Black Box Techniques

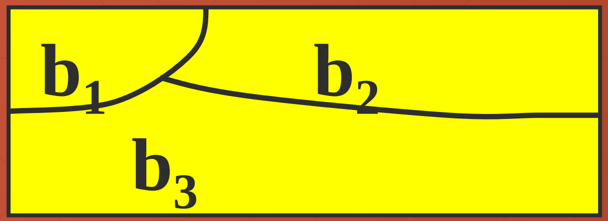
SENG 5811 – Spring 2023 – Week #5

Equivalence Partitioning

Equivalence partitioning

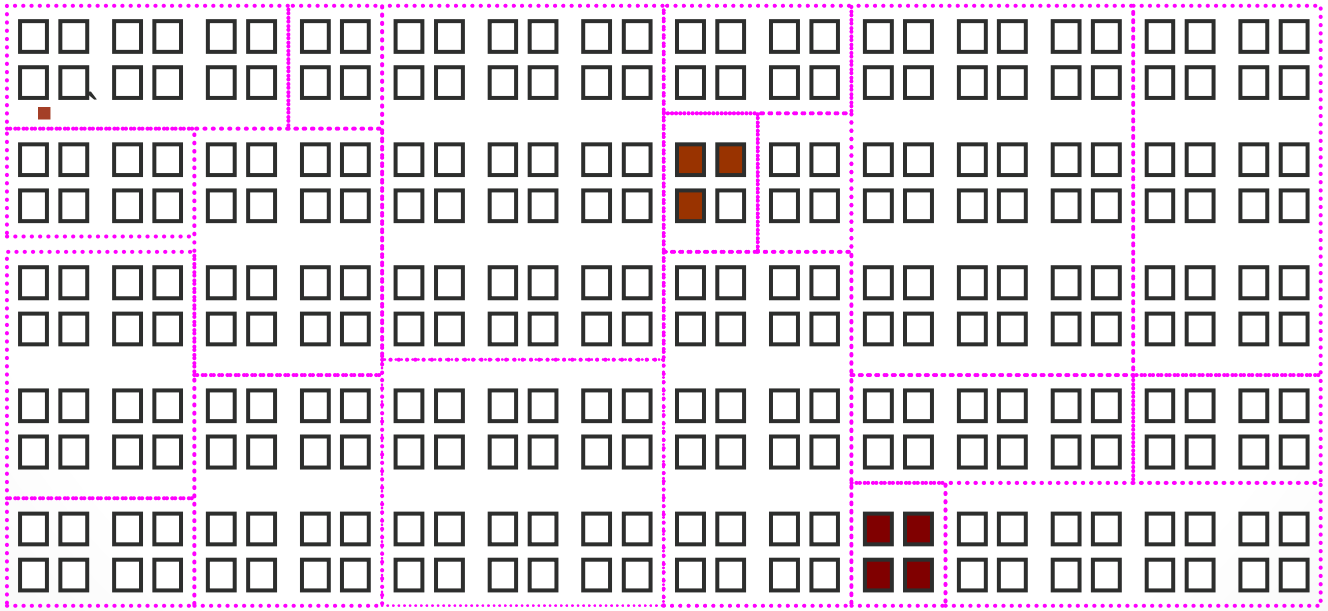
If a given set of input values all cause the program to behave in *exactly the same way*, then that's a candidate for an

equivalence class



Motivation

- **If** failures are sparse in the space of possible inputs but dense in some parts of the space,
- **then** systematically testing some cases from each part, we will include the dense parts



Determining Equivalencies

- Application to testing – **partition one input parameter at a time**
- Determine characteristics of that parameter and <u>partition</u> the input domain accordingly
  - Look for ranges of numbers or values
  - Look for memberships in groups
  - Include invalid input classes: both “junk” and outside any boundaries
  - Include internal boundaries
- **All values in a class are assumed to be equally useful for testing** (thus minimizing the number of test cases)
- Choose a value from each partition *for that one parameter*
- <u>Choose tests</u> by combining values from different parameters
- Don’t worry if a parameter’s classes overlap each other — better to be redundant than to miss something
- Equivalence Class test cases will almost surely overlap with Boundary Value test cases

Testing Based on Equivalence Partitioning

- ***Weak*** *equivalence class testing*: One data point from each valid class for each variable*
- ***Strong*** *equivalence class testing*: one data point from each valid class in the cross product of the classes
- **Robust**: Include invalid classes if it makes sense

