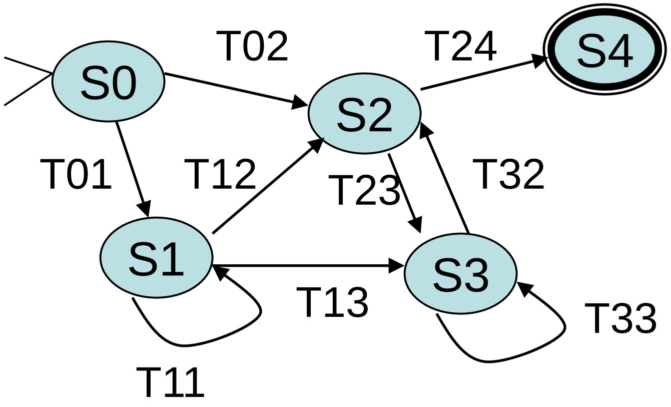
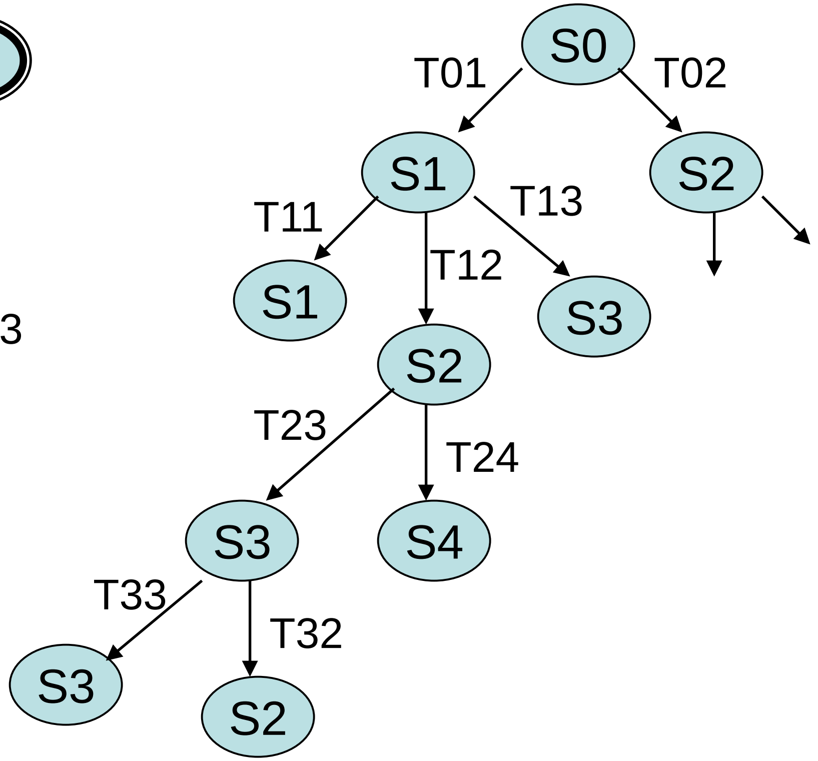
State-Based Testing





Test cases:

Level #1: S0->T01->S1
S0->T02>S2
Level #2: S0->T01->S1->T11->S1
S0->T01->S1->T12->S2
S0->T01->S1->T13->S3
…….

Black-Box Software Testing (Part I)

Speaker: Jerry Gao Ph.D.

Computer Engineering Department
San Jose State University

email: jerry.gao@sjsu.edu
URL: http://www.engr.sjsu.edu/gaojerry

Presentation Outline

- Introduction to Black Box Software Testing?
    - Definition
    - Why Black Box Testing?
    - Testing Objectives and Focuses
- An Example
- Graph-based Testing Methods
- Equivalence Partitioning
- Boundary Value Analysis

Introduction to Black Box Testing

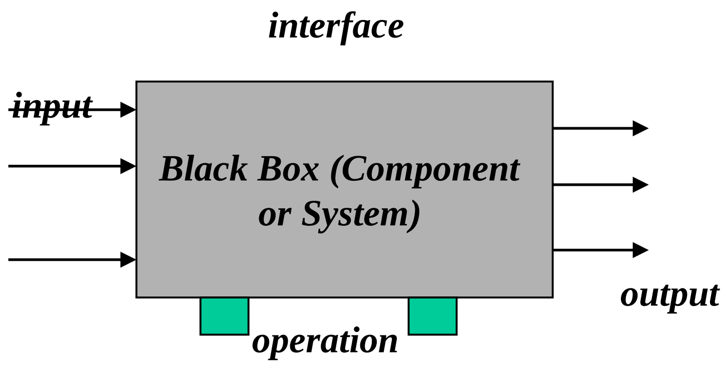
What is black box testing?

- Black box testing also known as specification-based testing.

- Black box testing refer to test activities using specification-based testing methods and criteria to discover program errors based on program requirements and product specifications.

The major testing focuses:

- *specification-based function errors*
- *specification-based component/system behavior errors*
- *specification-based performance errors*
- *user-oriented usage errors*
- *black box interface errors*



Introduction to Black Box Testing

Under test units in black-box: ***Software components, subsystems, or systems***

What do you need?

- For software components, you need component specification, user interface doc.

-For a software subsystem or system, you need requirements specification, and product specification document.

You also need:

- Specification-based software testing methods

- Specification-based software testing criteria

- good understanding of software components (or system)

An Example

Testing a triangle analyzer:

Program specification:

Input: *3 numbers separated by commas or spaces*

Processing:

Determine if three numbers make a valid triangle; if not, print message NOT A TRIANGLE.

If it is a triangle, classify it according to the length of the sides as scalene (no sides equal), isosceles (two sides equal), or equilateral (all sides equal).

If it is a triangle, classify it according to the largest angle as acute (less than 90 degree), obtuse (greater than 90 degree), or right (exactly 90 degree).

