

Functional Testing

Mohammad Mousavi

Halmstad University, Sweden

`http://ceres.hh.se/mediawiki/DT8021_Ed_2015`

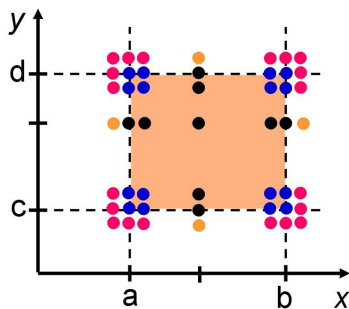
Testing and Verification of Embedded Systems (DT8021),
March 23, 2015

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**

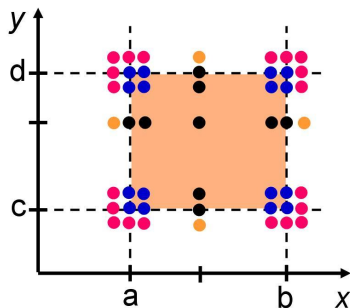
Boundary Value Testing

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



Boundary Value Testing: Pros and Cons

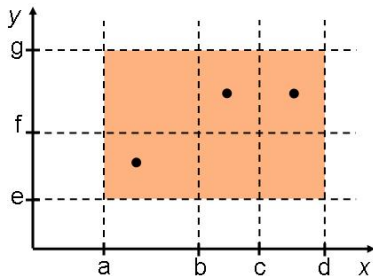
- + straightforward test-case generation
- no sense of covering the input domain *
- awkward for logical vars. *
- only independent input domains *
- not using white-box information



*: Today's order of business.

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

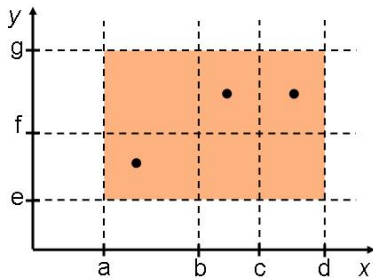


Little Puzzle

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**?

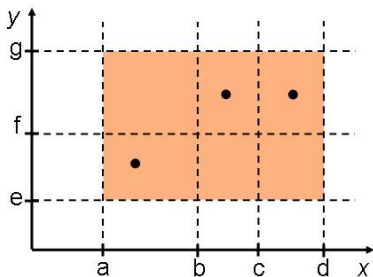
$\max(m,n)$:

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



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?
 $\max_x |S_x|$.



Mortgage Example (recap)

Spec. Write a program that takes three **inputs**: gender (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	Male	Female
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	Male	Female
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]
- ▶ **male:** as strange as boundary value! true, false

Weak Normal EC Testing

if (male) then return

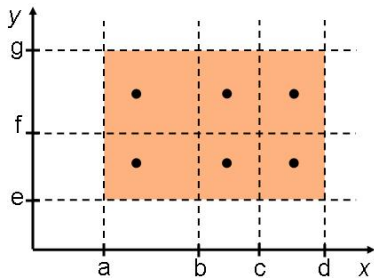
$((18 \leq \text{age} < 35)?(75 * \text{salary}) : (31 \leq \text{age} < 40)?(55 * \text{salary}) : (30 * \text{salary}))$

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

Gender	Age	Salary	Output	Correct Out.	Pass/Fail
male	20	1000	75*1000	75*1000	P
female	32	1000	50*1000	50*1000	P
male	38	1000	55*1000	50*1000	P
female	42	1000	35*1000	35*1000	P
male	48	1000	30*1000	30*1000	P
female	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? $\prod_x |S_x|$



Strong Normal EC Testing

Category	Male	Female
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]
- ▶ **male:** true, false

Strong Normal EC Testing

if (male) then return

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

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

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

Strong Normal EC Testing

if (male) then return

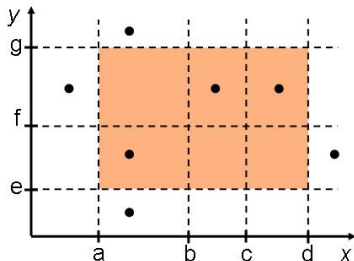
$((18 \leq \text{age} < 35)?(75 * \text{salary}) : (31 \leq \text{age} < 40)?(55 * \text{salary}) : (30 * \text{salary}))$