Example

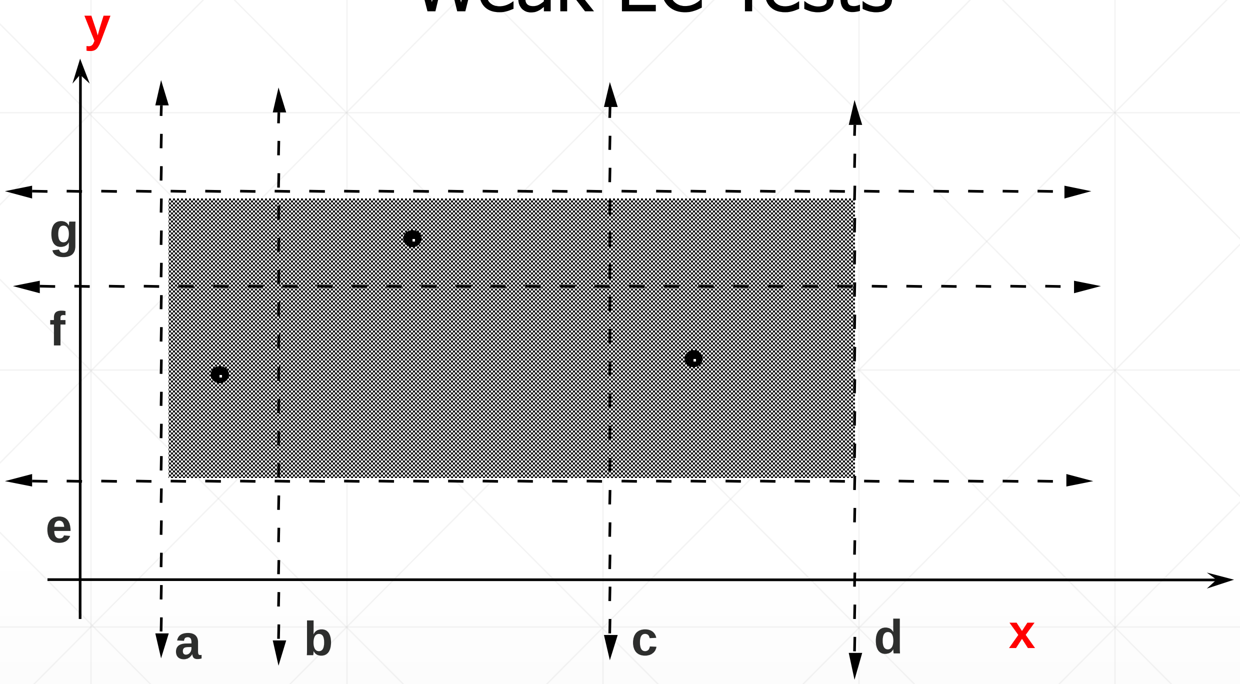
- Suppose a program has 2 input variables, x and y
- Suppose x can lie in 3 valid non-equivalent classes of a partition:

  $\mathbf{a \le x < b,\ b \le x < c,\ or\ c \le x \le d}$

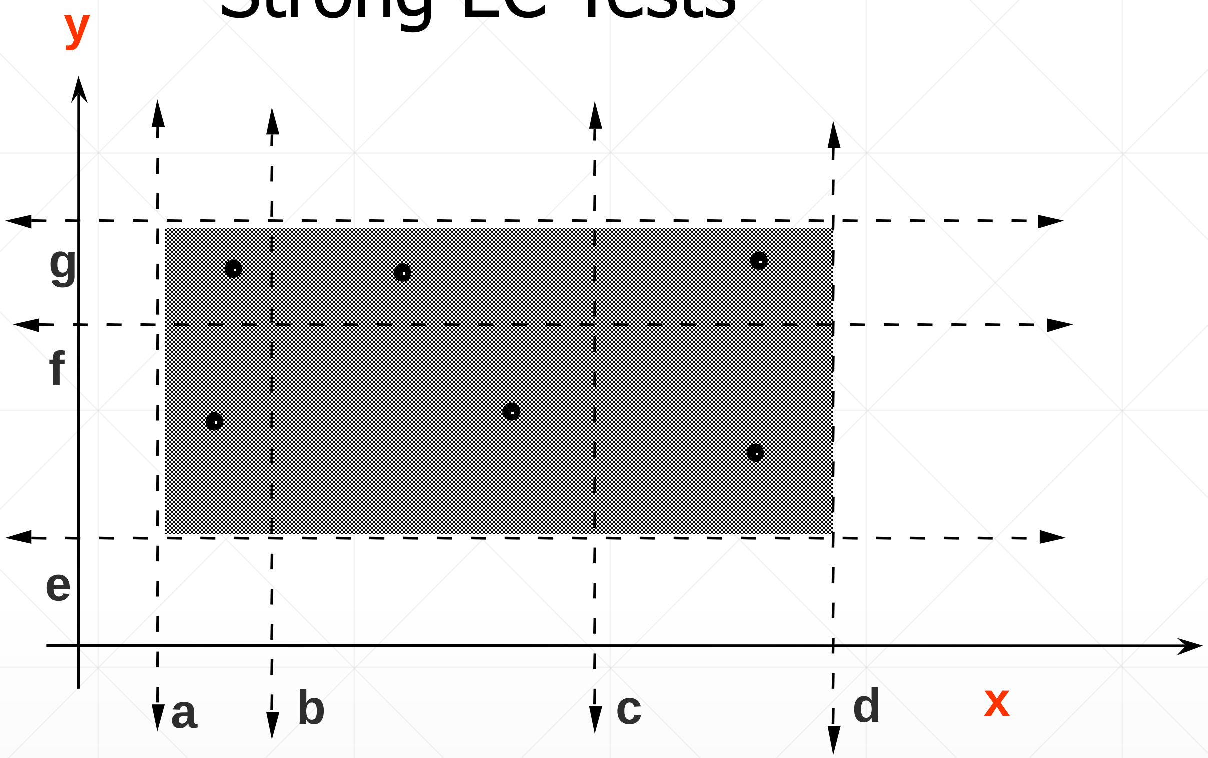
- Suppose y can lie in 2 valid non-equivalent classes of a partition:

  $\mathbf{e \le y < f\ or\ f \le y \le g}$

Weak EC Tests



Strong EC Tests



Example 1

The Triangle Problem

In order for 3 integers a, b, and c to be the sides of a triangle, we must have

c1 $a + b > c$

c2 $a + c > b$

c3 $b + c > a$

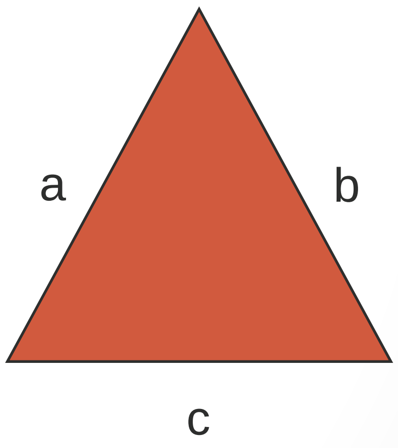
A triangle is:

Equilateral if all 3 sides are equal

Isosceles if 2 sides are equal

Scalene if no two sides are equal

We also require $1 \le a,b,c \le 200$.



- This program reads in a date in a certain format and prints out the next day’s date.
- For example, an input of 31 Mar 1998 gives an output of 01 Apr 1998.
- The year is constrained to lie between 1814 and 2014 inclusive.

Example 2

Next Date Function

Boundary Value Analysis

Main ideas

- The idea: To test values, sizes or quantities near the design limits
  - Value limits [e.g., currency, numbers in general]
  - Length limits [e.g., text strings]
  - Volume limits [e.g., networks, table size]
  - First and last elements in a table or data structure
  - Null strings, one-character strings
  - Hardware limits both large and small
- Some boundaries are “natural” and have nothing to do with design limits [e.g. dates, discount values]
- Errors tend to occur near the extreme values of inputs or at the edges of internal boundaries
- Applications, of course, but they may have *internal* boundaries too

Boundary Value Analysis

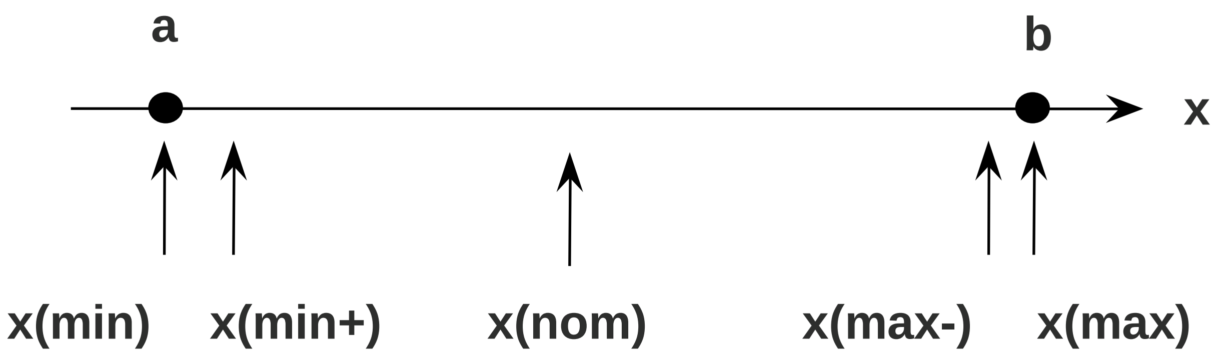
- **Normal BV testing**:
  - Never go outside the boundaries
  - When values on a GUI come from drop-down lists, e.g., this is forced.

- **Robust BV testing**:
  - Try values outside the boundaries
  - Can the software correctly handle values that are outside the designed limits?

