SE-3002
SOFTWARE QUALITY ENGINEERING

RUBAB JAFFAR

RUBAB.JAFFAR@NU.EDU.PK

Part II-Software Testing

Functional Testing

Lecture # 16, 17, 18

04,05,06 Oct

TODAY'S OUTLINE

- Decision Table Based Testing
- Cause Effect Testing
- Pairwise Testing

WHY DO BOTH EP AND BVA?

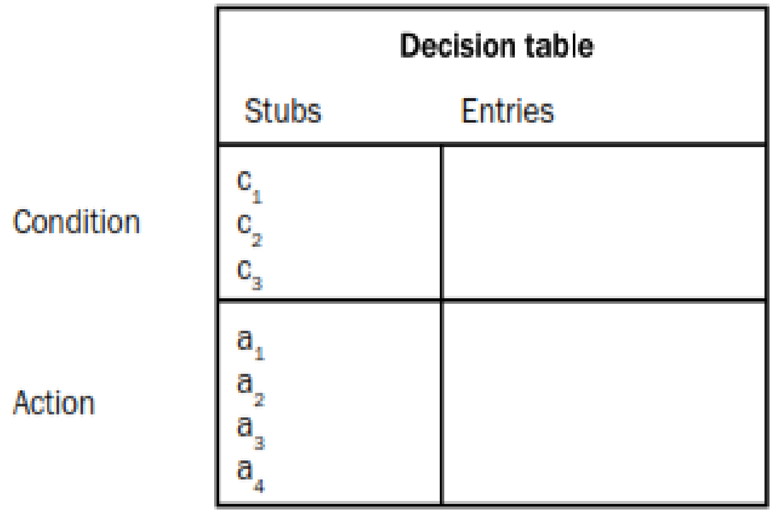
- If you do boundaries only, you have covered all the partitions as well
  - technically correct and may be OK if everything works correctly!
  - if the test fails, is the whole partition wrong, or is a boundary in the wrong place - have to test mid- partition anyway
  - testing only extremes may not give confidence for typical use scenarios (especially for users)
  - boundaries may be harder (more costly) to set up

DECISION TABLE BASED TESTING

- In Software Engineering, boundary value and equivalent partition are other similar techniques used to ensure better coverage.
- They are used if the system shows the **same** behavior for a large set of inputs.
- However, in a system where for each set of input values the system behavior is **different**, boundary value and equivalent partitioning technique are not effective in ensuring good test coverage.
- In this case, decision table testing is a good option. This technique can make sure of good coverage, and the representation is simple so that it is easy to interpret and use.
- This table can be used as the reference for the requirement and for the functionality development since it is easy to understand and cover all the combinations.

DECISION TABLE BASED TESTING

- Decision tables are used in many engineering disciplines to represent complex logical relationships.
- An output may be dependent on many input conditions and decision tables give a pictorial view of various combinations of input conditions.
- There are four portions of the decision table. The decision table provides a set of conditions and their corresponding actions.



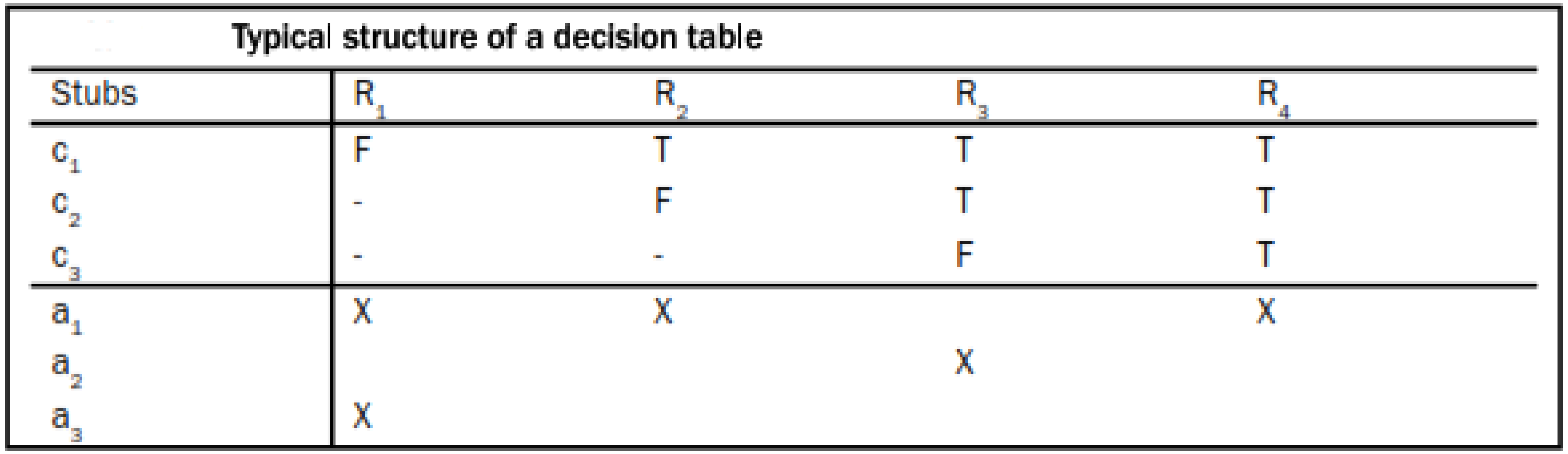
Four Portions

1. Condition Stubs
2. Condition Entries
3. Action Stubs
4. Action Entries

PARTS OF THE DECISION TABLE

- The four parts of the decision table are given as:
- Condition Stubs: All the conditions are represented in this upper left section of the decision table. These conditions are used to determine a particular action or set of actions.
- Action Stubs: All possible actions are listed in this lower left portion of the decision table.
- Condition Entries: In the condition entries portion of the decision table, we have a number of columns and each column represents a rule. Values entered in this upper right portion of the table are known as inputs.
- Action Entries: Each entry in the action entries portion has some associated action or set of actions in this lower right portion of the table. These values are known as outputs and are dependent upon the functionality of the program.

