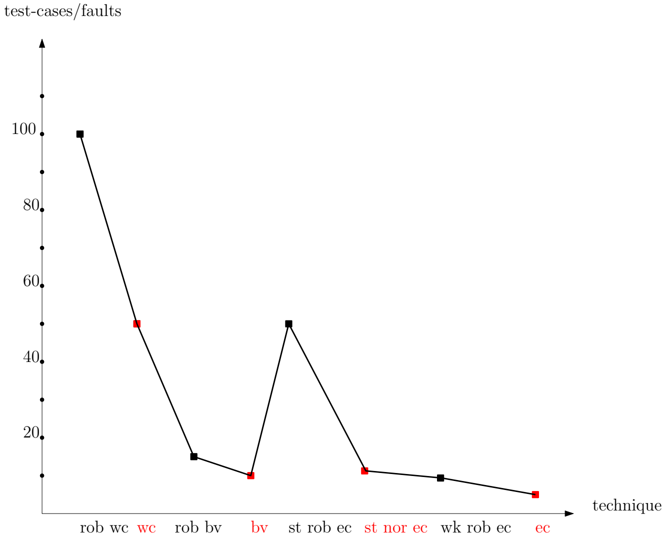
- Considering the boundaries of each partition relevant
- Example:
  Robust worst case testing of of partitions



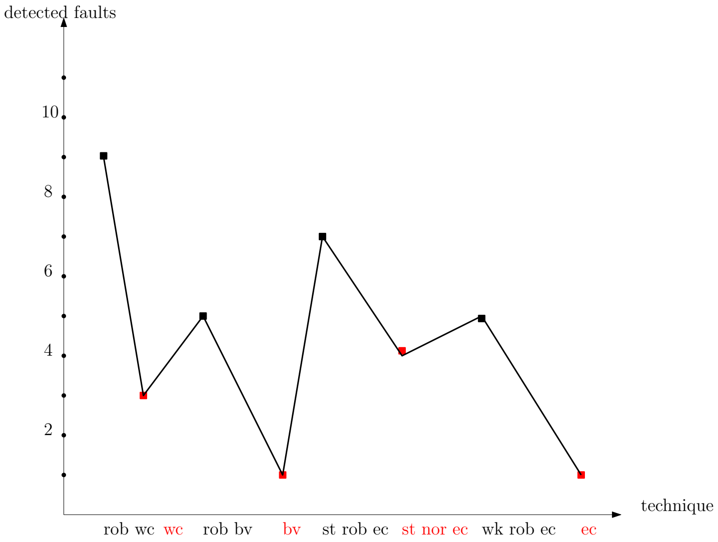
Strong Robust EC + Robust BV

Full-time	Age	Salary	Output	Correct Out.	Pass/Fail
true	17	-1	30*-1	too young!	F 1
true	17	1000	30*1000	too young!	F 1
true	17	10001	30*10001	too young!	F 1
true	56 ‘	-1	30*-1	too late	F 2
true	56	1000	30*1000	too late	F 2
true	56	10001	30*10001	too late	F 2
false	17	-1	30*-1	too young!	F 3
false	17	1000	30*1000	too young!	F 3
false	17	10001	30*10001	too young!	F 3
false	56	-1	30*-1	too late	F 4
false	56	1000	30*1000	too late	F 4
false	56	10001	30*10001	too late	F 4