- **Best Practices** says to use input variable values at
  - their **minimum**
  - just **above** the minimum
  - at a **nominal** value
  - just **below** the maximum
  - and at the **maximum**

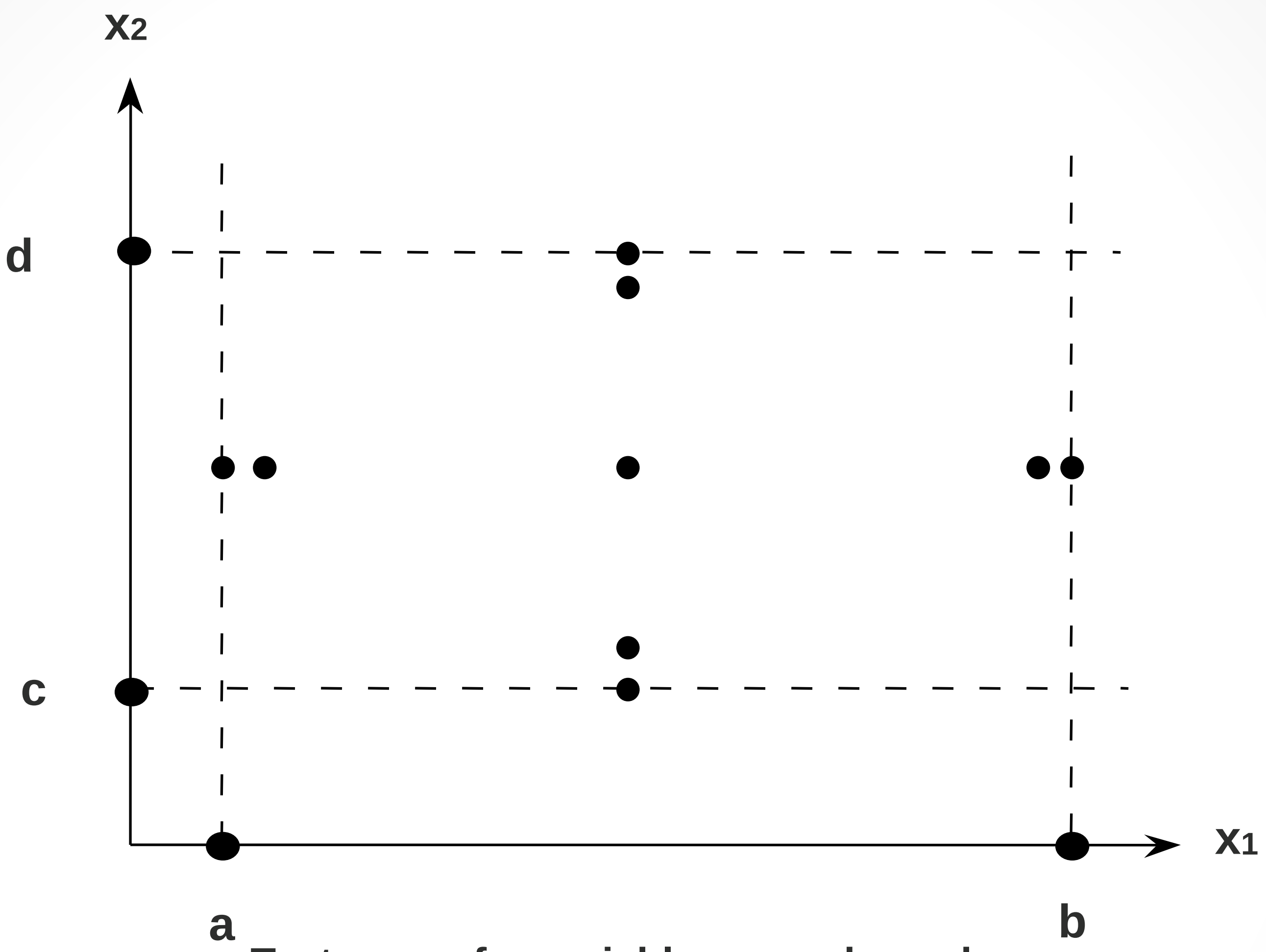
Normal Boundary Value Testing

One variable



Test cases for a single variable x, where a ≤ x ≤ b

Experience shows that errors occur more frequently for extreme values of a variable.