Output: *One line listing the three numbers provided as input and the classification or the not a triangle message.*

Example:	*3,4,5*	*Scalene*	*Right*
	6,1,6	*Isosceles*	*Acute*
	5,1,2	*Not a triangle*	

An Example

Functional Test Cases:

	Acute	*Obtuse*	*Right*
Scalene:	*6,5,3*	*5,6,10*	*3,4,5*
Isosceles:	*6,1,6*	*7,4,4*	*1,2, 2^(0.5)*
Equilateral:	*4,4,4*	*Not possible*	*Not possible*

Functional Test Cases:

Input	*Expected Results*
4,4,4	*Equilateral acute*
1,2,8	*Not a triangle*
6,5,3	*Scalene acute*
5,6,10	*Scalene obtuse*
3,4,5	*Scalene right*
6,1,6	*Isosceles acute*
7,4,4	*Isosceles obtuse*
1,1,2^(0.5)	*Isosceles right*

An Example

Test cases for special inputs and invalid formats:

3,4,5,6	***Four sides***
646	***Three-digit single number***
3,,4,5	***Two commas***
3 4,5	***Missing comma***
3.14.6,4,5	***Two decimal points***
4,6	***Two sides***
5,5,A	***Character as a side***
6,-4,6	***Negative number as a side***
-3,-3,-3	***All negative numbers***
	Empty input

An Example

Boundary Test Cases:

(1) Boundary conditions for legitimate triangles

1,1,2	***Makes a straight line, not a triangle***	
0,0,0	***Makes a point, not a triangle***	
4,0,3	***A zero side, not a triangle***	
1,2,3.00001	***Close to a triangle but still not a triangle***	
9170,9168,3	***Very small angle***	***Scalene, acute***
.0001,.0001,.0001	***Very small triangle***	***Equilateral, acute***
83127168,74326166,96652988	***Very large triangle, scalene, obtuse***	

Boundary conditions for sides classification:

3.0000001,3,3	***Very close to equilateral,***	***Isosceles, acute***
2.999999,4,5	***Very close to isosceles***	***Scalene, acute***

Boundary conditions for angles classification:

3,4,5.000000001	***Near right triangle***	***Scalene, obtuse***
1,1,1.41141414141414	***Near right triangle***	***Isosceles, acute***

Software Testing Principles

Davids [DAV95] suggests a set of testing principles:

- All tests should be traceable to customer requirements.

- Tests should be planned long before testing begins.

- The Pareto principle applies to software testing.
- 80% of all errors uncovered during testing will likely be traceable to 20% of all program modules.

- Testing should begin "in the small" and progress toward testing "in the large".

- Exhaustive testing is not possible.

- To be most effective, testing should be conducted by an independent third party.

Software Testability

According to James Bach:

Software testability is simply how easily a computer program can be tested.

A set of program characteristics that lead to testable software:

- Operability: "the better it works, the more efficiently it can be tested."
- Observability: "What you see is what you test."

- Controllability: "The better we can control the software, the more the testing can be automated and optimized."

- Decomposability: "By controlling the scope of testing, we can more quickly isolate problems and perform smarter retesting."

- Simplicity: "The less there is to test, the more quickly we can test it."
- Stability: "The fewer the changes, the fewer the disruptions to testing."
- Understandability:"The more information we have, the smarter we will test."

Equivalence Partitioning

Equivalence partitioning ***is a black-box testing method***

- divide the input domain of a program into classes of data

- derive test cases based on these partitions.

Test case design for equivalence partitioning is based on an evaluation of equivalence classes for an input domain.

An equivalence class represents a set of valid or invalid states for input condition.

An input condition is:

- a specific numeric value, a range of values

- a set of related values, or a Boolean condition

Equivalence Partitioning

Equivalence partitioning ***is a black-box testing method***

- divide the input domain of a program into classes of data

- derive test cases based on these partitions.

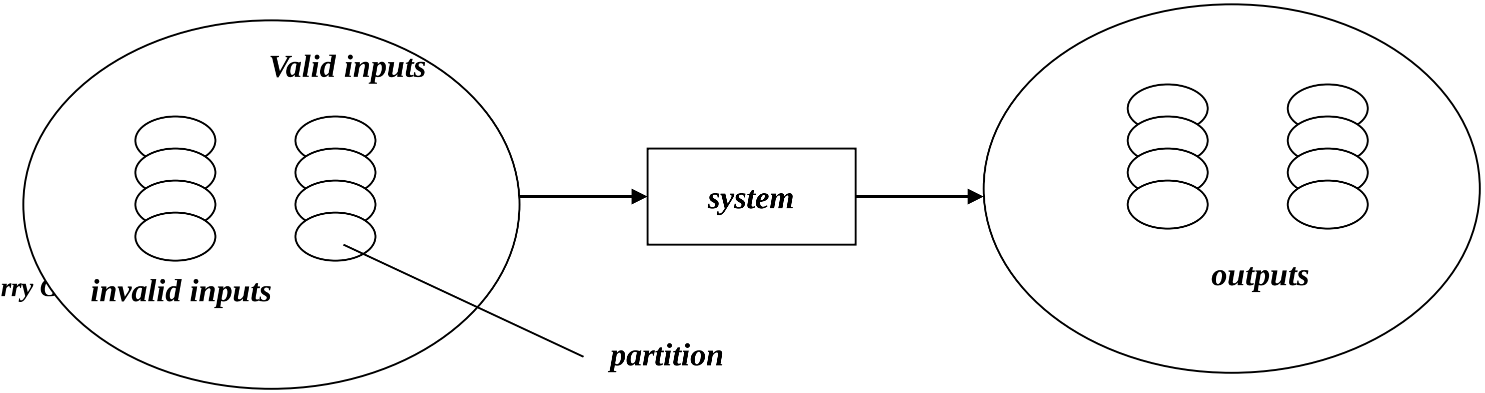
Test case design for equivalence partitioning is based on an evaluation of equivalence classes for an input domain.

An equivalence class represents a set of valid or invalid states for input condition.

An input condition is:

- a specific numeric value, a range of values

- a set of related values, or a Boolean condition



Equivalence Classes

Equivalence classes can be defined using the following guidelines:

- If an input condition specifies a range, one valid and two invalid equivalence class are defined.

- If an input condition requires a specific value, one valid and two invalid equivalence classes are defined.

- If an input condition specifies a member of a set, one valid and one invalid equivalence classes are defined.

- If an input condition is Boolean, one valid and one invalid classes are defined.

Examples:

area code: input condition, Boolean - the area code may or may not be present.
input condition, range - value defined between 200 and 900

password: input condition, Boolean - a password nay or may not be present.
input condition, value - six character string.

command: input condition, set - containing commands noted before.

Boundary Value Analysis

Boundary value analysis(BVA) ***- a test case design technique***
- complements to equivalence partition

Objective:
Boundary value analysis leads to a selection of test cases that exercise bounding values.

Guidelines:
- If an input condition specifies a range bounded by values a and b,
test cases should be designed with value a and b, just above and below a and b.

Example: Integer D with input condition [-3, 10],
test values: -3, 10, 11, -2, 0

- If an input condition specifies a number values, test cases should be developed to exercise the minimum and maximum numbers. Values just above and below minimum and maximum are also tested.

Example: Enumerate data E with input condition: {3, 5, 100, 102}
test values: 3, 102, -1, 200, 5

Boundary Value Analysis

- Guidelines 1 and 2 are applied to output condition.

