Equivalence class partitioning

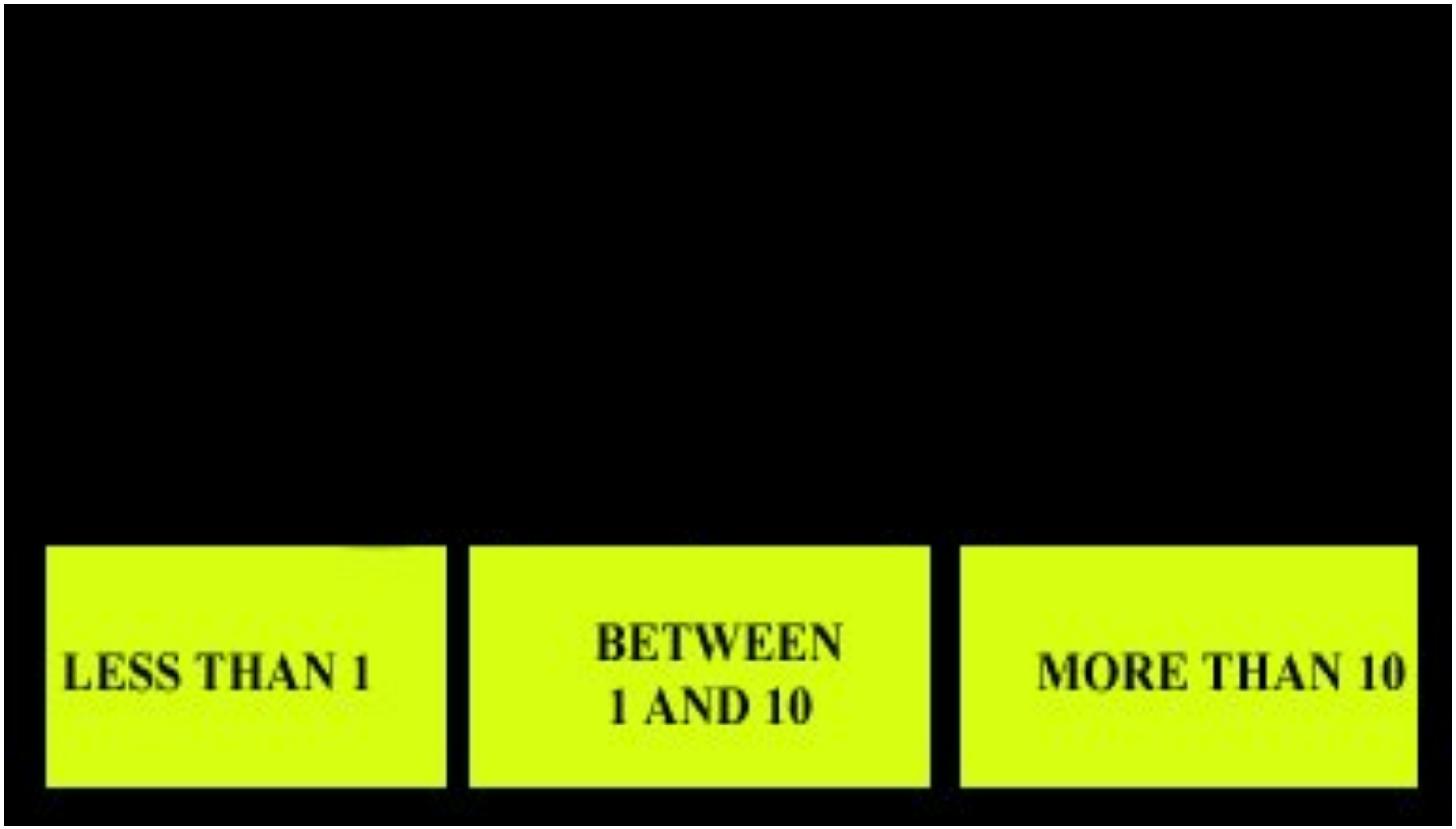
Classifying data-based black box testing

- All techniques
  - Requirements: data-based (some logical/mathematical input/output relation)
  - Purpose: unit testing, functional coding mistakes
  - Technique: black box, data-based
  - Assumption: both single and multiple-fault assumption
- Boundary value testing
  - Purpose: typical errors (boundaries of domains)
- Equivalence partitioning/decision tables
  - Purpose: input domain coverage, efficiency in # of tests

INTRODUCTORY EXAMPLE

- Let’s suppose that you wanted to test a program that only accepted integer values from 1 to 10.
- The possible test cases for such a program would be the range of all integers.
- How can we narrow the number of test cases down from all integers to a few good test cases?