TYPICAL STRUCTURE OF DECISION TABLE

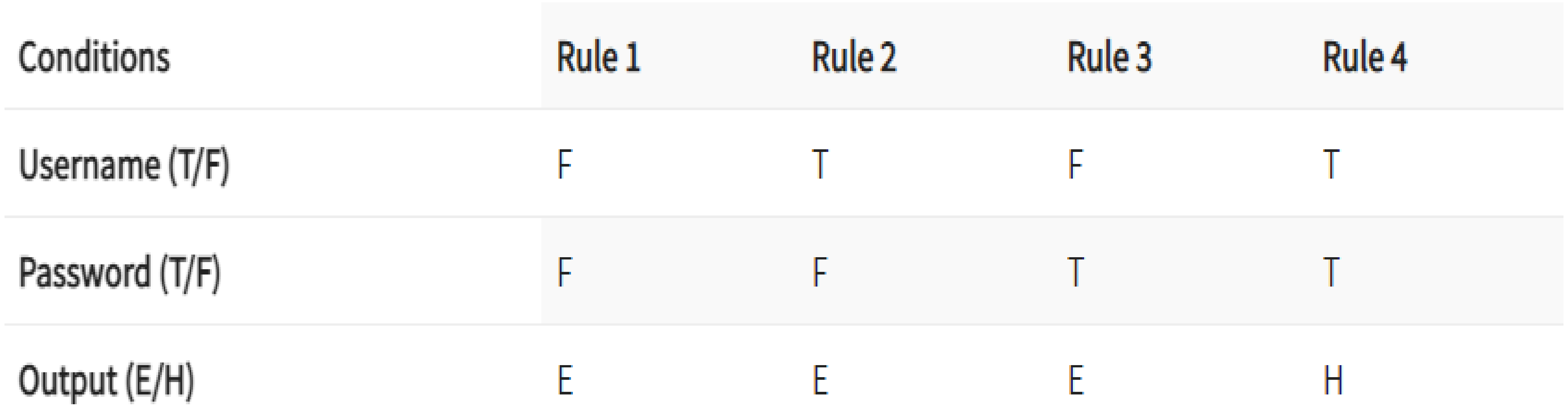


Typical structure of a decision table

Stubs	R_1	R_2	R_3	R_4
c_1	F	T	T	T
c_2	-	F	T	T
c_3	-	-	F	T
a_1	X	X		X
a_2			X	
a_3	X			

DECISION TABLE FOR A LOGIN SCREEN.

- The condition is simple if the user provides correct username and password the user will be redirected to the homepage. If any of the input is wrong, an error message will be displayed.
- Case 1 – Username and password both were wrong. The user is shown an error message.
- Case 2 – Username was correct, but the password was wrong. The user is shown an error message.
- Case 3 – Username was wrong, but the password was correct. The user is shown an error message.
- Case 4 – Username and password both were correct, and the user navigated to homepage

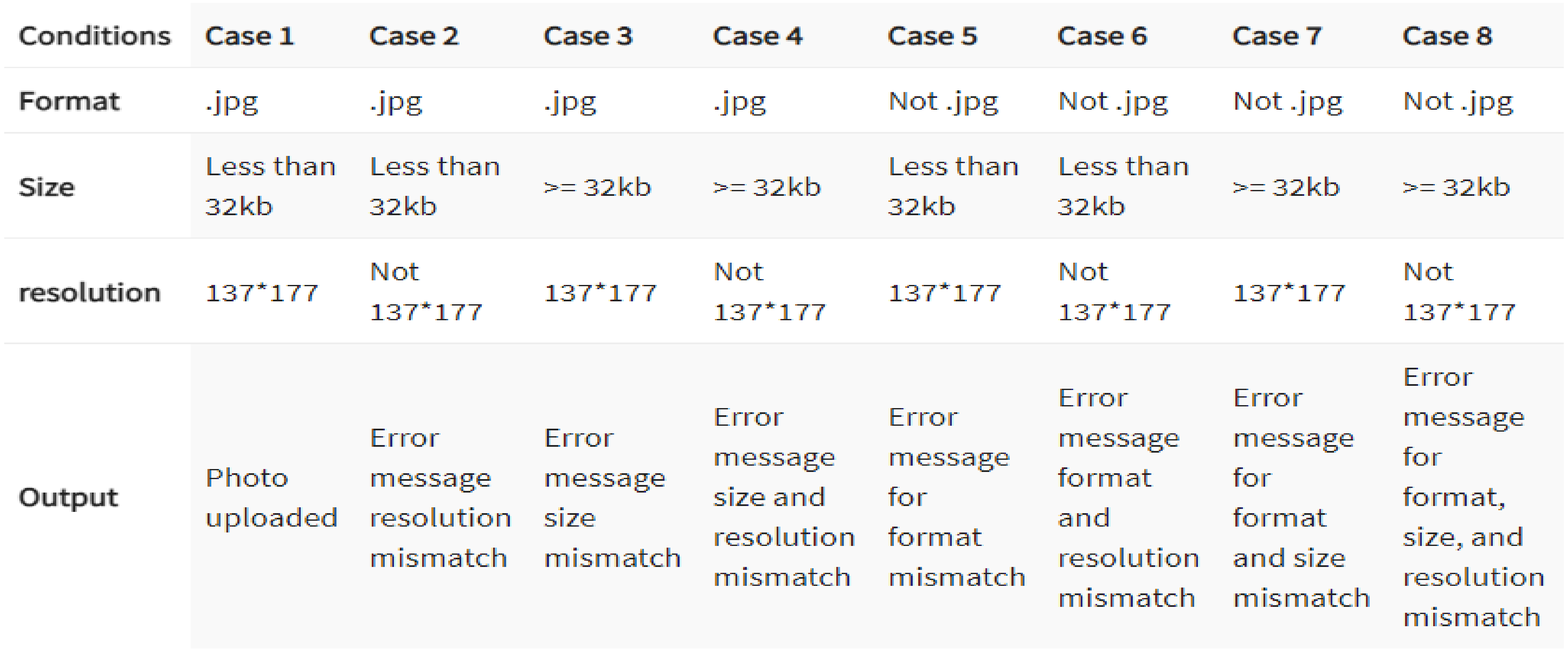


Conditions	Rule 1	Rule 2	Rule 3	Rule 4
Username (T/F)	F	T	F	T
Password (T/F)	F	F	T	T
Output (E/H)	E	E	E	H

EXAMPLE 2: DECISION TABLE FOR UPLOAD SCREEN

- Develop a decision table for an application that allows a user to upload an image. For uploading an image, application opens a dialogue box which will ask the user to upload photo with certain conditions like –
- You can upload only '.jpg' format image
- file size less than 32kb
- resolution 137*177.

