Unit 6. Domain Testing

1. Domain Testing
2. Domain Matrix Testing

Reading: TB-Chapter 6 (6.1-6.7)

1. Domain Testing

-Domain or Equivalence partitioning is an essential technique in the arsenal of virtually every professional tester.

-It is a test strategy which uses heuristics for test data selection based on ***equivalence classes***, ***boundary values***, and ***special values***.

- An ***Equivalence class*** is a set of input values such that if any value is processed correctly (incorrectly), then it is assumed that all other values will be processed correctly (incorrectly).

- ***Boundary and special values tests*** are based on the assumptions that bugs are likely when input or state values are at or very near to minimum or maximum.

Basic Concepts

- Let $x_1, \ldots, x_n$ denote input variables corresponding to the input to a program

- The ***input space*** is an n-dimensional space represented by a vector X, also called input vector, $X = [x_1, \ldots, x_n]$

- The ***input domain*** consists of all the points representing all the allowable input combinations specified for the program in the product specification.

- An ***input sub-domain*** is a subset of the input domain; a sub-domain is sometimes defined by a set of inequalities like $f(x_1, \ldots, x_n) < K$

- A ***domain partition***, is a partition of the input domain into a number of sub-domains.
  -i.e., these sub-domains are ***mutually exclusive***, and ***collectively exhaustive***.

- A ***boundary*** is where two sub-domains meet; a boundary can be linear or nonlinear.
  -Example: with the above inequality, a boundary would be $f(x_1, \ldots, x_n) = K$

Boundary Problems: Example

-Sample Requirement: *Process employment applications based on a person's age*

0–16	Don't hire
16–18	Can hire on a part-time basis only
18–55	Can hire as a full-time employee
55–99	Don't hire

What are the issues here?

-Sample Implementation

```
If (applicantAge >= 0 && applicantAge <=16)      hireStatus="NO";
If (applicantAge >= 16 && applicantAge <=18)     hireStatus="PART";
If (applicantAge >= 18 && applicantAge <=55)     hireStatus="FULL";
If (applicantAge >= 55 && applicantAge <=99)     hireStatus="NO";
```


Boundary Problems: Categories

- **Closure problem**:
-Problem with whether the boundary points belong to the sub-domain under consideration.
-Would be an implementation that disagrees with the specification, or the specification that disagrees with the intention; e.g., an intended open boundary is specified or implemented as a closed one.

- **Boundary shift**:
-Refer to the disagreement with where exactly a boundary is between the intended and the actual boundary.
-e.g., for a boundary $f(x_1, \ldots, x_n) = K$, a small change in K is associated with a boundary shift.

- **Missing boundary**:
-Mean that two neighboring sub-domains will collapse into one sub-domain, and therefore all points in them would be treated similarly.

- **Extra boundary**:
-Mean that different points within the same sub-domain (which has been further partitioned) would receive different treatments because they belong to different equivalence classes.

Testing Strategies

-To deal with boundary problems, various domain testing strategies focusing on related sub-domains are used; these strategies are referred to as ***boundary testing strategies***.

-In these cases, the existence of "intended" or correct partitions is assumed.

- The actual specification (black box) or implementation (white box) of these partitions or boundaries may contain some mistakes.

- With this assumption, the result checking for testing can be done by using intended partitions or boundaries as oracles.

Extreme Point Combination Strategy (EPC)

-One of the oldest ***boundary testing strategies*** that is still used and supported by some testing tools.

- The basic idea is that testing for extreme values would help reveal system design and implementation problems

- The systematic definition and usage of such extreme values when multiple variables are involved give the so-called extreme point combination (EPC) strategy

Extreme Point Combination Strategy (ctd.)

-The EPC strategy involves the following steps:

1. Given a domain with *n* dimensions

2. Conduct domain analysis to identify the domain limits in each dimension

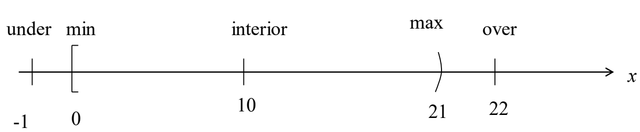
-For each variable x_i, we need to find out the maximal, "max_i", and minimal, "min_i", values for this sub-domain, and to test the limits, we define the values , "$under_i$" to be slightly under "min_i", and "$over_i$", to be slightly over "max_i".

2. Produce all the possible combinations of input with each of variable x_i taking on one of the four values, "$under_i$", "min_i", "max_i", and "$over_i$". Each of these combinations will be a test case in this *n*-dimensional space.

-The number of test cases would be 4n + 1, with 4n the cross product of those four values for each dimension, plus 1 for sampling inside the sub-domain.

Exercise 6.1: EPC strategy for 1-dimensional sub-domains

-Consider the input domain $0 \leq x < 21$



- Determine the test points according to EPC

Exercise 6.2: According to the specification, a program accepts 4 to 10 inputs, which are five-digit integers greater than 10,000, and computes their average.

Identify equivalence partitions and boundary cases.

Strengths & Weaknesses of Domain Testing

- Strengths
  - –Find highest probability errors with a relatively small set of tests.
  - –Intuitively clear approach, generalizes well

- Blind spots
  - –Errors that are not at boundaries or in obvious special cases.
  - –The actual sets of possible values are often unknowable.
  - –The selection of partition has ***often*** no necessary relationship to the discovery of bugs– in essence, they are ***pure guesses***.