ILLUSTRATION PROVIDED



GENERAL IDEA

- Equivalence partitioning is the process of taking all of the possible test values and placing them into classes (groups).
- Each element of a class should be “equivalent” in its likeliness to expose a bug.
- Then, we only need to use a few test cases per class to represent all test values.

GOOD TEST CASES

- A good test case has a reasonable probability of finding an error.
- In the previous program, all integers up to 0 and beyond 10 will elicit an error.
- There must be a way of reducing all possible test cases into a small subset.
- This small subset should have the highest probability of finding the most errors.

BEST TEST CASE SUBSET 1

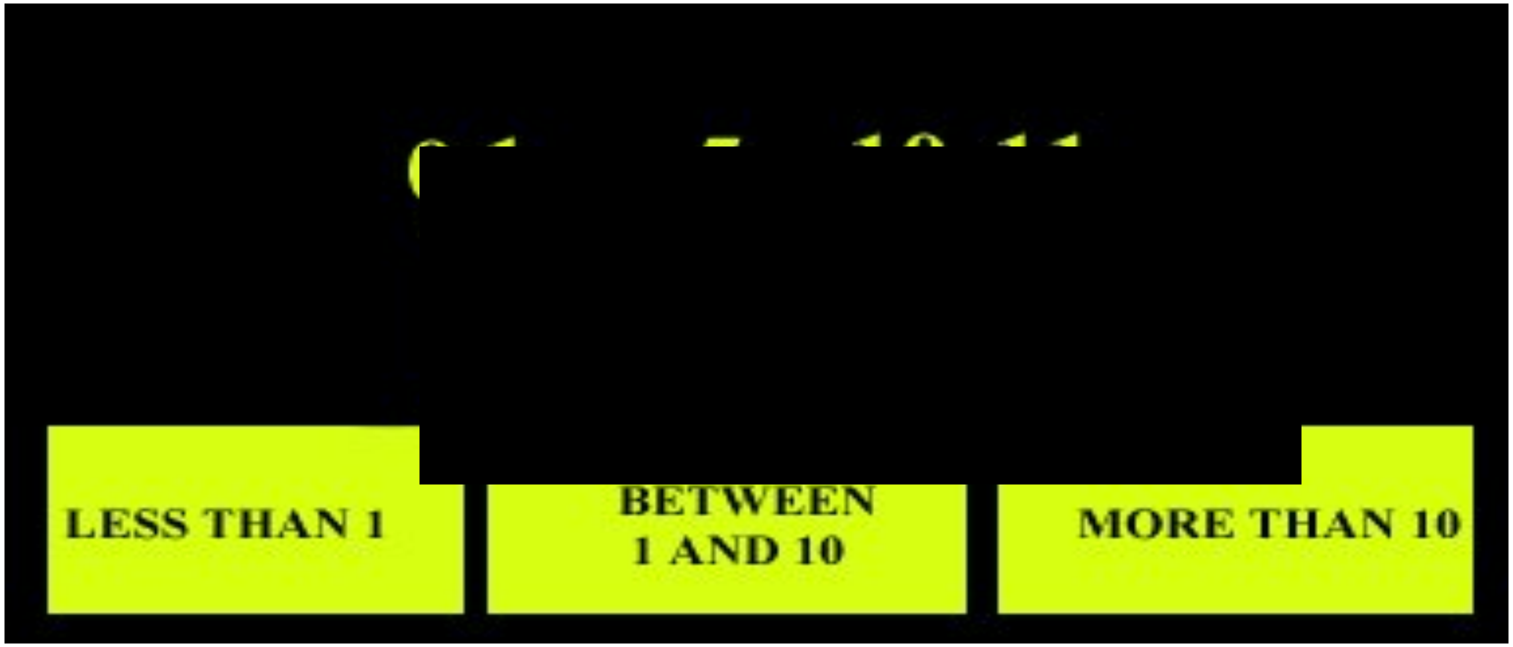
- A well-selected test case has two other properties:
- It reduces, by more than a count of one, the number of other test cases that must be developed to achieve “reasonable testing.”
- It does this by invoking as many different input conditions as possible.

BEST TEST CASE SUBSET 2

- The second property of a well-selected test case is that it covers a large set of other possible test cases.
- If such a test case detects an error, it is reasonable to assume that all other values in that subset of test cases will detect an error as well. If such a test case does not detect an error, assume that none will.

EXAMPLE PROVIDED

- It is reasonable to assume that if 11 fails, any value greater than 11 will fail.