DECISION TABLE FOR UPLOAD SCREEN



Conditions	Case 1	Case 2	Case 3	Case 4	Case 5	Case 6	Case 7	Case 8
Format	.jpg	.jpg	.jpg	.jpg	Not .jpg	Not .jpg	Not .jpg	Not .jpg
Size	Less than 32kb	Less than 32kb	>= 32kb	>= 32kb	Less than 32kb	Less than 32kb	>= 32kb	>= 32kb
resolution	137*177	Not 137*177	137*177	Not 137*177	137*177	Not 137*177	137*177	Not 137*177
Output	Photo uploaded	Error message resolution mismatch	Error message size mismatch	Error message size and resolution mismatch	Error message for format mismatch	Error message format and resolution mismatch	Error message for format and size mismatch	Error message for format, size, and resolution mismatch

YOUR TURN:

- A company’s online shopping cart applies different discount rules based on customer type, cart total, and promotional codes. The system should work as follows: Customer Type can be , Regular customer, Premium customer, Employee
- Cart Total amount can be Less than $100, Between $100 and $500, Greater than $500
- Promotional Code can be : No code, 10% discount code, Free shipping code
  - If the customer is an employee, they always receive a 30% discount, regardless of cart total or promotional code.
  - Premium customers get a 20% discount if their cart total exceeds $500.
  - Regular customers only get a 10% discount if they use the 10% discount code and their cart total is between $100 and $500.
  - The free shipping code only applies to customers with a cart total of $100 or more.
  - No discount is applied if the cart total is less than $100, unless the customer is an employee.
- Question:
- Using a decision table, design test cases to cover the different discount scenarios for this online shopping cart.

APPLICABILITY

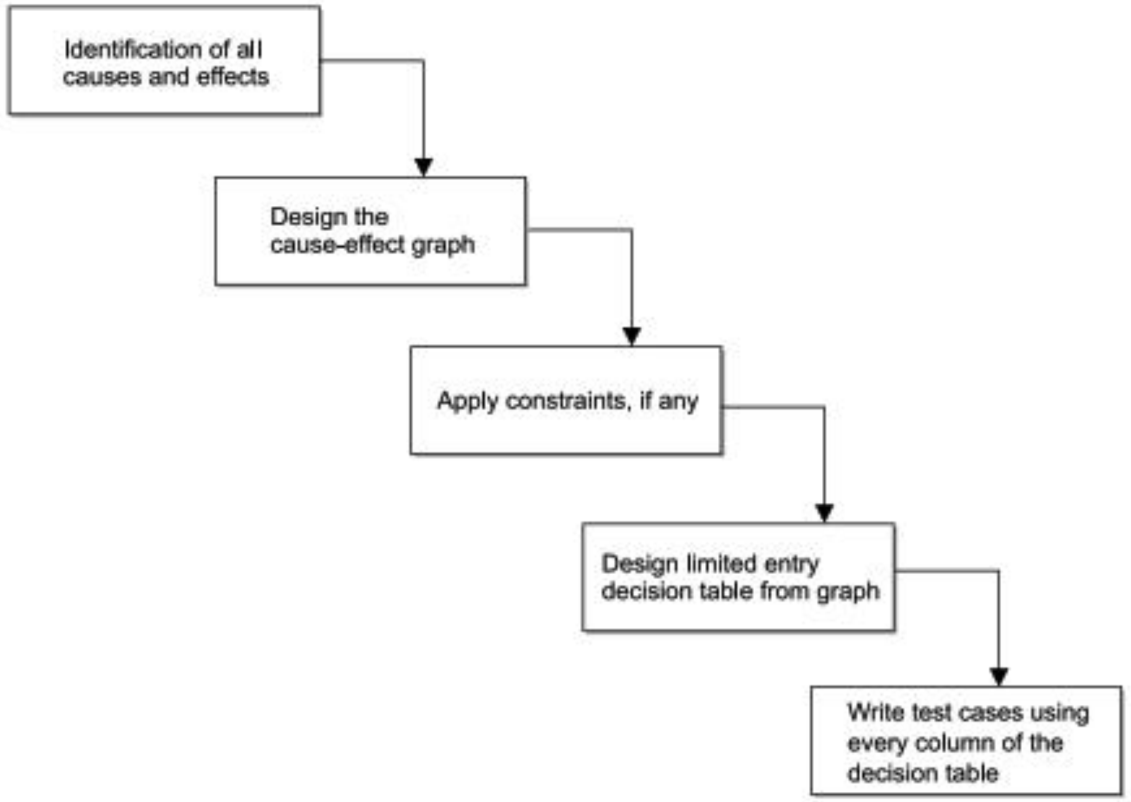
- Decision tables are popular in circumstances where an output is dependent on many conditions and a large number of decisions are required to be taken.
- They may also incorporate complex business rules and use them to design test cases.
- Every column of the decision table generates a test case.
- As the size of the program increases, handling of decision tables becomes difficult and cumbersome.
- In practice, they can be applied easily at unit level only. System testing and integration testing may not find its effective applications.

APPLICABILITY

- It's a tabular representation of input conditions and resulting actions. Additionally, it shows the causes and effects. Therefore, this technique is also called a ***cause-effect table.***
- Testing combinations can be a challenge, especially if the number of combinations is enormous. Moreover, testing all combinations is not practically feasible as it's not cost and time effective. Therefore, we have to be satisfied with testing just a small subset of combinations. That is to say, the success of this technique depends on our choice of combinations.

CAUSE-EFFECT GRAPHING TECHNIQUE

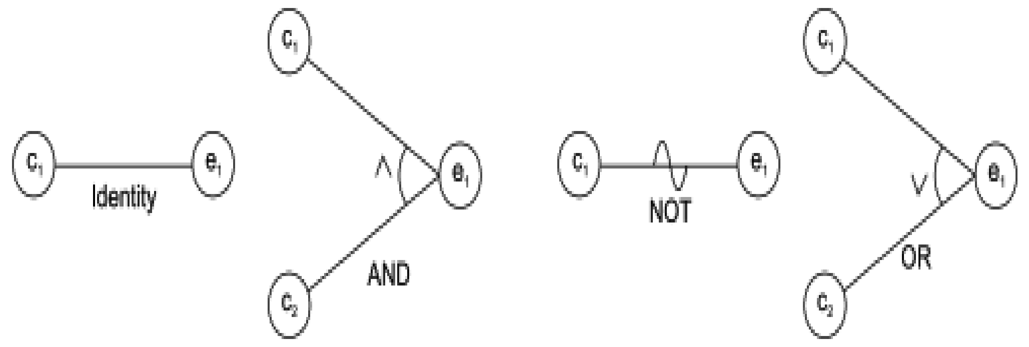
- Popular technique for small programs and considers the combinations of various inputs.
- Two terms: **Causes and Effects**, which are nothing but inputs and outputs respectively.
- SRS document is used for the identification of causes and effects.
- A list is prepared for all causes and effects.


Identification of all causes and effects
Design the cause-effect graph
Apply constraints, if any
Design limited entry decision table from graph
Write test cases using every column of the decision table

Steps for the generation of test cases

DESIGN OF CAUSE-EFFECT GRAPH

- Each node represents either true or false state and may be assigned 1 and 0 value respectively.