Test cases for variables x_1 and x_2, where $a \leq x_1 \leq b$ and $c \leq x_2 \leq d$

Normal Boundary Value Testing

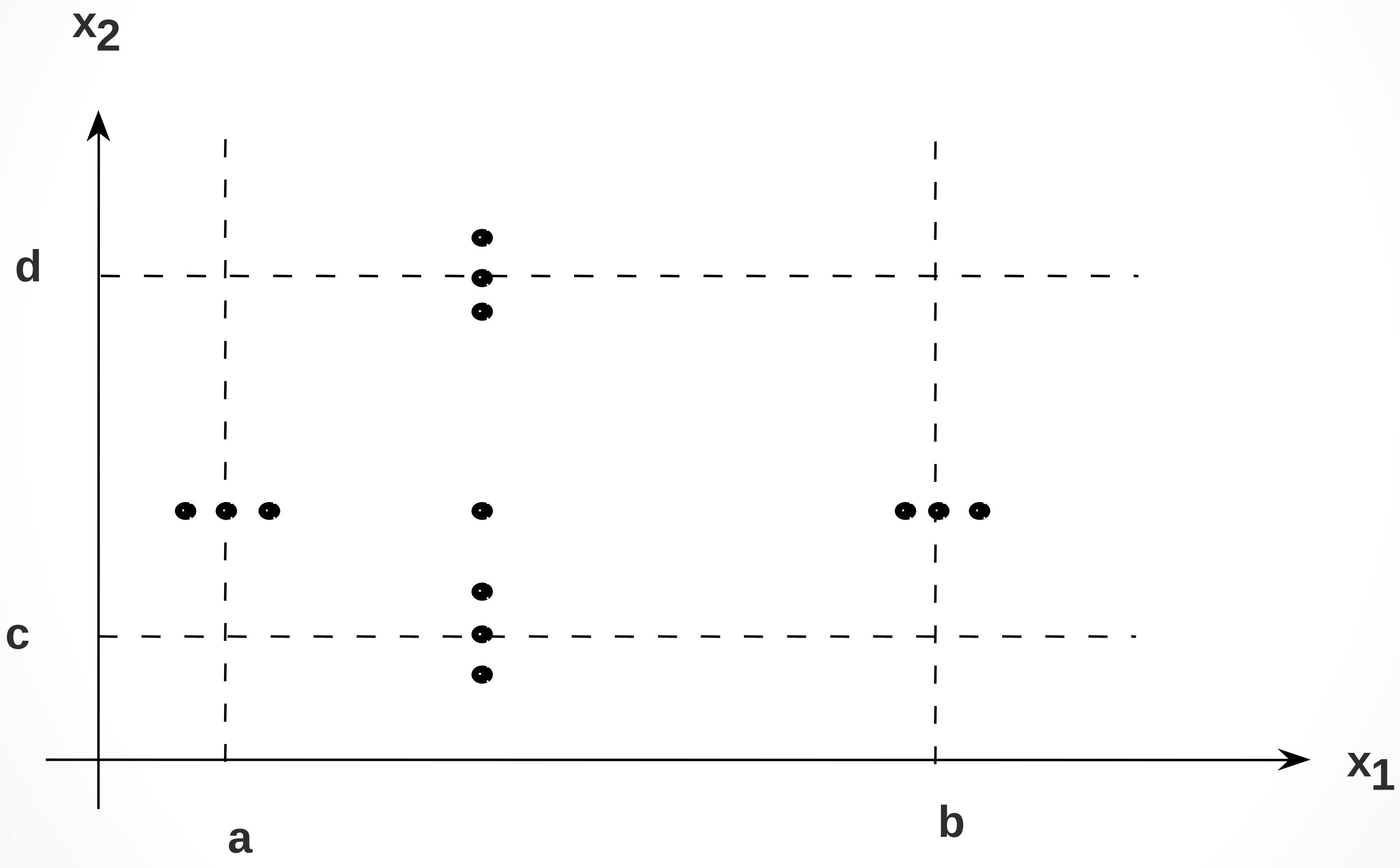
2 Variables

Robustness BV Testing