SYNTHESIS OF IDEAS

- These two other properties of a well-selected test case form the black-box methodology of equivalence partitioning.
- First, develop a set of “interesting” conditions to be tested. In other words, identify the equivalence classes.
- Second, develop a minimal set of test cases that will cover those classes.

IDENTIFYING EQUIV. CLASSES

- Identify each input condition, it’s usually a sentence or phrase in the specification.
- Partition each condition into two or more groups.
- Valid equivalence classes represent valid inputs to the program.
- Invalid equivalence classes represent other states of the condition (erroneous input).

Few more tips

- Equivalence class for invalid inputs
  - Looks for Range in numbers
  - Look for membership in a group
  - Analyze responses to lists and menus
  - Looks for variables that must be equal
  - Create time-determined equivalence classes
  - Look for equivalent output events
  - Look for variable groups that must calculate to a certain value or range
  - Look for equivalent operating environments

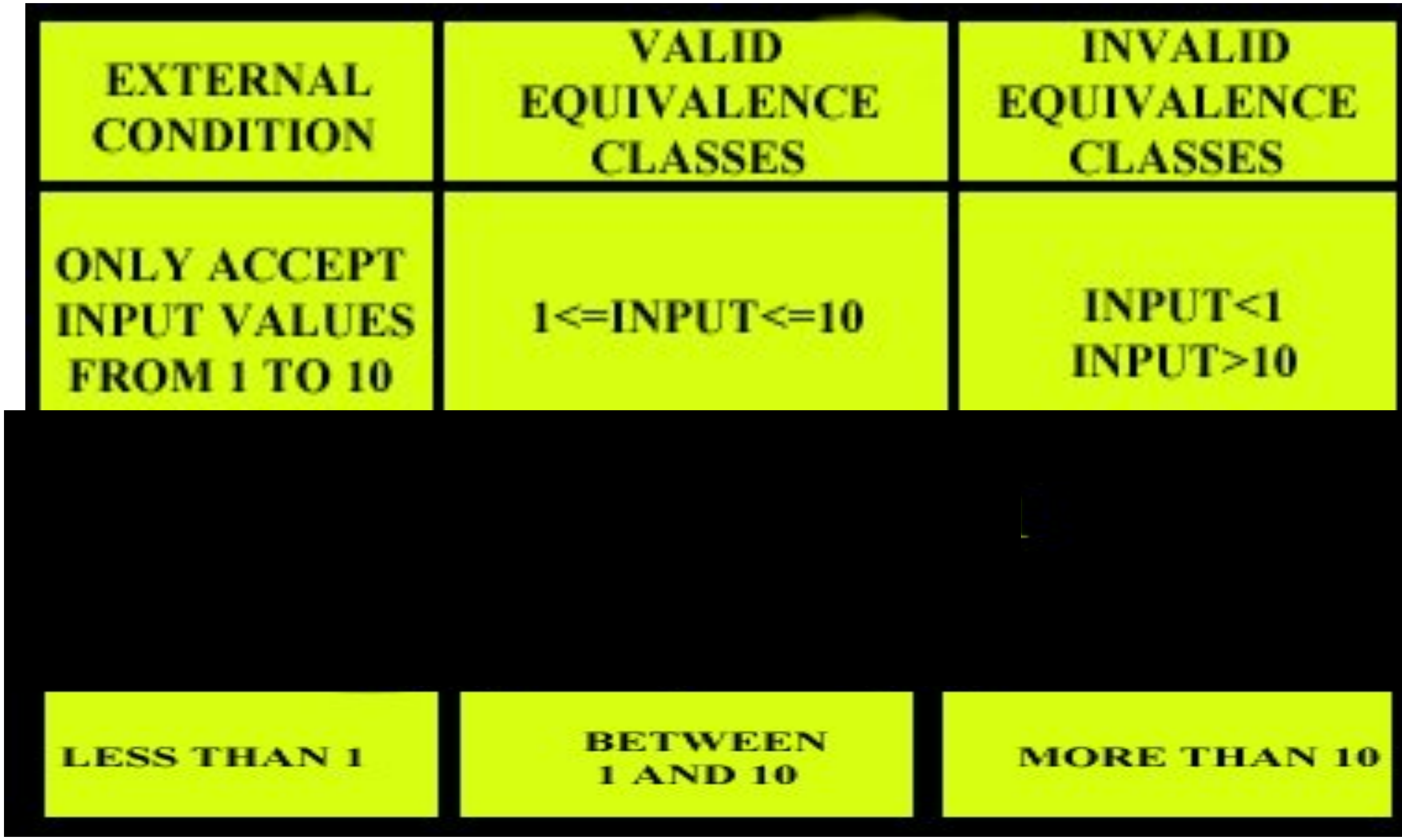
Equivalence Class

- It is one of the black box testing technique
- The input and output domain is partitioned into mutually exclusive parts i.e. disjoint sets (i.e. No overlap with input or output values does not overlap)
- The partition should be made in such a way that any one sample from a class is representative of the entire class
- Analyze before generation of test cases both output conditions and input conditions for the case considered