Basic notations used in cause-effect graph

(a) Identity: This function states that if c_1 is 1, then e_1 is 1; else e_1 is 0.
(b) NOT: This function states that if c_1 is 1, then e_1 is 0; else e_1 is 1.
(c) AND: This function states that if both c_1 and c_2 are 1, then e_1 is 1; else e_1 is 0.
(d) OR: This function states that if either c_1 or c_2 is 1, then e_1 is 1; else e_1 is 0.

The AND and OR functions are allowed to have any number of inputs.

USE OF CONSTRAINTS IN CAUSE-EFFECT GRAPH

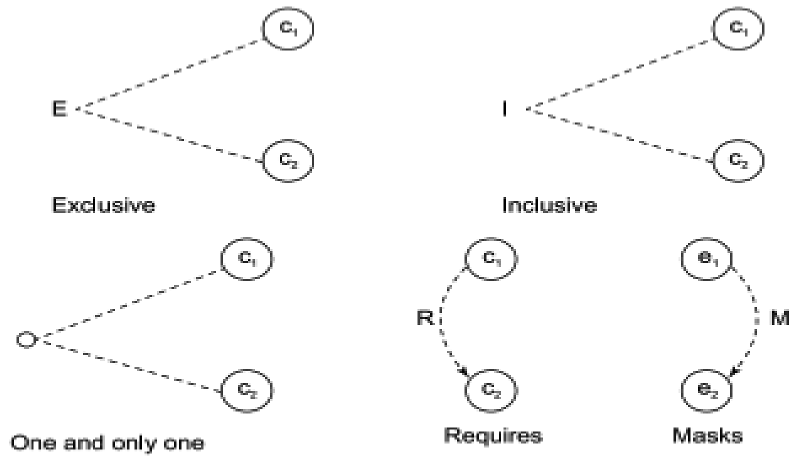
(a) Exclusive
The Exclusive (E) constraint states that at most one of c_1 or c_2 can be 1 (c_1 or c_2 cannot be 1 simultaneously). However, both c_1 and c_2 can be 0 simultaneously.

(b) Inclusive
The Inclusive (I) constraints states that at least one of c_1 or c_2 must always be 1. Hence, both cannot be 0 simultaneously. However, both can be 1.

(c) One and Only One
The one and only one (O) constraint states that one and only one of c_1 and c_2 must be 1.

(d) Requires
The requires (R) constraint states that for c_1 to be 1, c_2 must be 1; it is impossible for c_1 to be 1 if c_2 is 0.

(e) Mask
This constraint is applicable at the effect side of the cause-effect graph. This states that if effect e_1 is 1, effect e_2 is forced to be 0.



Constraint symbols for any cause-effect graph

SUMMARY OF THE STEPS

- Draw the circles for effects and Causes.
- Start from effect and then pick up what is the cause of this effect.
- Draw mutually exclusive causes (exclusive causes which are directly connected via one effect and one cause) at last.
- Use logic gates to draw dynamic test cases.

EXAMPLE

- Consider the example of keeping the record of marital status and number of children of a citizen. The value of marital status must be 'U' or 'M'. The value of the number of children must be digit or null in case a citizen is unmarried. If the information entered by the user is correct then an update is made. If the value of marital status of the citizen is incorrect, then the error message 1 is issued. Similarly, if the value of number of children is incorrect, then the error message 2 is issued.

causes are:

c_1: marital status is 'U'

c_2: marital status is 'M'

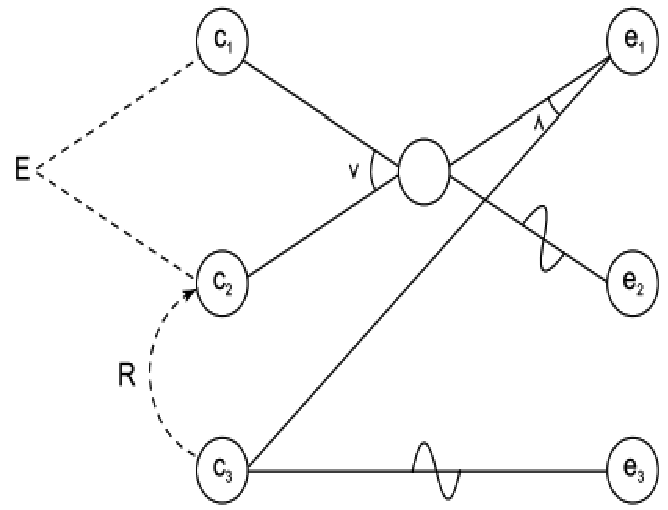
c_3: number of children is a digit

effects are:

e_1: updation made

e_2: error message 1 is issued

e_3: error message 2 is issued



EXAMPLE

- A tourist of age greater than 21 years and having a clean driving record is supplied a rental car. A premium amount is also charged if the tourist is on business, otherwise it is not charged. If the tourist is less than 21 year old, or does not have a clean driving record, the system will display the following message: “Car cannot be supplied”
- Draw the cause-effect graph and generate test cases.

APPLICABILITY

- Cause-effect graphing considers dependency of inputs using some constraints.
- Effective only for small programs because, as the size of the program increases, the number of causes and effects also increases and thus complexity of the cause-effect graph increases.
- For large-sized programs, a tool may help us to design the cause-effect graph with the minimum possible complexity.
- Limited applications in unit testing and hardly any application in integration testing and system testing.

PAIR-WISE TESTING

- **Pairwise Testing** is a test design technique that tries to delivers hundred percent test coverage.
- *A black-box test design technique in which test cases are designed to execute all possible discrete combinations of each pair of input parameters.*
- Techniques like boundary value analysis and equivalence partitioning can be useful to identify the possible values for individual factors. But it is impractical to test all possible combinations of values for all those factors. So instead **a subset of combinations is generated** to satisfy all factors.
- All-Pairs technique is very helpful for designing tests for applications involving multiple parameters. Tests are designed such that for each pair of input parameters to a system, there are all possible discrete combinations of those parameters. The test suite covers all combinations; therefore it is not exhaustive yet very effective in finding bugs.