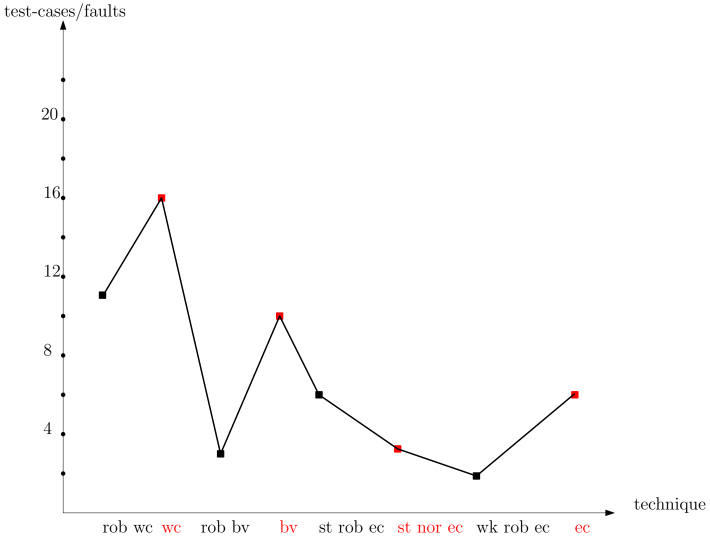
Mortgage Case: #Test-Cases



Mortgage Case: Detected Fault

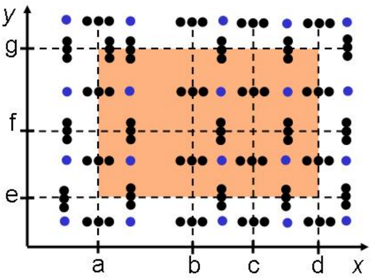


Mortgage Case: #Test-Cases/Fault



Problems

- Problems:
  1. No constraints on the equivalence classes
  2. Dependencies among different variables not taken into account
  3. No choice among relevant classes (e.g., apply worst-case testing on some and boundary values on others)
- Solutions: Attend the coming lecture!



Outline

Idea

- Goal: Summarize the logic of the program (à la Karnaugh maps)
- Find a few conditions on input determining the output behavior
  need not be independent
  relaxing the independence assumption in all previous techniques
- Determine the output actions
  for each combination of condition evaluations
- also called: cause-effect graph testing, or
  tableau testing

Basic Concepts

- Stub:
  - condition part
    the most dominating conditions first
    multi-valued conditions and special cases last
  - action part
    exceptions
    preferably combined actions as new rows

Stub	Entry		
c1	F	T	T
c2	-	F	T
c3	-	-	F
a1	X	-	-
a2	-	X	-
a1;a2	-	-	X

Basic Concepts

- Entry
  - columns are called rules
  - condition part: true, false, (possibly other values) or don't care
  - action part

Stub	Entry		
c1	F	T	T
c2	-	F	T
c3	-	-	F
a1	X	-	-
a2	-	X	-
a1;a2	-	-	X

Basic Concepts

- Completeness check for independent variables
  - each don't care counts for two rules
  - there must be $2^{|\{c_i\}|}$ rules (for n_i-valued conditions: $\prod_i n_i$)

c1	F	T	T
c2	-	F	T
c3	-	-	F
a1	X	-	-
a2	-	X	-
a1;a2	-	-	X

Basic Concepts

- Completeness check for independent variables
  - each don't care counts for two rules
  - there must be $2^{|\{c_i\}|}$ rules (for n_i-valued conditions: $\prod_i n_i$)

c1	F	T	T	T
c2	-	F	T	T
c3	-	-	F	T
a1	X	-	-	-
a2	-	X	-	-
a1;a2	-	-	X	-
error	-	-	-	X

Conditions/Actions									
c7: 0≤salary≤10000?	F	T	T	T	T	T	T	T	T
c1: full-time?	-	-	-	T	T	T	F	F	F
c2: too young? [..,18]	-	T	F	F	F	F	F	F	F
c3: young? ft:[18,..,35], pt:[18,..,30]	-	F	F	T	F	F	T	F	F
c4: mid? ft:[36,..,45], pt:[31,..,40]	-	F	F	F	T	F	F	T	F
c5: old? ft:[46,..,55], pt:[40,..,50]	-	F	F	F	F	T	F	F	T
c6: too old? ft:[56,..], pt:[51,..]	-	F	T	F	F	F	F	F	F
a1: wrong inputs	X	X	X	-	-	-	-	-	-
a2: 75*salary	-	-	-	X	-	-	-	-	-
a3: 70*salary	-	-	-	-	-	-	X	-	-
a4: 55*salary	-	-	-	-	X	-	-	-	-
a5: 50*salary	-	-	-	-	-	-	-	X	-
a6: 35*salary	-	-	-	-	-	-	-	-	X
a7: 30*salary	-	-	-	-	-	X	-	-	-