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

Gender	Age	Salary	Output	Correct Out.	Pass/Fail
male	20	1000	75*1000	75*1000	P
male	32	1000	50*1000	75*1000	F
male	38	1000	55*1000	50*1000	P
male	42	1000	30*1000	55*1000	F
male	48	1000	30*1000	30*1000	P
male	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?
 $(\max_x |S_x|) + 2 * n$



Weak Robust EC Testing

if (male) then return

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

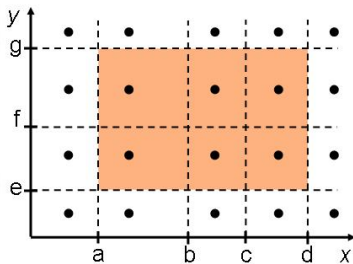
else return $((18 \leq \text{age} < 30)?(75 * \text{salary}) : (31 \leq \text{age} < 40)?(50 * \text{salary}) : (35 * \text{salary}))$

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

Strong Robust EC

- ▶ Same as strong normal but also checks for all out of range combinations
- ▶ number of test-cases?

$$\prod_x (|S_x| + 2)$$



Strong Robust EC

if (male) then return

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

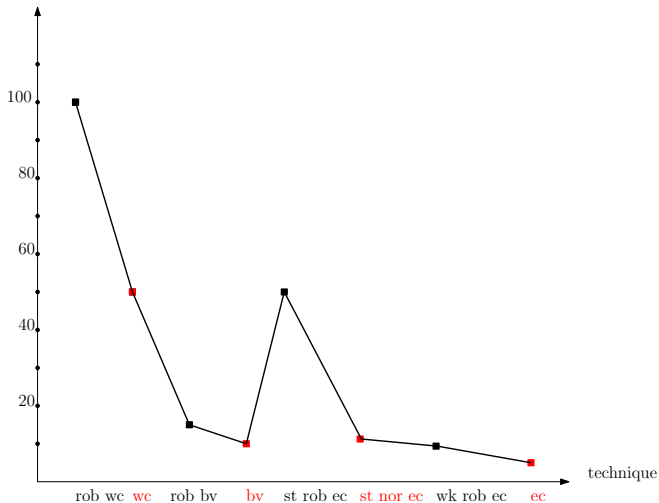
else return $((18 \leq \text{age} < 30)?(75 * \text{salary}) : (31 \leq \text{age} < 40)?(50 * \text{salary}) : (35 * \text{salary}))$

Mostly similar faults to Weak Robust EC:

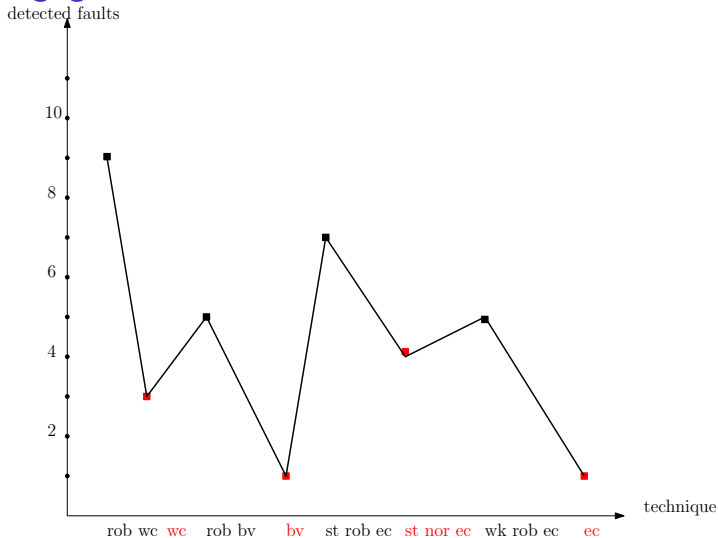
Gender	Age	Salary	Output	Correct Out.	Pass/Fail
male	17	1000	30*1000	too young!	F
female	56	1000	35*1000	too late	F
female	17	1000	35*1000	too young!	F
male	56	1000	30*1000	too late	F
male	36	-1	55*-1	0	F
female	36	10001	50*10001	50*10000	F
...					