- If internal program data structures have prescribed boundaries, be certain to design a test case to exercise the data structure at its boundary

Such as data structures:

- array

input condition:
empty, single element, full element, out-of-boundary

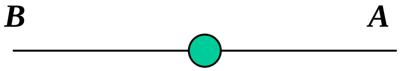
search for element:
- element is inside array or the element is not inside array

You can think about other data structures:

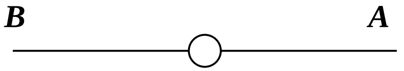
- list, set, stack, queue, and tree

One-Dimensional Domain Bugs in Open Boundaries

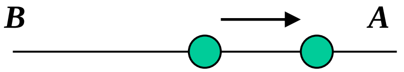
An Open Domain (A):



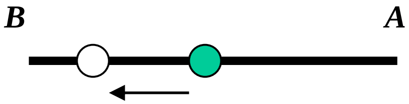
Closure Bug:



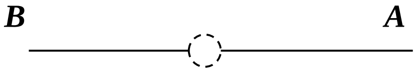
Boundary Shifted Right:



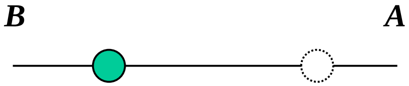
Boundary Shifted Left:



Missing Boundary:



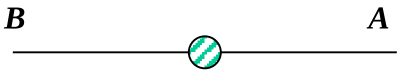
Extra Boundary:



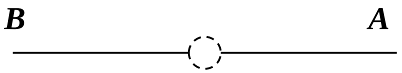
If the domain boundary is open, an off point is a point near the boundary but in the domain being tested.

One-Dimensional Domain Bugs in Closed Boundaries

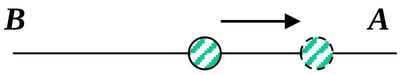
A Closed Domain (A):



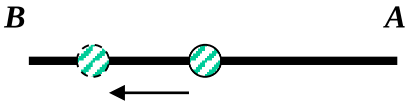
Closure Bug:



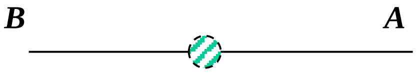
Boundary Shifted Right:



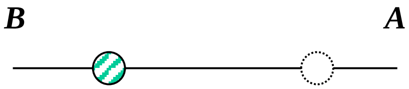
Boundary Shifted Left:



Missing Boundary:

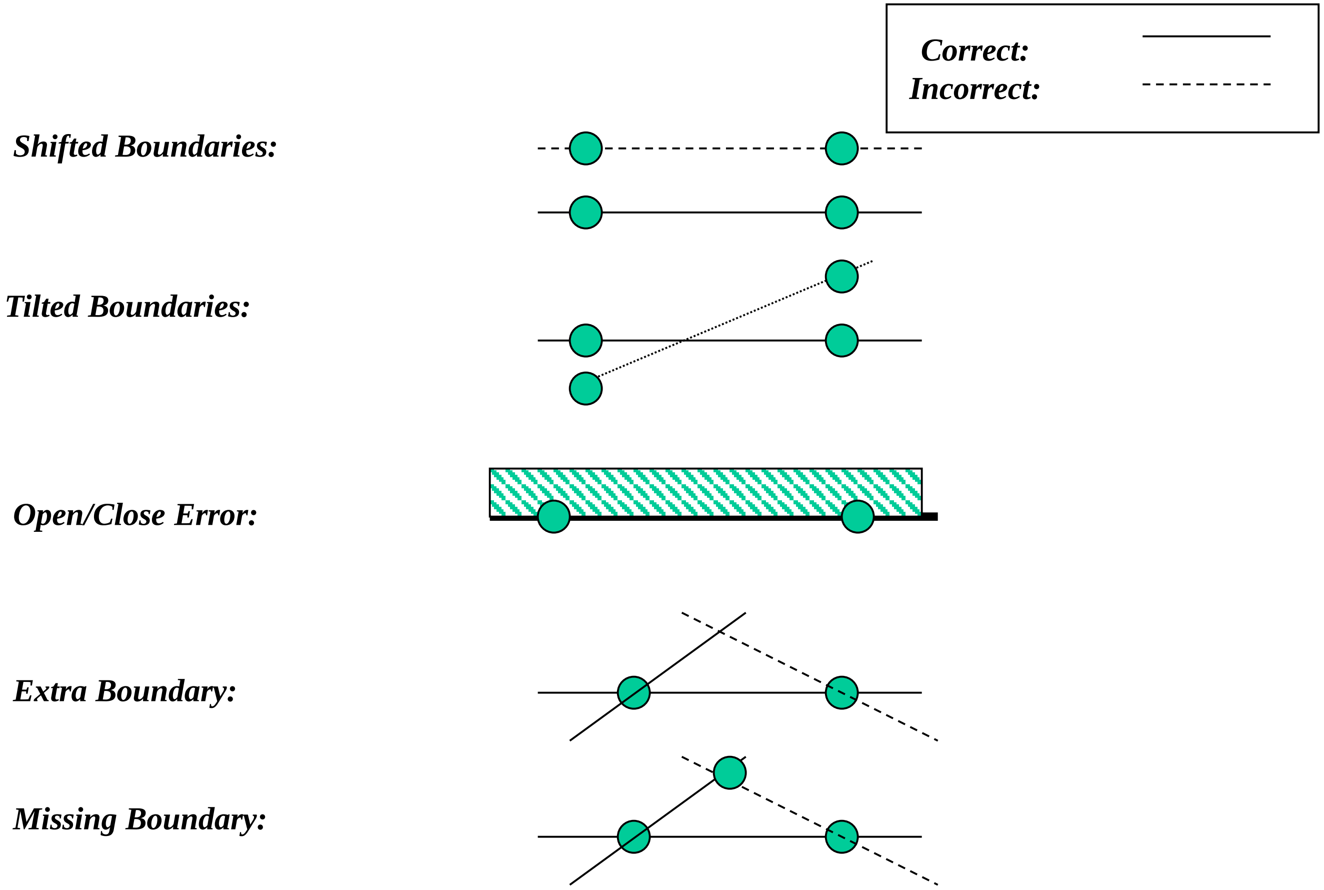


Extra Boundary:



If the domain boundary is closed, an off point is a point near the boundary but in the adjacent domain.