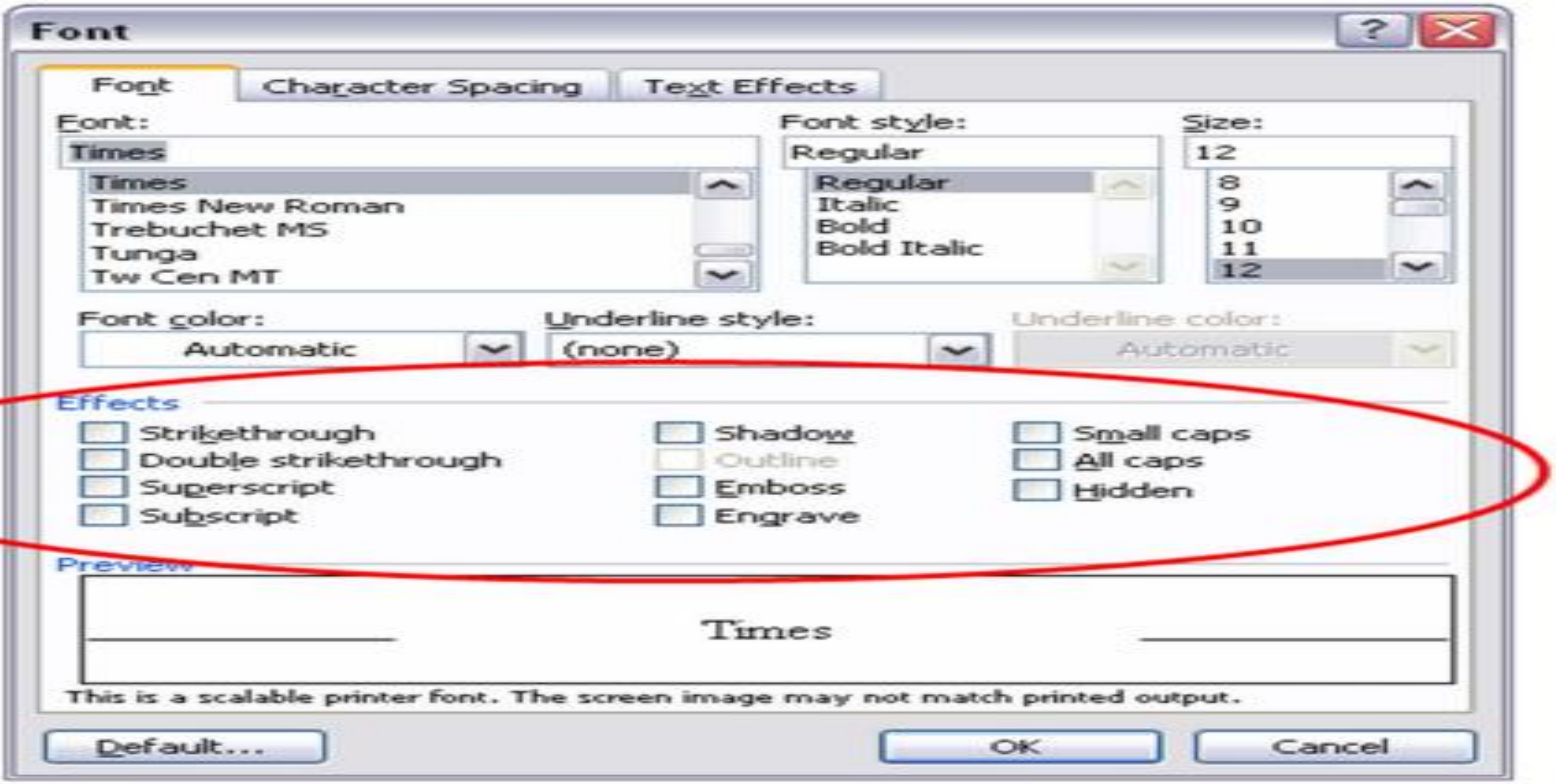
EXAMPLE- COMBINATIONAL TESTING

- An application with simple list box with 10 elements (Let's say 0,1,2,3,4,5,6,7,8,9) along with a checkbox, radio button, Text Box and OK Button. The Constraint for the Text box is it can accept values only between 1 and 100. Below are the values that each one of the GUI objects can take :
- List Box - 0,1,2,3,4,5,6,7,8,9
- Check Box - Checked or Unchecked
- Radio Button - ON or OFF
- Text Box - Any Value between 1 and 100
- Exhaustive combination of the product B is calculated
  - List Box = 10
  - Check Box = 2
  - Radio Button = 2
  - Text Box = 100
- Total Number of Test Cases using Cartesian Method : 10*2*2*100 = 4000
- Total Number of Test Cases including Negative Cases will be > 4000

SOME EXAMPLES

- Suppose we have a system with on-off switches:

SOME EXAMPLES


Font

Font | Character Spacing | Text Effects

Font: Times
Times
Times New Roman
Trebuchet MS
Tunga
Tw Cen MT

Font style: Regular
Regular
Italic
Bold
Bold Italic

Size: 12
8
9
10
11
12

Font color: Automatic
Underline style: (none)
Underline color: Automatic

Effects
- [ ] Strikethrough
- [ ] Double strikethrough
- [ ] Superscript
- [ ] Subscript
- [ ] Shadow
- [ ] Outline
- [ ] Emboss
- [ ] Engrave
- [ ] Small caps
- [ ] All caps
- [ ] Hidden

Preview

Times

This is a scalable printer font. The screen image may not match printed output.

Default... | OK | Cancel

SOME MORE EXAMPLES

- **Car Ordering Application:**
- The car ordering application allows for Buying and Selling cars. It should support trading in Delhi and Mumbai.
- The application should have registration numbers, may be valid or invalid. It should allow the trade of following cars: BMW, Audi, and Mercedes.
- Two types of booking can be done: E-booking and In Store.
- Orders can be placed only during trading hours.

STEP #1: LET'S LIST DOWN THE VARIABLES INVOLVED.

- **1)** Order category
  a. Buy
  b. Sell
- **2)** Location
  a. Delhi
  b. Mumbai
- **3)** Car brand
  a. BMW
  b. Audi
  c. Mercedes
- **4)** Registration numbers
  a. Valid (5000)
  b. Invalid
- **5)** Order type
  a. E-Booking
  b. In-store
- **6)** Order time
  a. Working hours
  b. Non-working hours

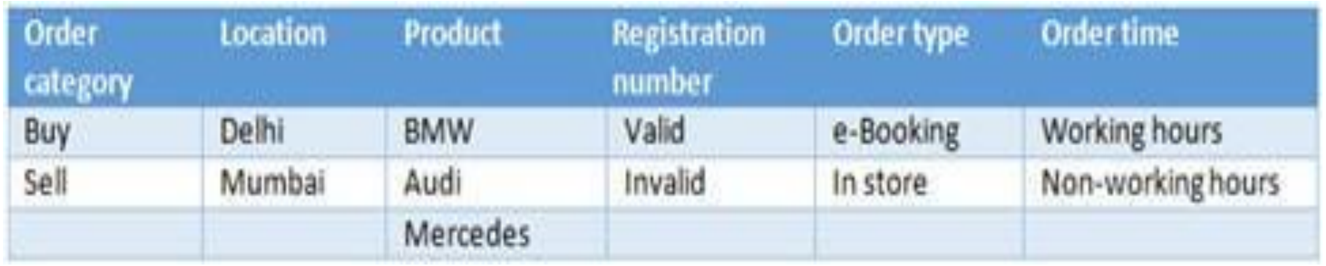
If we want to test all possible valid combinations:
= 2 X 2 X 3 X 5000 X 2 X 2
= 240000 Valid test cases combinations :(
There is also an infinite number of invalid combinations.