One Variable

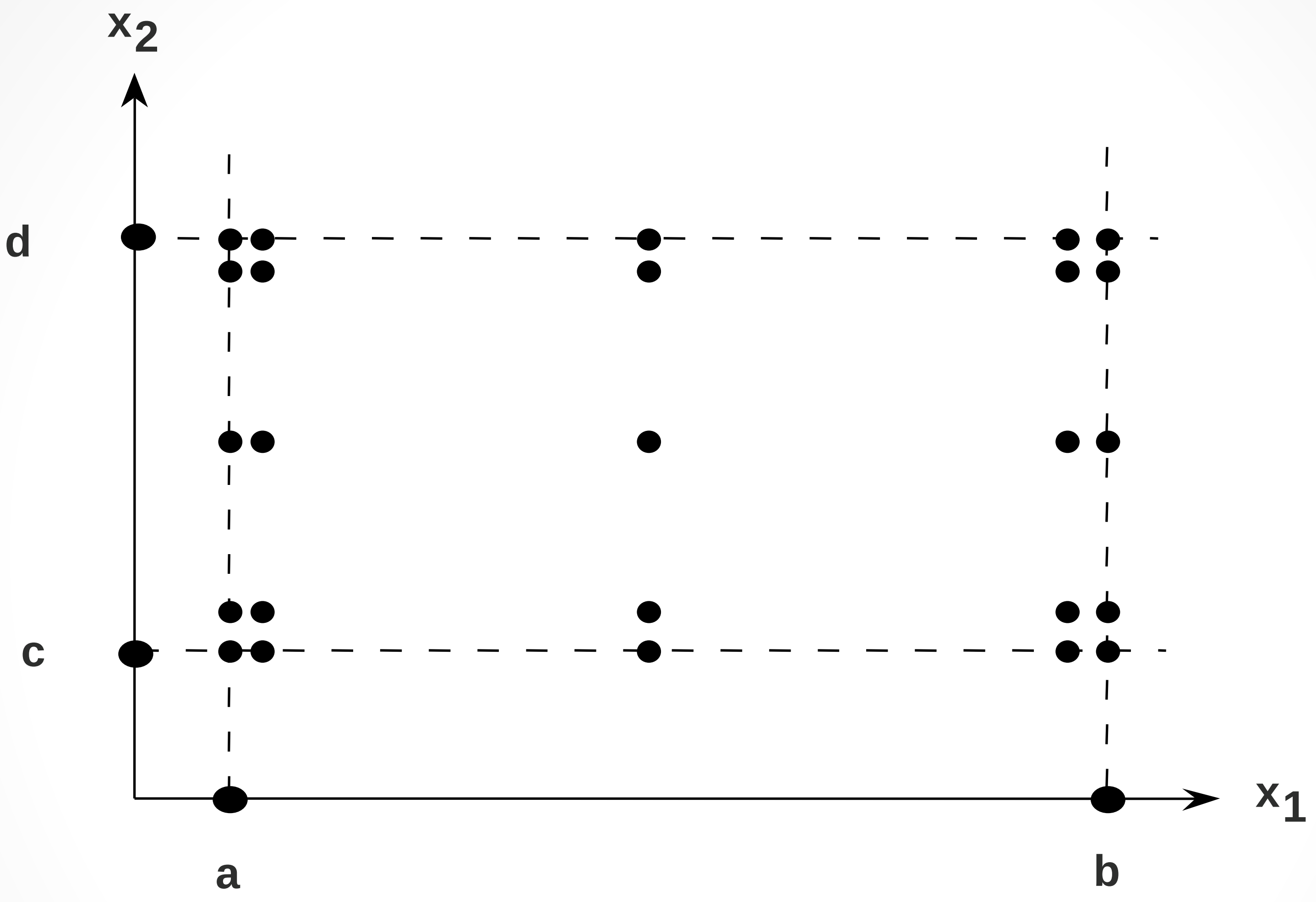
Test cases for a variable x, where $a \leq x \leq b$

1. **Stress input boundaries**
2. **Verify acceptable response for invalid inputs**