Generic Domain Bugs in One-Dimensional Domains



Correct: ———

Incorrect: - - - - -

Shifted Boundaries:

Tilted Boundaries:

Open/Close Error:

Extra Boundary:

Missing Boundary:

An Example

Given a function module which implements function Z= F(X, Y), which defined as follows:

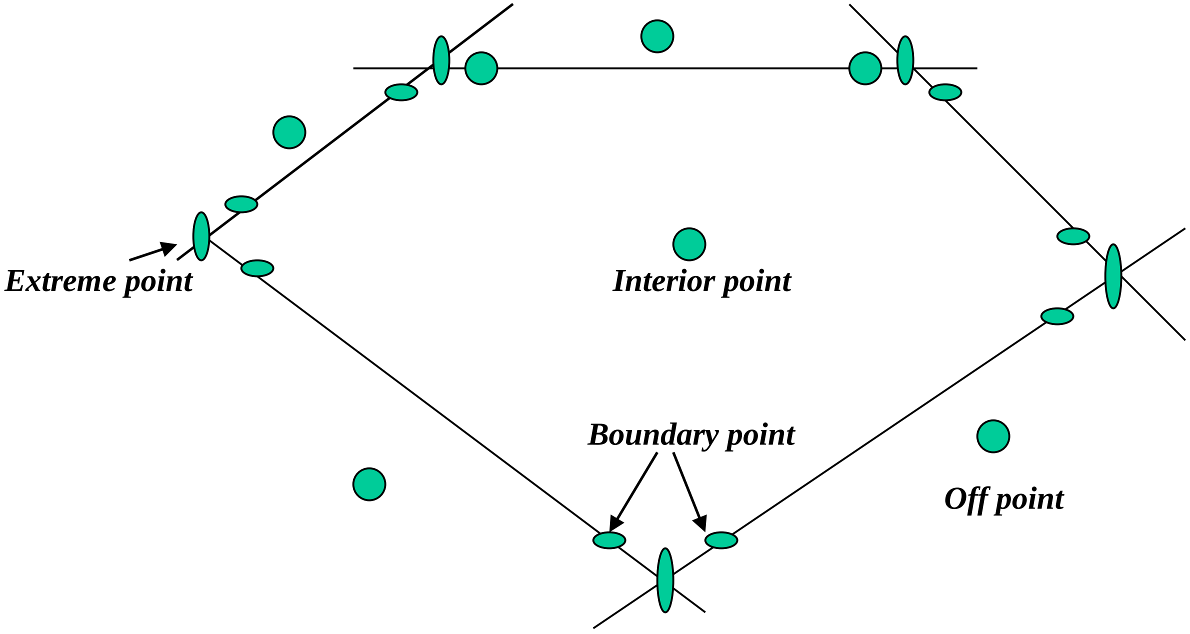
Z = F(X,Y), where X and Y are integer parameters for F. The detailed definition is given below.

$$Z = \begin{cases} X + Y & \text{when } 10<=X<=20, \text{ and } 12<=Y<=30 \\ X - Y & \text{when } 0<=X<10, \text{ and } 0<=Y<12 \\ 0 & \text{under other conditions} \end{cases}$$

Please answer the following questions:

- (4%) Identify the equivalence classes in [X, Y]. (hints: Considering Z as a function F with X and Y input variables.)
- (4%) List your test cases in [X, Y] based on the equivalence classes.
- (4%) Perform boundary value analysis, and list all boundary conditions for X and Y.
- (3%) List your test cases in [X, Y] based on boundary value analysis.

Domain Boundary



Two-dimension Domain Boundary

Testing Two-Dimensional Domains

- Closure bug.

For example, using a wrong operator (for example, x >= k when x > k is intended or vise versa). This bug could be detected due to the testing of different boundaries or trying interior and off points.

- Shifted boundary:

For example, a boundary is shifted due to the use of an incorrect constant in a predicate, such as x+y>=17 when x + y >=7 was intended. The off point catches this bug.

- Titled boundary:

A tilted boundary occurs when coefficients in the boundary inequality are wrong. For example, 3x+7y > 17 when 7x + 3y > 17 was intended. Testing different domain points can detect the bug.

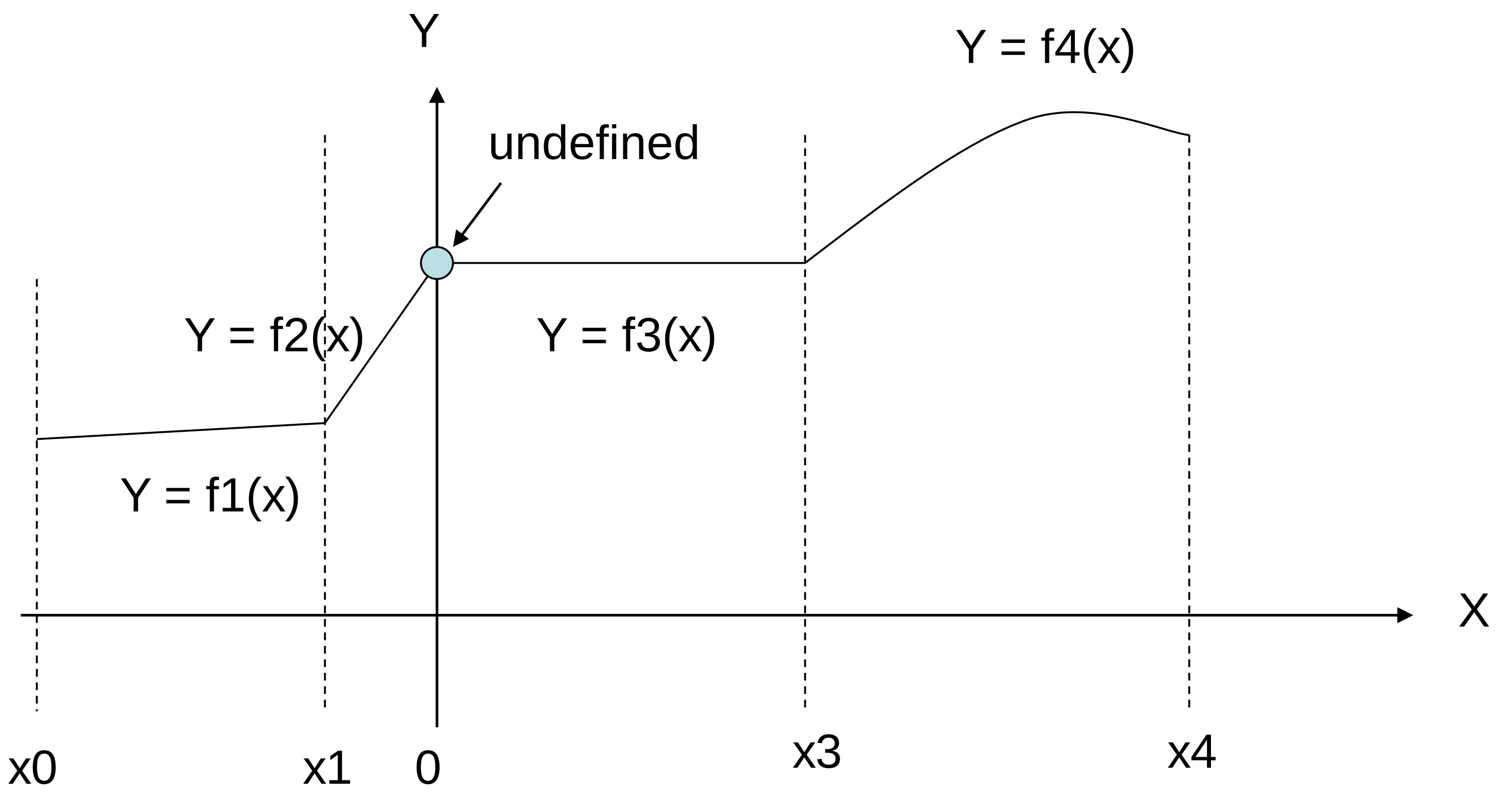
- Extra boundary:

An extra boundary is created by an extra predicate. Try different boundary points can detect this bug.