Equivalence Class Testing

- It is a technique of testing where input values are divided into valid and invalid inputs.
- Applied when
  - It is used in situations where exhaustive testing is desired
  - When there is a need to avoid redundancy
  -
  - Types of Equivalence Class Testing
    - Weak Normal Equivalence Class testing
    - Strong Normal Equivalence Class testing
    - Weak Robust Equivalence Class testing
    - Strong Robust Equivalence Class testing

EXAMPLE PROVIDED

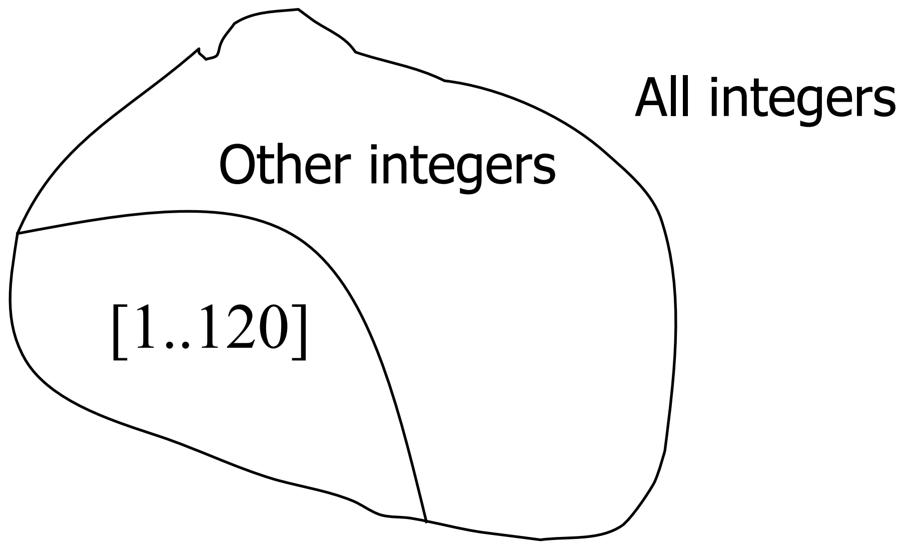


EXTERNAL CONDITION	VALID EQUIVALENCE CLASSES	INVALID EQUIVALENCE CLASSES
ONLY ACCEPT INPUT VALUES FROM 1 TO 10	1<=INPUT<=10	INPUT<1 INPUT>10

LESS THAN 1	BETWEEN 1 AND 10	MORE THAN 10

Sub domain classification

Consider an application A that takes an integer denoted by age as input. Let us suppose that the only legal values of age are in the range [1..120]. The set of input values is now divided into a set E containing all integers in the range [1..120] and a set U containing the remaining integers.

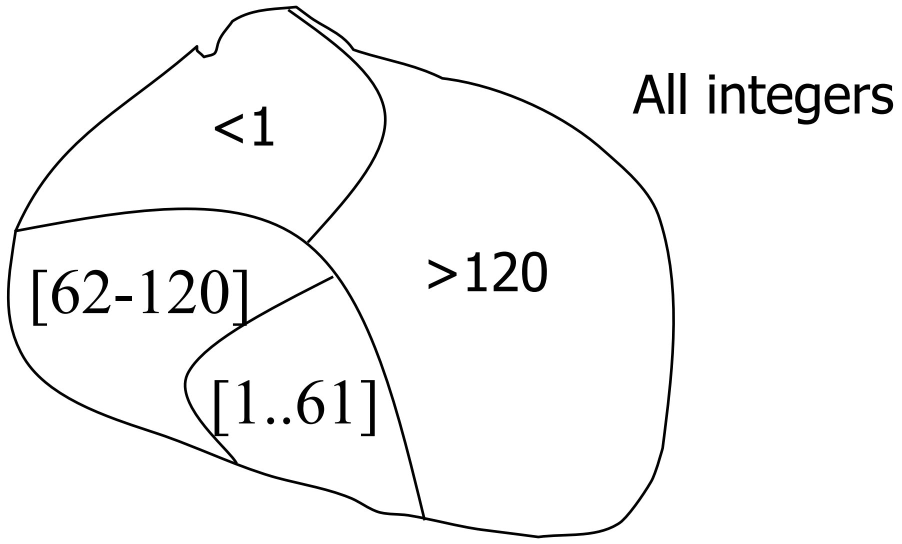


Sub domain classification(contd.)