Normal BV Testing

Two Variables



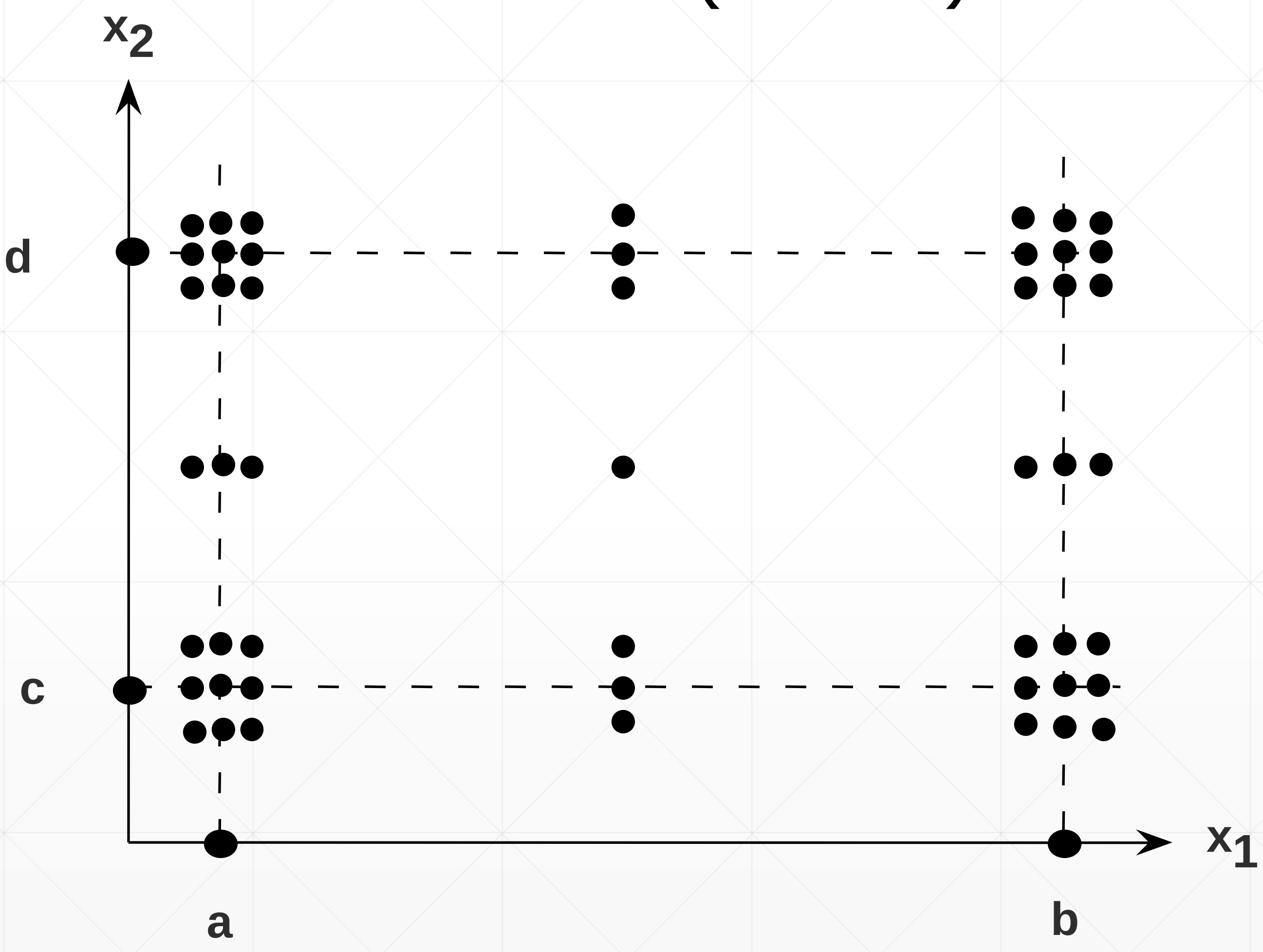
Eliminate the "single fault" assumption.

Murphy's Law

Normal BV Testing

2 Variables

Robustness (Paranoid) BV Testing with 2 Variables (no SFA)