STEP #2: LET’S SIMPLIFY

- – Use a smart representative sample.
  – Use groups and boundaries, even when data is non-discrete.
  – Reduce Registration Number to Two
- Valid registration number
- Invalid registration number
- Now let’s calculate the number of possible combinations
  = 2 X 2 X 3 X 2 X 2 X 2
  = 96

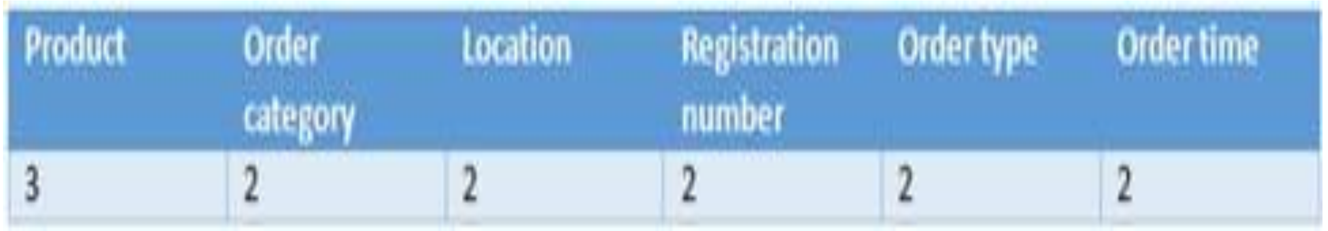
STEP #3: ARRANGING VARIABLES AND VALUES INVOLVED.

- When we arrange variables and values involved, it looks something like this.



Order category	Location	Product	Registration number	Order type	Order time
Buy	Delhi	BMW	Valid	e-Booking	Working hours
Sell	Mumbai	Audi	Invalid	In store	Non-working hours
		Mercedes			

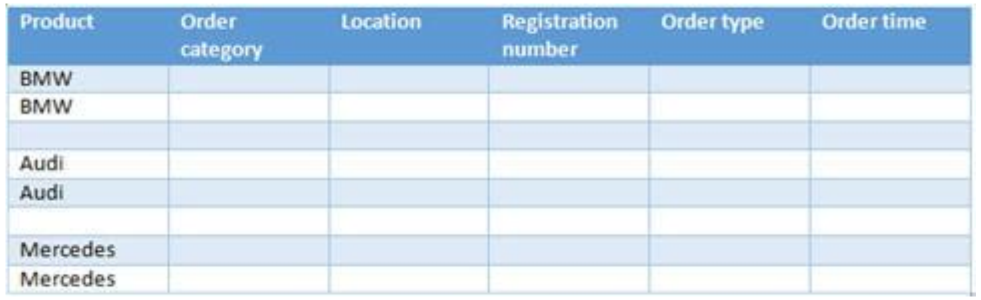
- Now order the variables so that the one with the most number of values is first and the least is last.



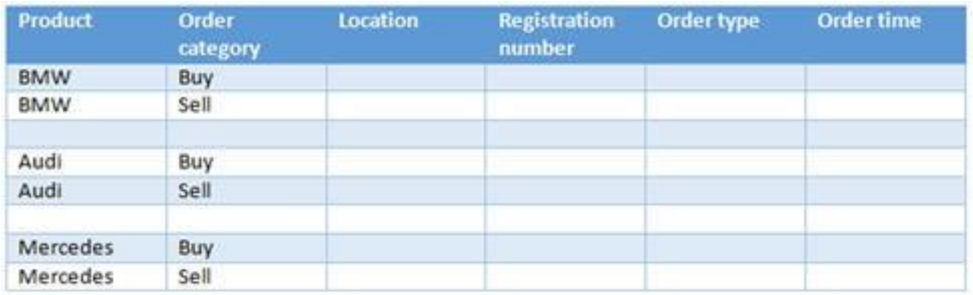
Product	Order category	Location	Registration number	Order type	Order time
3	2	2	2	2	2

STEP #4: ARRANGE VARIABLES TO CREATE A TEST SUITE

- Start filling in the table column by column. Initially, the table should look something like this. The three values of **Product** (variable having the highest number of values) should be written two times each (two is the number of values of next highest variable i.e. **Order category**).

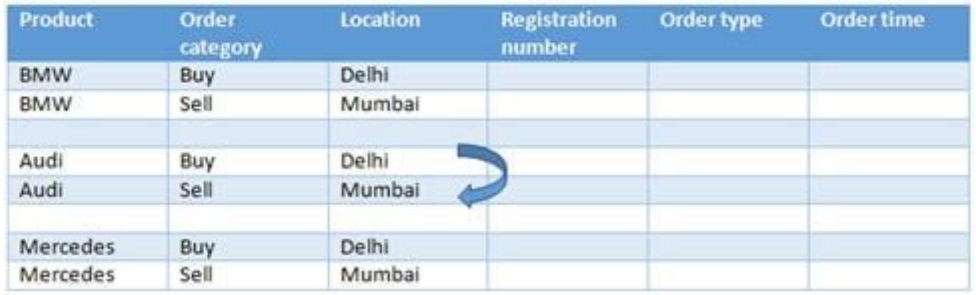


Product	Order category	Location	Registration number	Order type	Order time
BMW					
BMW					
Audi					
Audi					
Mercedes					
Mercedes					



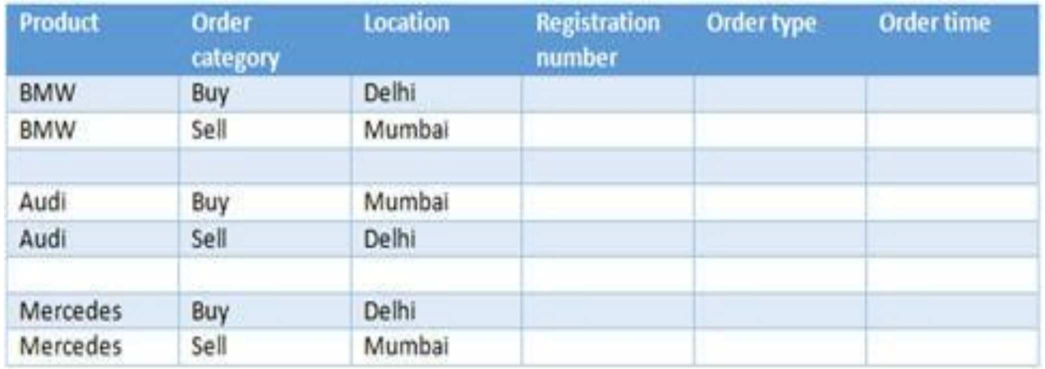
Product	Order category	Location	Registration number	Order type	Order time
BMW	Buy				
BMW	Sell				
Audi	Buy				
Audi	Sell				
Mercedes	Buy				
Mercedes	Sell				

- For each set of values in column 1, we put both values of column 2. Repeat the same for column 3.