Further, assume that the application is required to process all values in the range [1..61] in accordance with requirement R1 and those in the range [62..120] according to requirement R2.
Thus E is further subdivided into two regions depending on the expected behavior.

Similarly, it is expected that all invalid inputs less than or equal to 1 are to be treated in one way while all greater than 120 are to be treated differently. This leads to a subdivision of U into two categories.

Sub domain classification(contd.)



GUIDELINES 1

- Identifying equivalence classes is mainly an intuitive process. However, guidelines exist to assist with such identifications.
- As in the previous example, if an input condition is a particular range of values, let one valid equivalence class be the range.
- Let the values below and above the range be two respective invalid equivalence classes.

GUIDELINES 2

- If an input condition specifies a number of values:
- Identify one valid equivalence class:
    - The range from 1 to the number of values.
- Identify two invalid equivalence classes:
    - Zero values
    - More than the number of values

EXAMPLE PROVIDED

- Requirement: “The names of 1 to 3 references must be entered on the form.”
- One valid equivalence class would be 1<=names<=3
- One invalid equivalence class would be zero names.
- Another invalid equivalence class would be names >3.

GUIDELINES 3

- For input conditions requiring the use of enumeration values, do the following:
- Identify a valid equivalence class for each enumeration value.
- Identify one invalid equivalence class representing values that are not defined in the enumeration type.

EXAMPLE PROVIDED

- Suppose an input condition states, “The printer color must be black, magenta, cyan, or yellow.”
- Valid equivalence classes will be:
  - Black
  - Magenta
  - Cyan
  - Yellow
- The invalid equivalence class represents all other values, i.e. Pink

GUIDELINES 4

- If the input condition specifies a mandatory ("must be") condition:
- Identify one valid equivalence class representing the condition as being met.
- Identify one invalid equivalence class representing values that do not meet the condition.

EXAMPLE PROVIDED

- Suppose an input condition states “The first character of a code must be a digit.”
- The one valid equivalence class may contain the value 24432L.
- The one invalid equivalence class may contain a value such as G90125.

FINAL GUIDELINE

- After creating an equivalence class, ensure that the members will be handled in an identical manner by the program.
- If this does not happen to be the case, split the equivalence class into smaller equivalence classes.

IDENTIFYING TEST CASES

- After identifying the equivalence classes, identify the test cases.
- Assign a unique number to each equivalence class.
- Until all valid equivalence classes have been covered by test cases, write a new test case covering as many of the uncovered valid equivalence classes as possible.

IDENTIFYING TEST CASES

- Until all invalid equivalence classes have been covered by test cases, write a new test case that covers one, and only one, of the uncovered invalid equivalence classes as possible.
- The reason for this is that certain erroneous-input checks may supersede other erroneous-input checks.

ANALOGY NEEDED

- In programming languages, in an IF statement such as the following:
- if (var1=value1 AND var2=value2)
- If the var1=value1 condition evaluates to FALSE, the program will ignore, and therefore not test, the var2=value2 condition.

ANALOGY NEEDED

- In the same way, a test case that covers more than one invalid equivalence class may not evaluate the second condition if the first condition has a value of false.
- Likewise, a test case with two inputs may not check the second input for correctness if the first input is found to be invalid.