- Missing boundary:

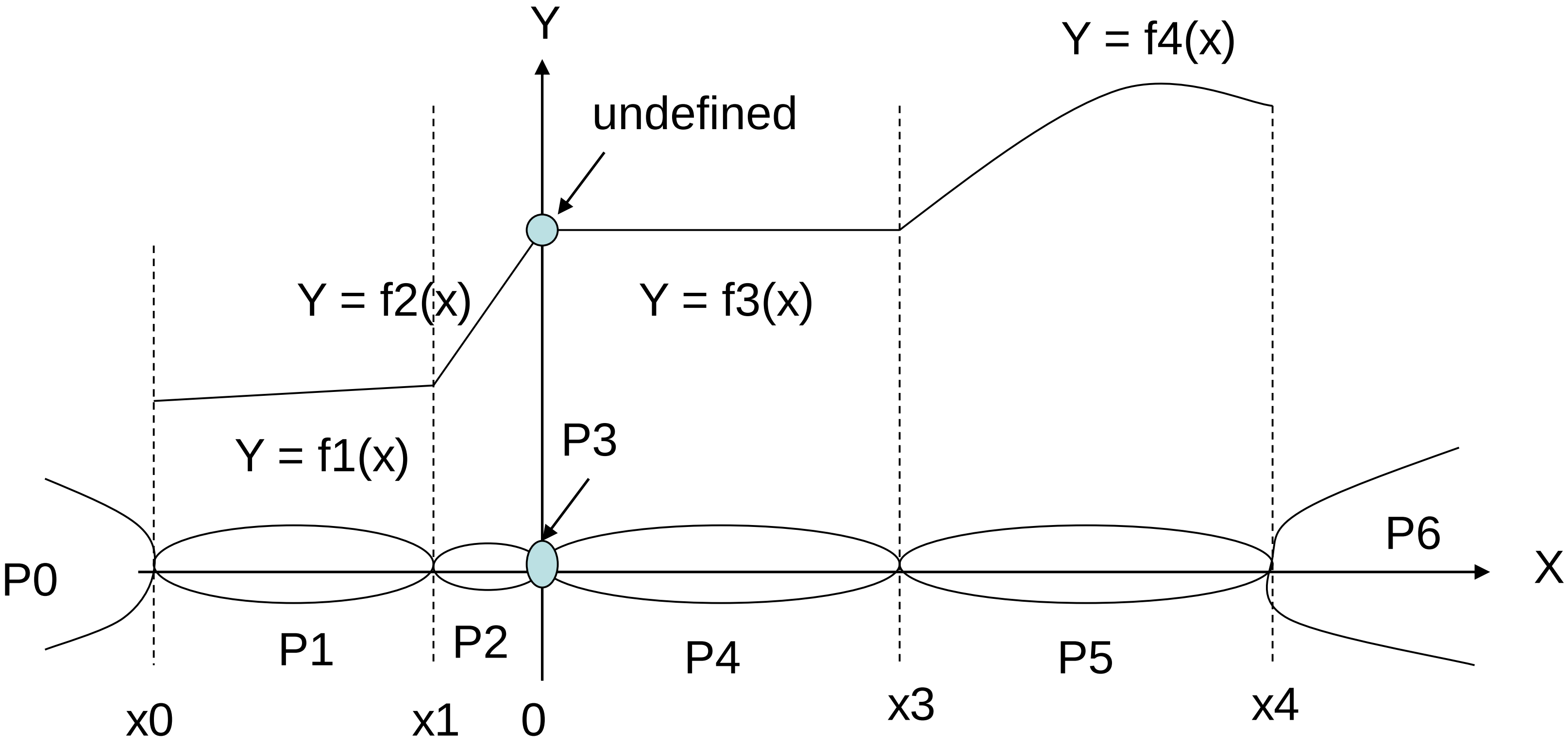
A missing boundary is created by leaving a boundary predicate out.

Equivalence Partition Testing Example



$$
Y = \begin{cases}
\text{f1(x)} & \text{x in [x0, x1]} \\
\text{f2(x)} & \text{x in (x1, 0)} \\
\text{Undefined} & \text{x = 0} \\
\text{f3(x)} & \text{x in (0, x3]} \\
\text{f4(x)} & \text{x in (x3, x4]}
\end{cases}
$$