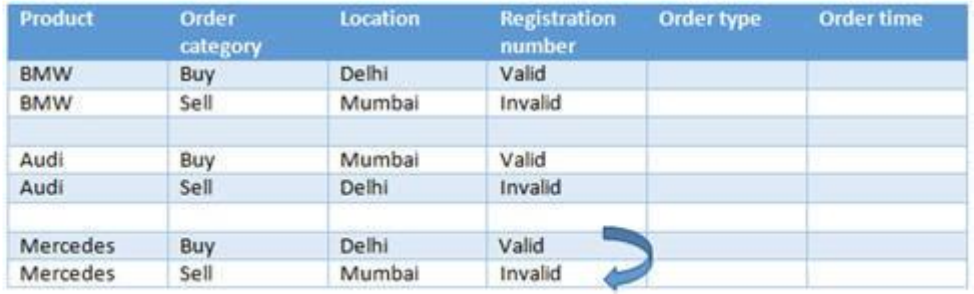
Product	Order category	Location	Registration number	Order type	Order time
BMW	Buy	Delhi			
BMW	Sell	Mumbai			
Audi	Buy	Delhi			
Audi	Sell	Mumbai			
Mercedes	Buy	Delhi			
Mercedes	Sell	Mumbai			

- We have a Buy and Delhi, but wait – there's no Buy and Mumbai. We have a Sell and Mumbai, but there's no Sell and Delhi. Let's swap around the values in the second set in the third column.

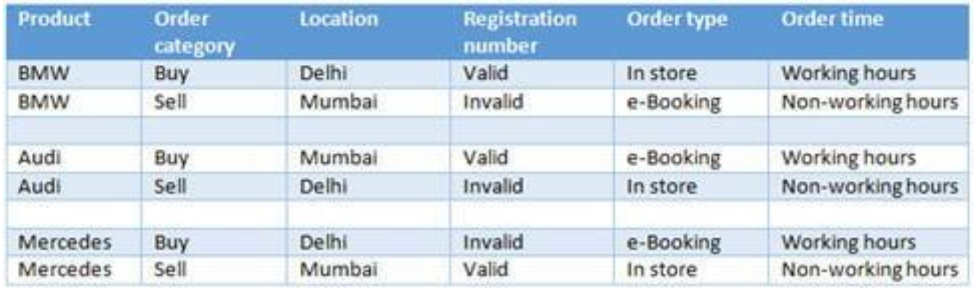


Product	Order category	Location	Registration number	Order type	Order time
BMW	Buy	Delhi			
BMW	Sell	Mumbai			
Audi	Buy	Mumbai			
Audi	Sell	Delhi			
Mercedes	Buy	Delhi			
Mercedes	Sell	Mumbai			

We will repeat the same steps for column 3 and 4.



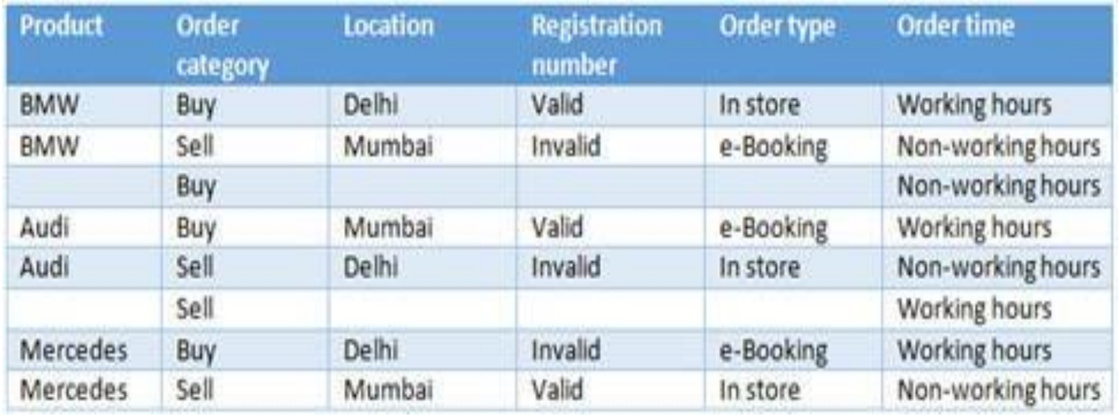
Product	Order category	Location	Registration number	Order type	Order time
BMW	Buy	Delhi	Valid		
BMW	Sell	Mumbai	Invalid		
Audi	Buy	Mumbai	Valid		
Audi	Sell	Delhi	Invalid		
Mercedes	Buy	Delhi	Valid		
Mercedes	Sell	Mumbai	Invalid		



Product	Order category	Location	Registration number	Order type	Order time
BMW	Buy	Delhi	Valid	In store	Working hours
BMW	Sell	Mumbai	Invalid	e-Booking	Non-working hours
Audi	Buy	Mumbai	Valid	e-Booking	Working hours
Audi	Sell	Delhi	Invalid	In store	Non-working hours
Mercedes	Buy	Delhi	Invalid	e-Booking	Working hours
Mercedes	Sell	Mumbai	Valid	In store	Non-working hours