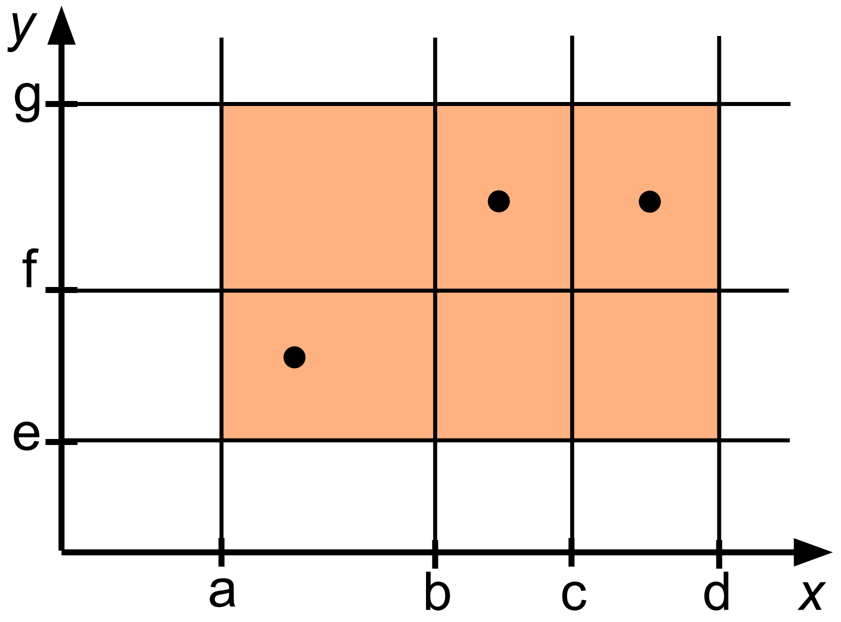
Equivalence partitioning

- detect errors to do with computational mistakes
- for integer input *x* with domain [*a*,*d*] partitioned in s_x subdomains, test input values from each subdomain
- assumes:
  - independent partitioning
  - redundancy in subdomain

weak normal variant:

- assumes:
  - – independent variables
  - – single-fault assumption
- #tests = $\mathbf{Max}_x\ s_x$

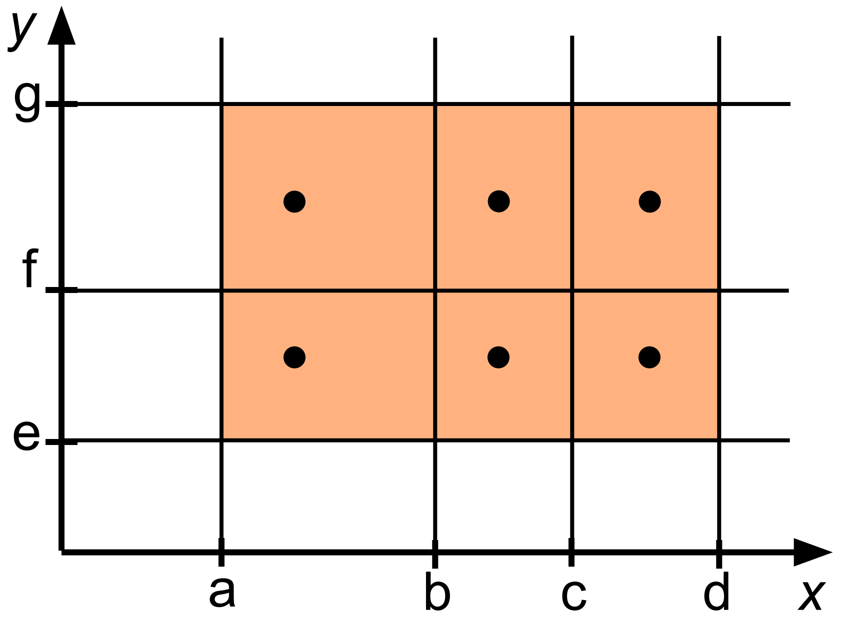
(maximal number of partions)



Equivalence partitioning

strong normal variant:

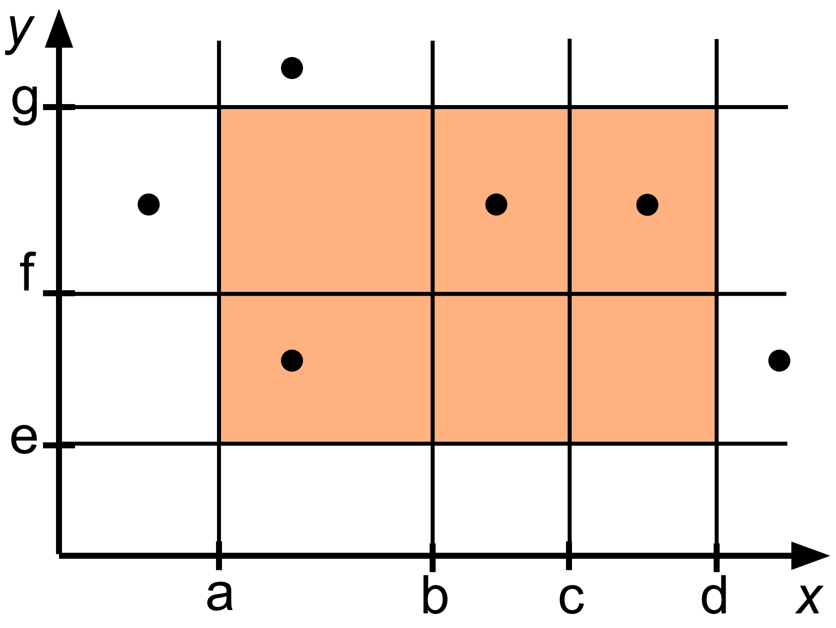
- assumes:
  - multiple-fault assumption
- #tests = $\Pi_x \, s_x$
  (product of number of partitions)