Mortgage Case: #Test-Cases

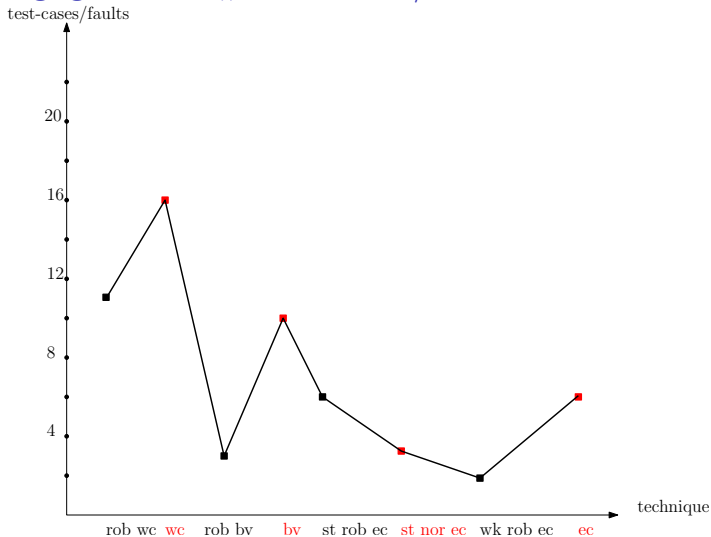
test-cases/faults



Mortgage Case: Detected Fault

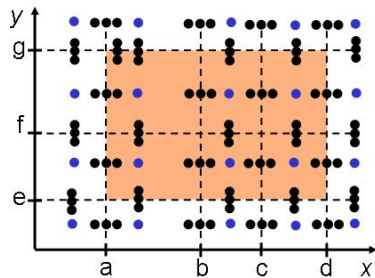


Mortgage Case: #Test-Cases/Fault



Idea

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



Strong Robust EC + Robust BV

Gender	Age	Salary	Output	Correct Out.	Pass/Fail
male	17	-1	30*-1	too young!	F 1
male	17	1000	30*1000	too young!	F 1
male	17	10001	30*10001	too young!	F 1
male	56	-1	30*-1	too late	F 2
male	56	1000	30*1000	too late	F 2
male	56	10001	30*10001	too late	F 2
female	17	-1	30*-1	too young!	F 3
female	17	1000	30*1000	too young!	F 3
female	17	10001	30*10001	too young!	F 3
female	56	-1	30*-1	too late	F 4
female	56	1000	30*1000	too late	F 4
female	56	10001	30*10001	too late	F 4

Strong Robust EC + Robust BV (Cont'd)

Gender	Age	Salary	Output	Correct Out.	Pass/Fail
female	18	1000	75*1000	70*1000	F 5
female	19	1000	75*1000	70*1000	F 5
female	20	1000	75*1000	70*1000	F 5
female	29	1000	75*1000	70*1000	F 5
female	30	1000	35*1000	70*1000	F 6
female	31	1000	50*1000	50*1000	P
female	32	1000	50*1000	50*1000	P
female	34	1000	50*1000	50*1000	P
female	35	1000	50*1000	50*1000	P
female	36	1000	50*1000	50*1000	P
female	38	1000	50*1000	50*1000	P
female	39	1000	50*1000	50*1000	P
female	40	1000	35*1000	50*1000	F 7

Strong Robust EC + Robust BV (Cont'd)

Gender	Age	Salary	Output	Correct Out.	Pass/Fail
female	41	1000	35*1000	35*1000	P
female	42	1000	35*1000	35*1000	P
female	44	1000	35*1000	35*1000	P
female	45	1000	35*1000	35*1000	P
female	46	1000	35*1000	35*1000	P
female	49	1000	35*1000	35*1000	P
female	50	1000	35*1000	35*1000	P
female	51	1000	35*1000	too late!	F 7
female	52	1000	35*1000	too late!	F 7
female	53	1000	35*1000	too late!	F 7
female	54	1000	35*1000	too late!	F 7
female	55	1000	35*1000	too late!	F 7

Strong Robust EC + Robust BV (Cont'd)