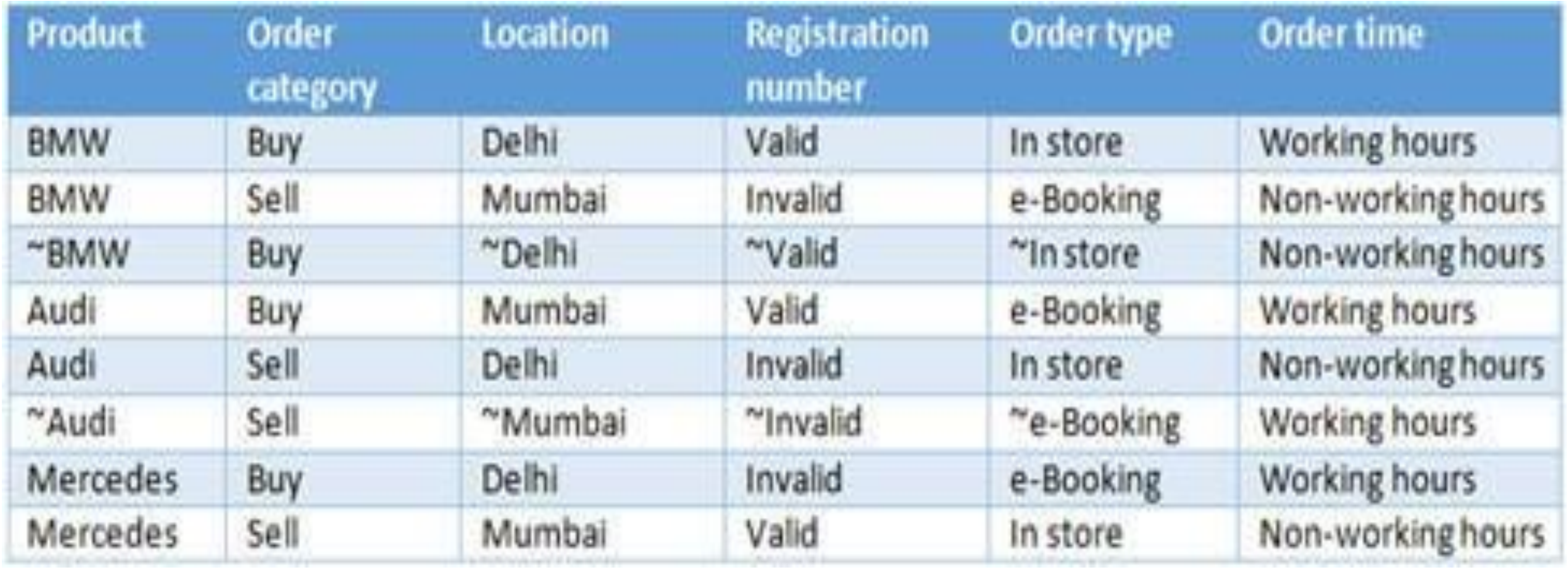
HURRAY! ALL PAIRS IN 8 CASES, INSTEAD OF ALL COMBINATIONS IN 96!

- Column 6 (Order time) is problematic. We are missing Buy/Non-working hours and Sell/Working hours. We can't fit our missing pairs by swapping around values as we already swapped all the rows if we swap now we may miss other possible pairs which are already sorted. So, we add two more test cases that contain these pairs. Hence, the blank rows!



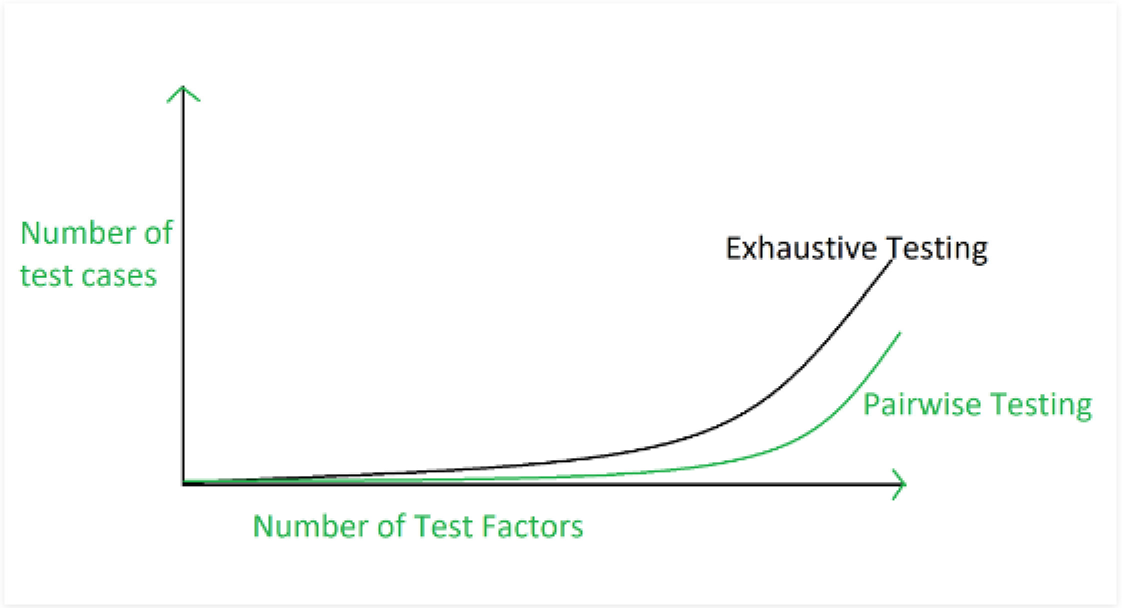
Product	Order category	Location	Registration number	Order type	Order time
BMW	Buy	Delhi	Valid	In store	Working hours
BMW	Sell	Mumbai	Invalid	e-Booking	Non-working hours
	Buy				Non-working hours
Audi	Buy	Mumbai	Valid	e-Booking	Working hours
Audi	Sell	Delhi	Invalid	In store	Non-working hours
	Sell				Working hours
Mercedes	Buy	Delhi	Invalid	e-Booking	Working hours
Mercedes	Sell	Mumbai	Valid	In store	Non-working hours

HURRAY! ALL PAIRS IN 8 CASES, INSTEAD OF ALL COMBINATIONS IN 96!



Product	Order category	Location	Registration number	Order type	Order time
BMW	Buy	Delhi	Valid	In store	Working hours
BMW	Sell	Mumbai	Invalid	e-Booking	Non-working hours
~BMW	Buy	~Delhi	~Valid	~In store	Non-working hours
Audi	Buy	Mumbai	Valid	e-Booking	Working hours
Audi	Sell	Delhi	Invalid	In store	Non-working hours
~Audi	Sell	~Mumbai	~Invalid	~e-Booking	Working hours
Mercedes	Buy	Delhi	Invalid	e-Booking	Working hours
Mercedes	Sell	Mumbai	Valid	In store	Non-working hours

GRAPHICAL REPRESENTATION OF PAIRWISE TESTING



ADVANTAGES & DISADVANTAGES OF PAIRWISE TESTING

- Pairwise testing reduces the number of execution of test cases.
- Pairwise testing increases the test coverage almost up to hundred percentage.
- Pairwise testing increases the defect detection ratio.
- Pairwise testing takes less time to complete the execution of the test suite.
- Pairwise testing reduces the overall testing budget for a project.

- Pairwise testing is not beneficial if the values of the variables are inappropriate.
- In pairwise testing it is possible to miss the highly probable combination while selecting the test data.
- In pairwise testing, defect yield ratio may be reduced if a combination is missed.
- Pairwise testing is not useful if combinations of variables are not understood correctly.

That is all