Equivalence partitioning

weak robust variant:

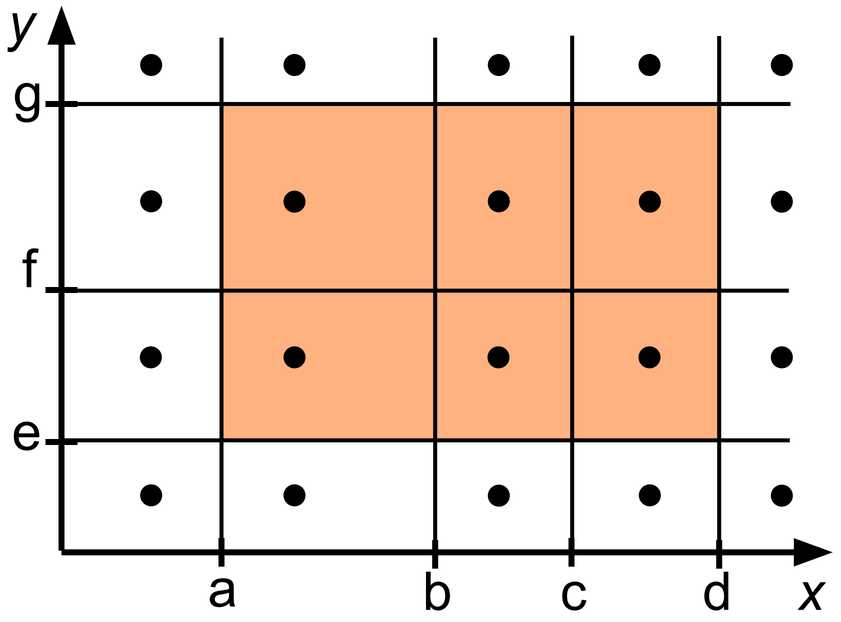
- tests outside domain
- assumes:
  - independent variables
  - single-fault assumption
- #tests = $\textbf{Max}_{\boldsymbol{x}}\, s_x + \Sigma_x\, 2$
  (weak normal +2*(#inputs))



Equivalence partitioning

strong robust variant:

- tests outside domain
- assumes:
  - multiple-fault assumption
- #tests = $\Pi_x (s_x+2)$
- product of (X+2) *(y+2)

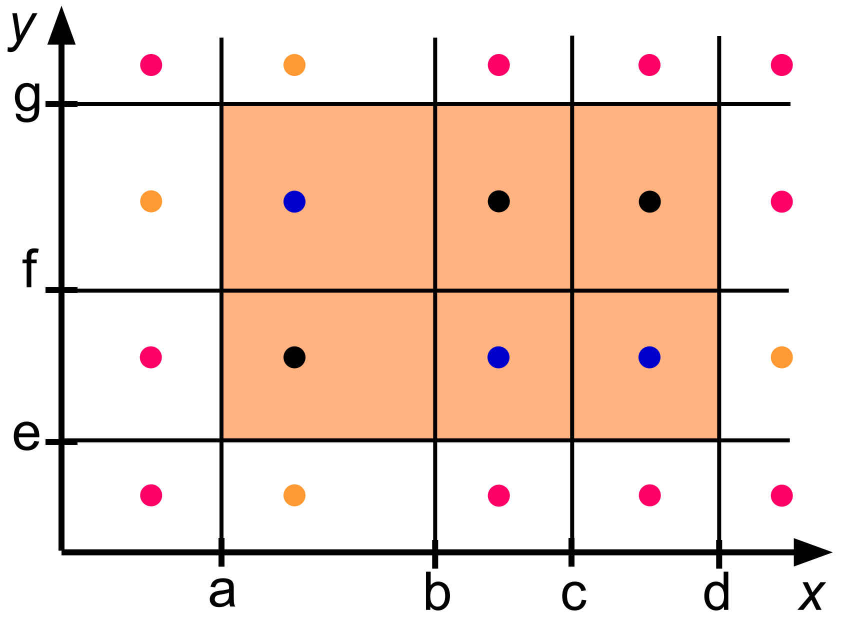


Subsume relations

Which subsume relations for equivalence partition variants?