Gender	Age	Salary	Output	Correct Out.	Pass/Fail
male	18	1000	75*1000	75*1000	P
male	19	1000	75*1000	75*1000	P
male	20	1000	75*1000	75*1000	P
male	29	1000	75*1000	75*1000	P
male	30	1000	75*1000	75*1000	P
male	31	1000	55*1000	75*1000	F 8
male	32	1000	55*1000	75*1000	F 8
male	34	1000	55*1000	75*1000	F 8
male	35	1000	55*1000	75*1000	F 9
male	36	1000	55*1000	55*1000	P
male	38	1000	55*1000	55*1000	P
male	39	1000	55*1000	55*1000	P
male	40	1000	55*1000	20*1000	F 10

Strong Robust EC + Robust BV (Cont'd)

Gender	Age	Salary	Output	Correct Out.	Pass/Fail
male	41	1000	30*1000	30*1000	P
male	42	1000	30*1000	30*1000	P
male	44	1000	30*1000	30*1000	P
male	45	1000	30*1000	30*1000	P
male	46	1000	30*1000	30*1000	P
male	49	1000	30*1000	30*1000	P
male	50	1000	30*1000	30*1000	P
male	51	1000	30*1000	30*1000	P
male	52	1000	30*1000	30*1000	P
male	53	1000	30*1000	30*1000	P
male	54	1000	30*1000	30*1000	P
male	55	1000	30*1000	30*1000	P

Strong Robust EC + Robust BV (Cont'd)

Gender	Age	Salary	Output	Correct Out.	Pass/Fail
female	17	-1	35*-1	0	F 11
female	18	-1	75*-1	0	F 11
.....					

Strong Robust EC + Robust BV (Cont'd)

Gender	Age	Salary	Output	Correct Out.	Pass/Fail
female	17	10001	35*10001	too young!	F 11
female	18	10001	75*10001	75*10000	F 12
...					

Strong Robust EC + Robust BV (Cont'd)

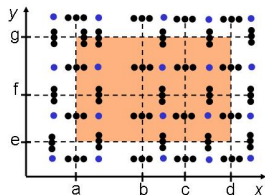
Gender	Age	Salary	Output	Correct Out.	Pass/Fail
male	17	-1	30*-1	0	F 12
male	18	-1	70*-1	0	F 12
...					

Strong Robust EC + Robust BV (Cont'd)

Gender	Age	Salary	Output	Correct Out.	Pass/Fail
male	17	10001	30*10001	too young!	F 12
male	18	10001	70*10001	75*10000	F 12
...					

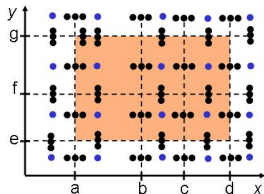
Problems

- ▶ Example:
Strong EC + Robust BV
number of test-cases:
 $\sim \prod_x 4(|S_x| + 1)$, whopping!



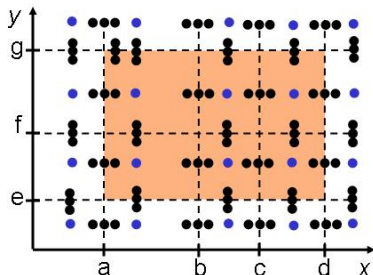
Problems

- ▶ **>100** test-cases for the **mortgage** example
- ▶ **too many** for any **real-life** program
e.g., 5 vars., each 5 partitions:
~ **8 million** test-cases
1 sec. for each test-case:
3 months testing!



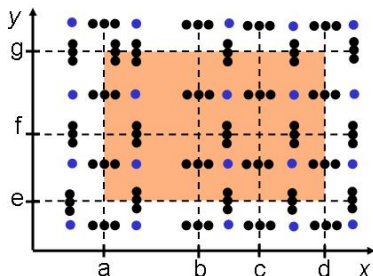
Problems

- ▶ 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!



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!