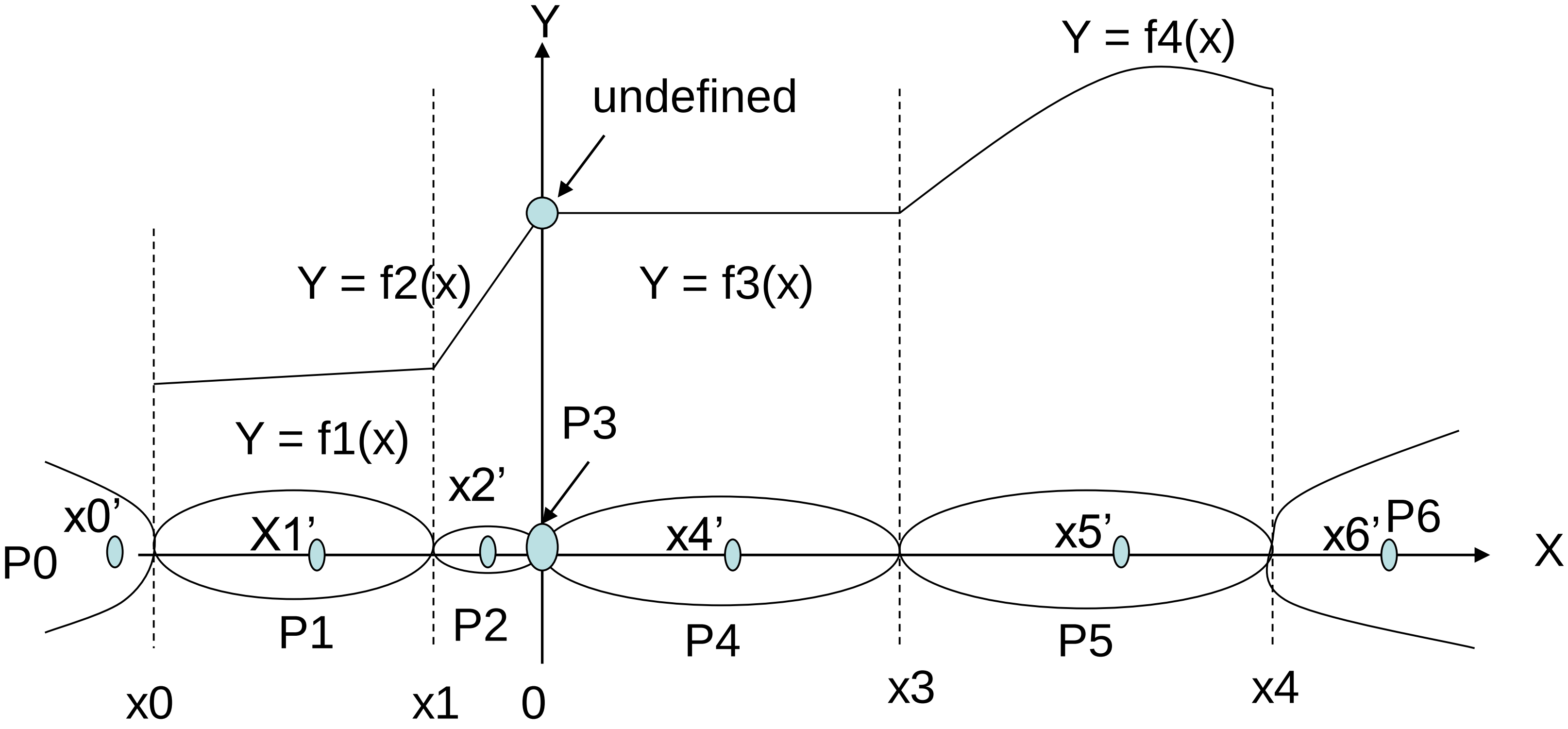
Equivalence Partition Testing Example



EQ Partitions:

P0: x < x0 or x in (x0, Very Small No.)
P1: x in [x0, x1]
P2: x in (x1, 0)
P3: x = 0
P4: x in (0, x3]
P5: x in (x3, x4]
P6: x > x4 or x in (x4, Very Larger No.)

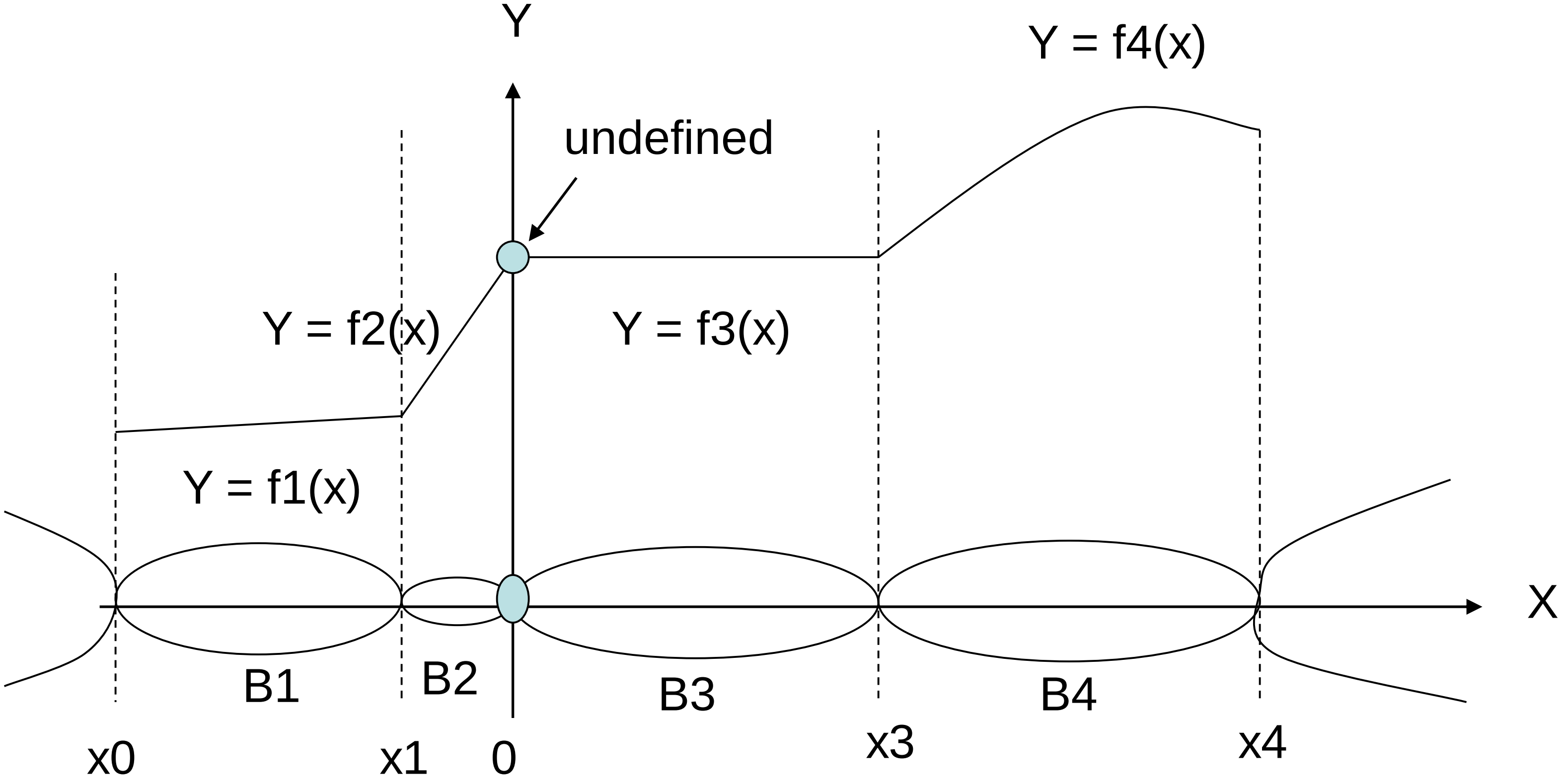
Equivalence Partition Testing Example



Test Cases for EQ Partitions:

x= x1',	y = f1(x1') = y1'
x = x2',	y = f2(x2') = y2'
x = 0,	y = 0
x = x4',	y = f3(x4') = y3'
x = x5',	y = f4(x5') = y4'
x = x0',	y = out of boundary
x = x6',	y = out of boundary