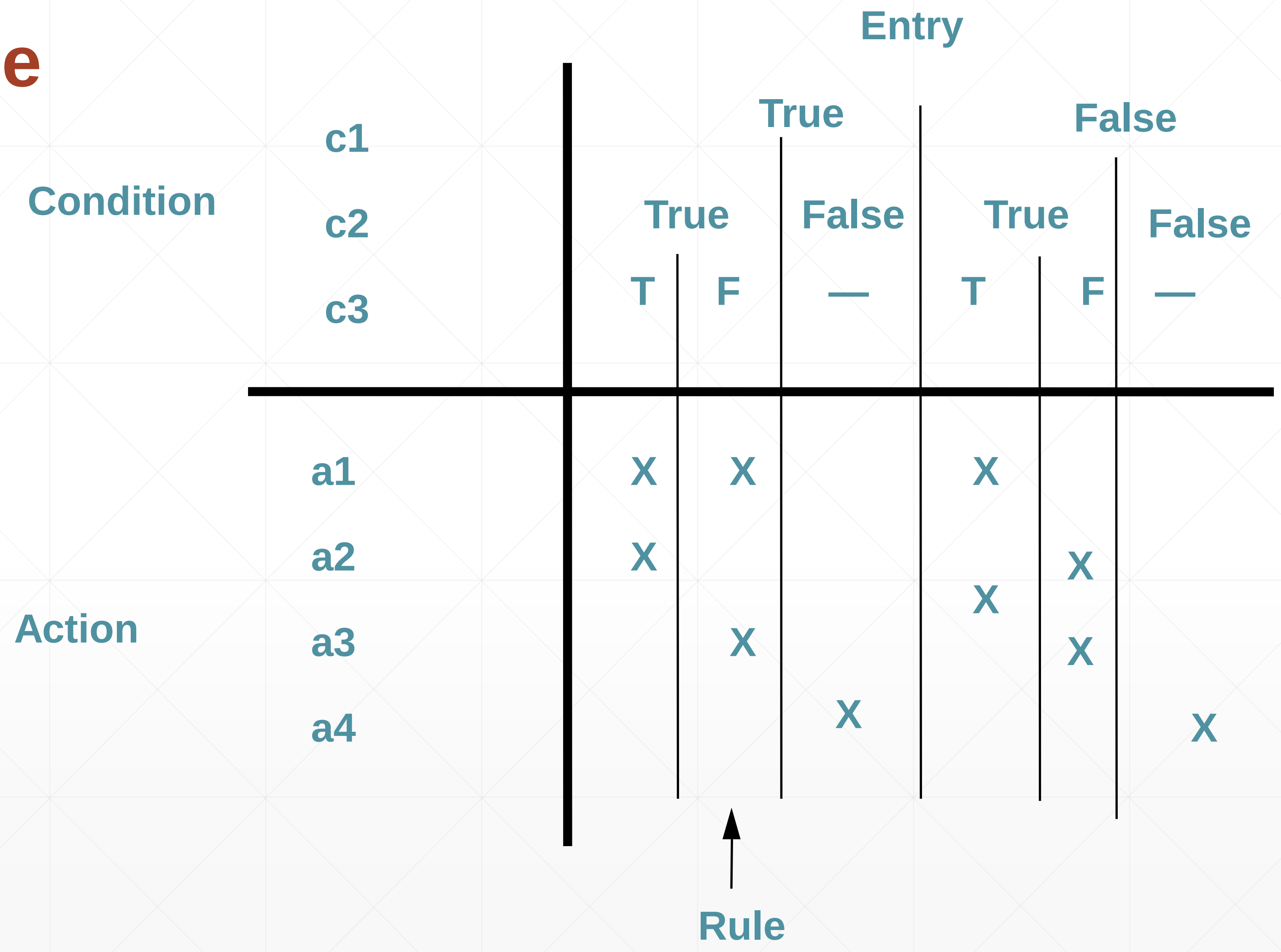


Decision Tables

Format of a Decision Table

- Inputs are interpreted as **binary** *conditions* (Text: *inputs* or *causes*)
- Outputs are interpreted as *actions* (Text: *effects*)
- The columns in a table are *rules* — they show which actions result from which conditions
- Every rule then becomes a Logical Test Case
- Dashes represent *don't care* conditions

Decision Table Terminology



Decision Table-Based Testing

- Rigorous functional method — it forces you to think of all the possible combinations of input conditions, and of the actions or effects of complicated logical relationships
- Decision Tables support consistency and completeness
- Dependencies can yield impossible combinations, so we usually have an “impossible” action

One Decision Table for the Triangle Program

c1: a, b, c are a triangle?	F	T							
c2: a = b?	—	T				F			
c3: a = c?	—	T		F		T		F	
c4: b = c?	—	T	F	T	F	T	F	T	F
a1: not a triangle	X								
a2: Scalene									X
a3: Isosceles					X		X	X	
a4: Equilateral		X							
a5: Impossible			X	X		X			

A Second Formulation

c1: a < b+c?	F	T	T	T	T	T	T	T	T	T	T
c2: b < a+c?	--	F	T	T	T	T	T	T	T	T	T
c3: c < a+b?	--	--	F	T	T	T	T	T	T	T	T
c4: a = b?	--	--	--	T	T	T	T	F	F	F	F
c5: a = c?	--	--	--	T	T	F	F	T	T	F	F
c6: b = c?	--	--	--	T	F	T	F	T	F	T	F
a1: Not a triangle	X	X	X								
a2: Scalene											X
a3: Isosceles							X		X	X	
a4: Equilateral				X							
a5: Impossible					X	X		X			

A Redundant Decision Table

Conditions	1-4	5	6	7	8	9
c1:	T	F	F	F	F	T
c2:	--	T	T	F	F	F
c3:	--	T	F	T	F	F
a1:	X	X	X	--	--	X
a2:	--	X	X	X	--	--
a3:	X	--	X	X	X	X

- Rule 9 is identical to Rule 4 (T, F, F)
- Since the action entries for rules 4 and 9 are identical, there is no ambiguity, just redundancy.

An Inconsistent Decision Table

Conditions	1-4	5	6	7	8	9
c1:	T	F	F	F	F	T
c2:	--	T	T	F	F	F
c3:	--	T	F	T	F	F
a1:	X	X	X	--	--	--
a2:	--	X	X	X	--	X
a3:	X	--	X	X	X	--

- Rule 9 is identical to Rule 4 (T, F, F)
- Since the action entries for rules 4 and 9 are different there is ambiguity.
- This table is inconsistent, and the inconsistency implies non-determinism — can't tell which rule to apply!

Procedure for Decision-Table Based Testing

1. Determine the conditions and actions.
2. Develop the Decision Table, watching for
   - completeness
   - don’t care entries
   - redundant and inconsistent rules
3. Create at least one test case for each rule (column)

Next Week

- Continue Black-Box Testing
  - Decision Tables
  - State-machine Based
- Start White-Box Testing
- Reading
  - Rest of Chapter 4 in [GBV] – Purple book
  - Chapter 7 in [AO] – Red book