(Assume same value selected for each input, for each subdomain)

$$\begin{tikzcd}
& \text{weak normal} & \\
\text{strong normal} \arrow[ur] & & \text{weak robust} \arrow[ul] \\
& \text{strong robust} \arrow[ul] \arrow[ur] &
\end{tikzcd}$$



Equivalence partitioning summary (1-2)

- A group forms a EC if
  - They all test the same thing
  - If one test case catches a defect, the others probably will too
  - If one test case doesn’t catch a defect, the others probably won’t either
- What makes us consider them as equivalent
  - They involve the same input variable
  - They result in similar operations in the program
  - They affect the same output variable
  - None force the program to do error handling or all of them do

Equivalence partitioning summary (2-2)

Coverage: moderate
#tests: small to moderate
Usage: some study of requirements needed
When to use:

- independent inputs
- enumerable quantities
- when suspecting computational errors
- when redundancy can be assumed
- may easily be combined with boundary value

See literature:

- Patton (chapter 5, pages 67-69)
- Jorgensen (chapter 7)
- Zhu (section 4.4)

ADVANTAGES

Equivalence partitioning is vastly superior to a random selection of test cases.
It uses the fewest test cases possible to cover the most input requirements.

DISADVANTAGES

- Although some guidelines are given to assist in the identification of equivalence classes, doing so is mostly an intuitive process.

- It overlooks certain types of high yield test cases that boundary-value analysis can determine.

Previous Date example

Previous Date problem

Steps

- Step 1 Identify both input equivalence class and output conditions by taking for input values both valid and invalid inputs
- Step 2 Generate Test cases using all equivalence classes

Solution: To first generate test cases based on input range

Then to generate test cases based on output range

Inputs:

- Valid input to get valid output
- Valid input but wrong output

Output conditions:

- Output 1 All valid outputs i.e. Previous Date
- Output 2 Invalid Date (eg 31 days in month of February)

Ie.

- Input1: 1 month 12
- Input 2: Month 1
- Input 3: Month 12
- Input 4: 1 Date 31
- Input 5: Date 1
- Input 6: Date 31
- Input 7: 1900 Year 2025
- Input 8: Year 2025
- Input 9: Year 1900

Test id	Month	Date	Year	Previous Date
001 (O/P conditions)	6	15	1962	14th June 1962
002	2 (3)	31 (1)	1962	Invalid Output
003 (I/P values)	6	15	1962	14th June 1962
004	0	15	1962	Invalid Input
005	13	15	1962	Invalid Input
006	6	15	1962	14th June 1962
007	6	-1	1962	Invalid Input
008	6	32	1962	Invalid Input
009	6	15	1962	14th June 1962
010	6	15	1989	Invalid Input
011	6	15	2026	Invalid Input

Triangle Problem

- Triangle problem

Sides lying between 1 Side 100

Output Conditions

- Output 1: Equilateral triangle (Say 50, 50, 50)
- Output 2: Isosceles triangle (Say 99, 50, 50)
- Output 3: Scalene triangle (Say 99, 50,100)
- Output 4: Not a triangle (50, 50,100)

Input Conditions

- Input 1: x 1
- Input 2: x 100
- Input 3*:* 1 x 100
- Input 4: y 1
- Input 6: y 100
- Input 7: 1 y 100
- Input 8: z 1
- Input 9: z 100
- Input 10: 1 z 100
- Input 10: x=y=z (Here we also look for existence of relationship between the variables (which was not found in previous date problem))
- Input 12: x=y , x z
- Input 13: x=z, x y
- Input 14: y=z, x
- Input 15: xy, yz
- Input 16: x=y + z,
- Input 17: y = x+ z
- Input 18: y x + z
- Input 19: z = x + y
- Input 20: z x + y

Test id	X	Y	Z	Expected Output
001	0	50	50	Input Invalid
002	101	50	50	Input Invalid
003	50	50	50	Equilateral
004	50	0	50	Invalid Input
005	50	101	50	Invalid Input
006	50	50	50	Equilateral
007	50	50	0	Input Invalid
008	50	50	101	Input Invalid
009	50	50	50	Equilateral
010	60	60	60	Equilateral
011	50	50	60	Isosceles
012	50	60	50	Isosceles
013	60	50	50	Isosceles
014	100	99	50	Scalene
015	100	50	50	Not a triangle (x= y+ z)
016	100	50	25	Not a triangle (x > y+ z)
017	50	100	50	Not a triangle Y = x + z
018	50	100	25	Not a triangle Y > x + z
019	50	50	100	Not a triangle (z = x+ Y)
020	25	50	100	Not a triangle (z > x + Y)

- Questions?