Decision Table for Testing

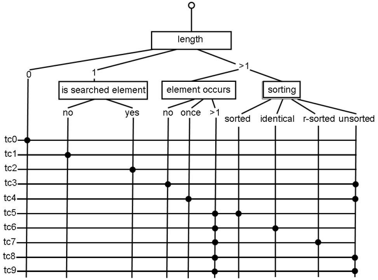
variables: Physical or Logical	P	P	P	P	P	L	L	L	L	L
Independent?	T	T	T	T	F	T	T	T	T	F
Single fault assum.?	T	T	F	F	-	T	T	F	F	-
Exception handling?	T	F	T	F	-	T	F	T	F	-
BV		X								
Robust	X									
WC				X						
Robust WC			X							
EC							X			
Strong (Normal) EC									X	
(Weak) Robust EC						X				
Strong Robust EC								X		
Decision Table					X					X

Outline

Basic Steps

Classification tree:

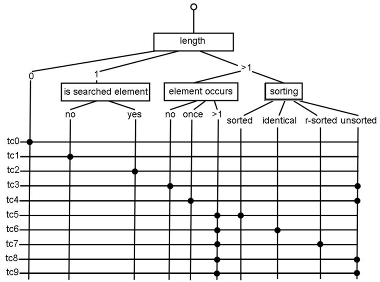
- Determine the aspects of specification influencing the logic
- Establish a hierarchy between aspects (the more global conditions first)
- Partition the input domain for each aspect
  cover the whole domain of the "parent" node



Basic Steps

Combination table:

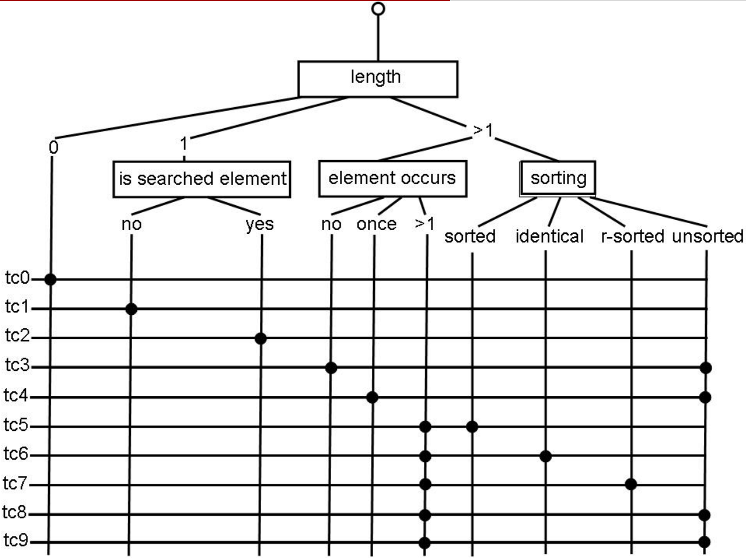
- Define a test-case for each relevant combination of inputs



Example

Informal Spec

Consider the function $search(list : List(El), el : El) : int$,
which takes an arbitrary list of elements (empty, ordered, unordered, or reversely-ordered),
and an element
and output the indices of the occurrences of the element in the list (if there are no occurrences or the list is empty, -1 should be returned; if there are repeated occurrences, all their indices should be returned).


length
0
1
>1
is searched element
no
yes
element occurs
no
once
>1
sorting
sorted
identical
r-sorted
unsorted
tc0
tc1
tc2
tc3
tc4
tc5
tc6
tc7
tc8
tc9

Mortgage Example

Classes

1. Salary: -1, [0..10000], >10000,
2. Full time: true, false,
3. Age: Too young, Young, Middle, Old, Too old (dependent on Full time)

Example

Informal Spec

Consider a computer vision system that takes different shapes as its input and classifies them based on their size (in two categories: large or small) and their colour (red, green, or blue). Large shapes are further classified based on their shape: circle, square, or triangle. Triangles are further classified into equilateral, isosceles, scalene.

Outline

Functional Testing

- Equivalence testing forms the basis:
  - Strong variants are often practically infeasible
  - Robust techniques are very effective for PL's with weak typing
- Decision tables and classification trees, help us in:
  1. summarizing the logic
  2. identifying and documenting the effective methods and test-cases.

One Sentence to Take Home

No perfect functional testing technique exists:
consider your domain and how much coverage of requirements is justified;
often classification tree (or decision-table)
provide a structured overview of the requirements.

How to Be More Selective?

Read the extra-curricular paper on combinatorial testing....