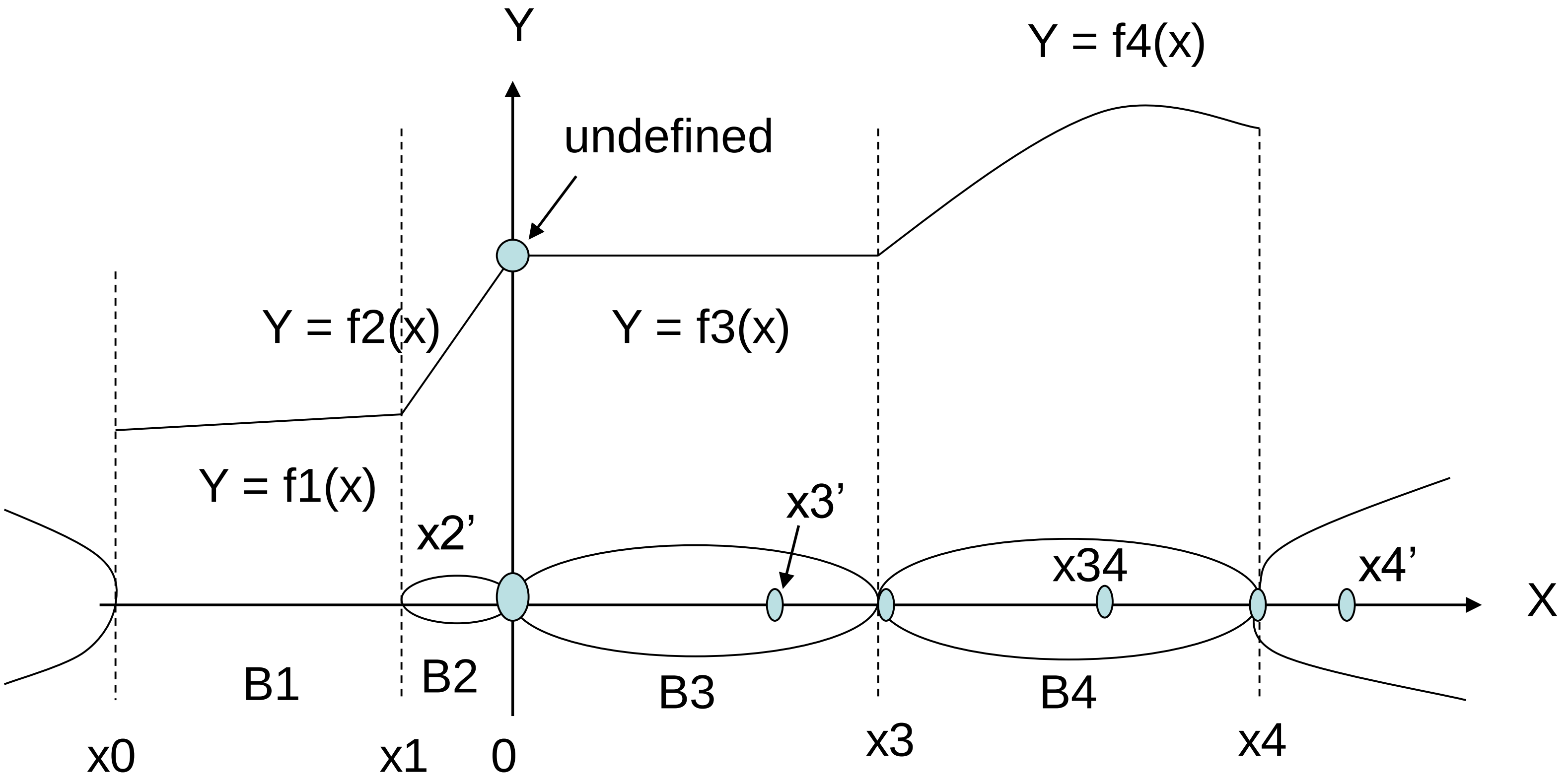
Boundary Value Analysis Testing Example



Existing Boundaries:

B1: x in [x0, x1]
B2: x in (x1, 0)
B3: x in (0, x3)
B4: x in [x3, x4]

Boundary Value Analysis Testing Example



Test Cases for Boundary #4:

x = x3’,   y = f3(x3’),   check y = ?

x = x3,   y = f4(x3),   check y = ?

x = x34, y = f4(x34),   check y=?

x = x4, y = f4(x),   check y=?

x =x4’, y = out of boundary

An Example of Using The Category-Partition Method

Test a command-line program that supports “find” operation as follows:

Command:

find

Syntax:

```
find <pattern> <file>
```

Function:

The find command is used to locate one or more instances of a given pattern in a text file. All lines in the file that contain the pattern are written to standard output. A line containing the pattern is written only once, regardless of the no. of times the pattern occurs in it.

The pattern is any sequence of characters whose length does not exceed the maximum length of a line in the file. To include a blank in the pattern, the entire pattern must be enclosed in quotes (“). To include a quotation mark in the pattern, two quotes in a row (“ ”) must be used.

An Example of Using The Category-Partition Method

Examples:

find john myfile

displays lines in the file myfile which contain *john.*

find “john smith” myfile

display lines in the file myfile which contains *john smith*.

find “john” ” smith” myfile

display lines in the file which contains *john” smith.*

When file is considered as a parameter, we need to consider the following:

- no. of occurrences of the pattern in the file.

- no. of occurrences of the pattern in a line that contains it.

- maximum line length in the file

……

An Example of Using The Category-Partition Method

Identified Category-Partitions by focusing on input parameters and related partitions::

Category partitions for this example:

Parameter “Pattern” related partitions:

- Pattern size:
    - empty
    - single character
    - many character
    - longer than any line in the file

-Quoting:
    - Pattern is quoted
    - Pattern is not quoted
    - pattern is improperly quoted

An Example of Using The Category-Partition Method

Parameter “Pattern” related partitions:

- Embedded blanks:
  - no embedded blank
  - One embedded blank
  - Several embedded blanks

- Embedded quotes:
  - no embedded quotes
  - One embedded quote
  - Several embedded quote

An Example of Using The Category-Partition Method

Parameter “File” related partitions

- File name:
    - Good file name
    - No file with this name
    - Omitted

- File access environment
    - File not accessible
    - File can’t read
    - File can’t open

- No. of occurrences of pattern in the file.
    - None
    - Exactly one
    - More than one

-Pattern occurrences on target line in the file:
    - one
    - more than one
    - None

An Example of Using The Category-Partition Method

Command line related partitions:

:

Command line:

Incorrect “command”