Possible Solution: Pairwise Testing

- ▶ Pairwise testing: for each two variables and each two partitions of their valuations, there is at least one test case
- ▶ T -wise testing: for each T variables and each T partitions of their valuations, there is at least one test case

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: $0 \leq \text{salary} \leq 10000$?	n	y	y	y	y	y	y	y	y
c1: male?	-	-	-	y	y	y	n	n	n
c2: too young? [...,18]	-	y	-	-	-	-	-	-	-
c3: young? m:[18,...,35], f:[18,...,30]	-	-	-	y	-	-	y	-	-
c4: mid? m:[36,...,45], f:[31,...,40]	-	-	-	-	y	-	-	y	-
c5: old? m:[46,...,55], f:[40,...,50]	-	-	-	-	-	y	-	-	y
c6: too old? m:[56,...], f:[51,...]	-	-	y	-	-	-	-	-	-
a1: wrong inputs	X	X	X	-	-	-	-	-	-
a2: $75 * \text{salary}$	-	-	-	X	-	-	-	-	-
a3: $70 * \text{salary}$	-	-	-	-	-	-	X	-	-
a4: $55 * \text{salary}$	-	-	-	-	X	-	-	-	-
a5: $50 * \text{salary}$	-	-	-	-	-	-	-	X	-
a6: $35 * \text{salary}$	-	-	-	-	-	-	-	-	X
a7: $30 * \text{salary}$	-	-	-	-	-	X	-	-	-

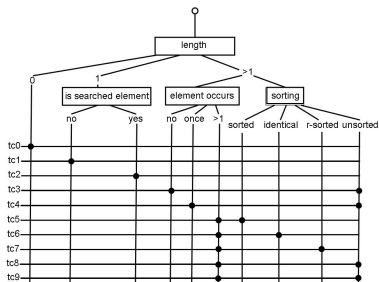
Decision Table for Testing

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

Basic Steps

Classification tree:

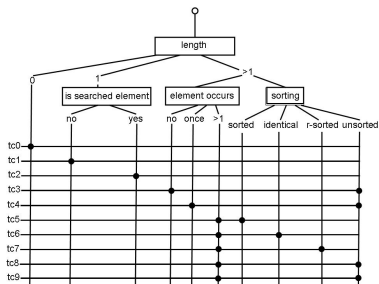
- ▶ Determine the **aspects** of specification influencing the **logic**
- ▶ Establish a **hierarchy** between aspects (the more **global** conditions **first**)
- ▶ **Partition** the input domain for each aspect
cover the whole domain of the "parent" node



Basic Steps

Combination table:

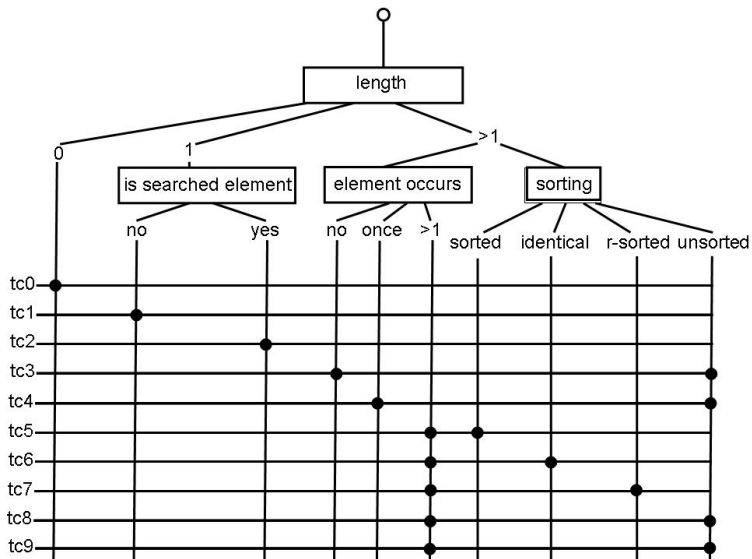
- Define a test-case for each relevant combination of inputs



Example

Informal Spec

Consider the function $count(list : List(EI), el : EI) : int$,
which takes a list of elements (with an order defined on them),
and an element
and output the number of occurrences of the element in the list.



Mortgage Example

Classes

1. Salary: -1 , $[0..10000]$, >10000 ,
2. Gender: Male, Female,
3. Age: Too young, Young, Middle, Old, Too old (**dependent** on gender)

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:
combination of classification tree (or DT)
with others should provide an effective mix.