Correct “command” with correct parameters

Missing input parameters

Extra input parameters

PIN Requirements

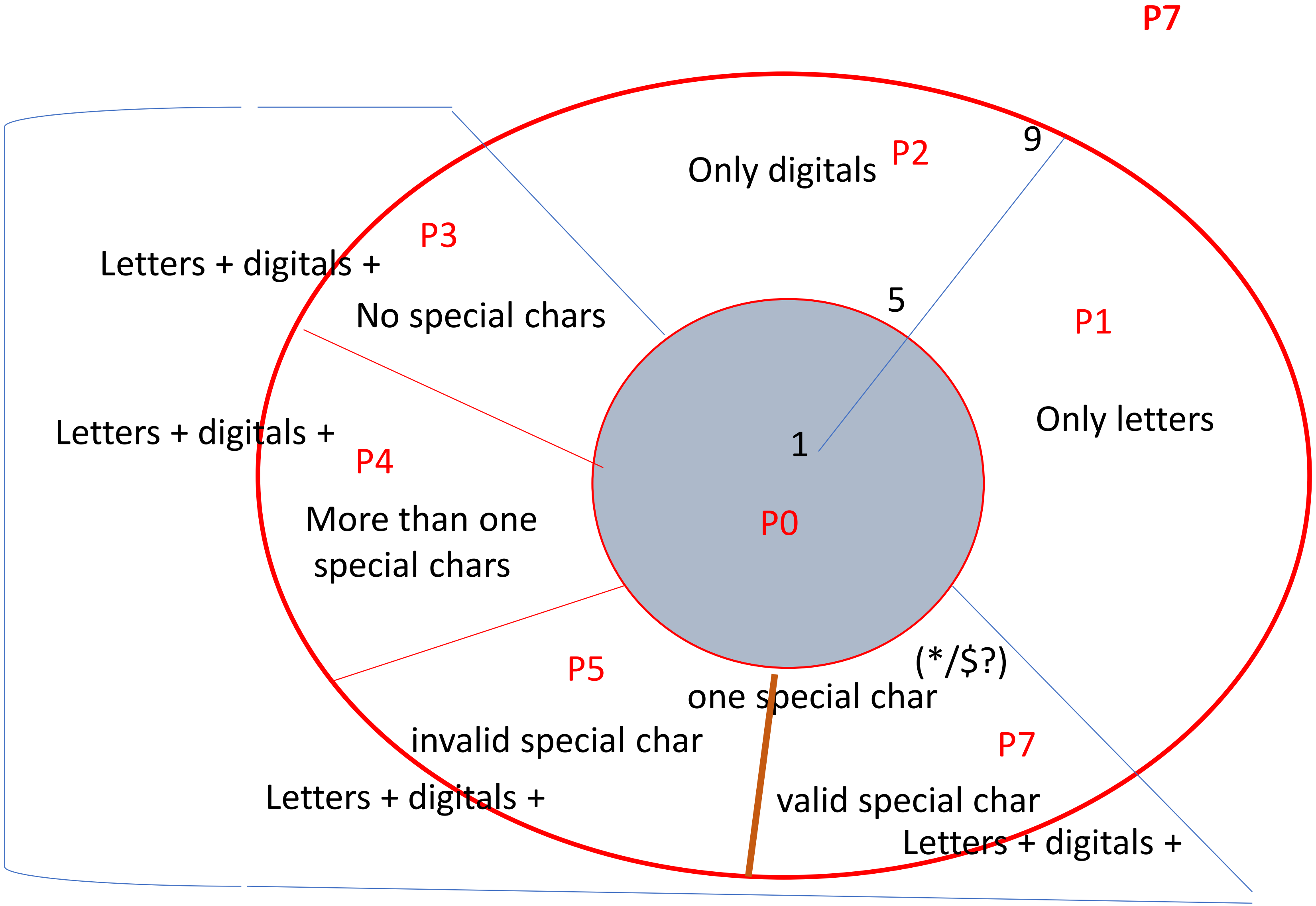
Pin’s Length should be from 5 to 9.

One valid special char(*/$?)

up-case and Low-case are considered to be the same

PIN Must include both Letters and digitals.

Diagram-Based Approach



<table>
<tr><th></th><th></th><th colspan="6"></th><th></th></tr>
<tr><td>Length</td><td>&lt;5</td><td colspan="6">5-9</td><td>&gt;9</td></tr>
<tr><td>Letters & digitals</td><td>-</td><td>Only letters</td><td colspan="4">letters + digitals +</td><td>only digitals</td><td>-</td></tr>
<tr><td rowspan="2">Special chars</td><td rowspan="2">-</td><td rowspan="2">-</td><td rowspan="2">No special chairs</td><td colspan="2">One special char</td><td rowspan="2">More than one special chars</td><td rowspan="2">-</td><td rowspan="2">-</td></tr>
<tr><td>Only one valid special char</td><td>One invalid special char</td></tr>
</table>

Category Partition Testing

Command:	Sort Sort-Pattern Input-Data-File Output-Data-File
Sort-Pattern:	Increasing Order/Decreasing Order
Input Data File:	Integer Data List
Output Data File:	Sorted Data List

Parameter #1: Sort-Pattern

- Invalidate Pattern Value
- Increasing Order
- Decreasing Order
- Empty

Parameter #2: Input-Data-File

File Name:

Existing/Correct File Name
Not Found
Not Entered

Access Environment:

Can’t Open
Not Readable
Can’t Access

Parameter #2: Input-Data-File

Data File Content:

Empty
Invalidate Data Type
Incorrect Format
Correct Data Format and Type

Data Order in Data File:

Random Order
Increasing Order
Decreasing Order

Category Partition Testing

Command: Sort Sort-Pattern Input-Data-File Output-Data-File

Sort-Pattern: Increasing Order/Decreasing Order

Input Data File: Integer Data List

Output Data File: Sorted Data List

Parameter #3: Output-Data-File

File Name:

- Existing/Correct File Name
- Not Found
- Not Entered

Access Environment:

- Can’t create/generate
- Not Readable
- Can’t Access
- Generate/Access

Parameter #3: Output-Data-File

Data Order in Data File:

- Increasing Order
- Decreasing Order

Decision Table Testing

Command:	Sort Sort-Pattern Input-Data-File Output-Data-File
Sort-Pattern:	Increasing Order/Decreasing Order
Input Data File:	Integer Data List
Output Data File:	Sorted Data List

Conditions:

Sort-Pattern – Conditions

- Existing (T/F)
- Given Increasing Order?(T/F)
- Given Decreasing Order? (T/F)

Input Data File - conditions

- Existing (T/F)
- Accessible? (T/F)
- Readable?(T/F)
- Openable?(T/F)

Content: - conditions

- Empty? (T/F)
- Increasing Order(T/F)
- Decreasing Order(T/F)
- Random Order(T/F)

Actions:???