Other Examples

Exercise 6.3: Equivalence Analysis based on Program Specification

Consider the specification of a search routine that searches a sequence of elements for a given element (the key). It returns the position of that element in the sequence. Identify equivalence classes from the system specification, and derive accordingly sample test cases.

```
Procedure Search (key: ELEM; T: SEQ of ELEM; found: in out Boolean; L: in out elem_index)

    Pre-condition
        /*The sequence has at least one element*/
        T'first ≤ T'last

    Post-condition
        /*The element is found and is referenced by L*/
        found and T(L) = key
        or
        /*The element is not in the sequence*/
        (not found and not (exists i, T'first ≤ i ≤ T'last, T(i)=key))
```


Exercise 6.4: Equivalence Analysis based on Program Code

Identify equivalence classes and derive test cases for the search routine using this time the following implementation based on a binary search function.

```
//The search function takes an array of ordered objects and a key and returns an object with 2
// attributes: index and found. The key is –1 if the element is not found.

Class BinSearch {
        public static void search(int key, int[] elemArray, Result r) {
                int bottom = 0;
                int top = elemArray.length – 1;
                int mid;
                r.found = false; r.index=-1;
                while (bottom <= top) {
                        mid = (top + bottom)/2;
                        if (elemArray [mid] == key) {
                                r.index = mid;
                                r.found = true;
                                return;
                        }
                        else {
                                if (elemArray[mid] < key) bottom = mid + 1;
                                else top = mid – 1;
                        }
                }
        }
}
```

2. Domain Matrix Testing

-*Systematic and elaborated form of equivalence analysis*, which attempts to define the boundaries of the domain by exploiting (possibly) inherent (mathematical) characteristics of the system (e.g., invariants, state predicates etc.).

-A program is divided into different execution paths, so-called control flows.

- In order to ensure that a program is running under the right path, a path condition, usually a ***predicate*** expression, must be **explicitly** specified.
- The domain testing fault model reveals anomalies indicated by incorrect path conditions.

-Domain analysis consists of identifying the test domain corresponding to the path conditions and partitioning it in suitable sub-domains.

-The domain and its boundary conditions can be defined based on either specification models or program source codes.

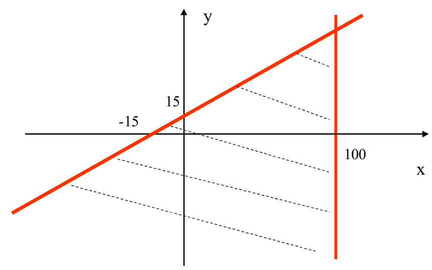
Example:

•Suppose we want to develop test cases for the following C++ function:

void compute (float x, float y) {/*…*/};

•Suppose that the specification of this function requires that input parameters meet the following assertion:

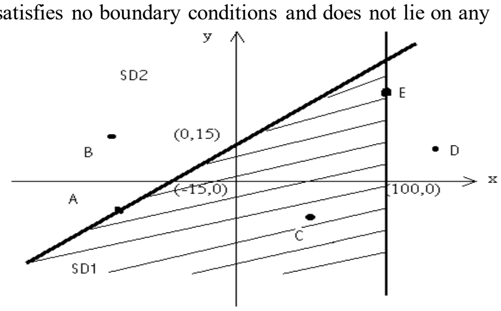
Assert ((x <= 100) && (y-x <=15))



Test Points

-In domain matrix testing, test data design consists of identifying special points in the domain: ***On, Off, In*** and ***Out*** points.

- ***On point***: a value that lies on a boundary.
- ***Off point***: a value not on a boundary.
- ***In point***: a value that satisfies all boundary conditions and does not lie on a boundary.
- ***Out point***: a value that satisfies no boundary conditions and does not lie on any boundary.



The One-by-One Selection Criteria

-The 1×1 domain testing strategy calls for one **on** point and one **off** point for each domain boundary.
-The selection rules for **on** and **off** points are straightforward.

- One **on** point and one **off** point for each relational condition; these points are to be as close as possible.
- One **on** point and two **off** points for each strict equality condition; again test points should be as close as possible.
  Example: suppose condition (x=10) is incorrectly implemented as (x>=10), test points (x=10 (on), x= 9 (off)) would not reveal such bug; the addition of (x=11) would reveal it.
- One **on** point and one **off** point for each nonscalar type (e.g., string, boolean, enumerations). The boundaries of such unordered data types are closed and binary: the variable either conforms to the condition or not.
- One **on** point and one **off** point for nonlinear boundaries.

Domain Matrix

-The results of the domain analysis are expressed in a domain matrix.
-A ***domain matrix*** consists of a ***table used to build a complete test suite***.

-The table includes the following items:

- ***Variable:*** list all the input variables in a domain.
- ***Condition***: indicate the boundary conditions for each variable.
- ***Type:*** specify the kind of points (e.g., on, off, in, out).
- ***Test Cases:*** correspond to the test points generated.
- ***Expected results:*** expected output or messages.

-The table may be either a column-like or a row-like table.

- In a row-like table, the items are displayed in rows, while in column-like, they are displayed in column.

Variable	**Condition**	**Type**	**Test cases**			
			1	**2**	**3**	**4**
x	x≤ 100	On	100			
		Off		101		
	Typical	In			99	99
y	y ≤ x+15	On			114	
		Off				116
	Typical	In	113	112		
Expected Result			Accept	Reject	Accept	Reject
Expected Output 1						
…						
Expected Output *n*						

- The term ***Typical*** is used in case where no restrictions are specified for corresponding variable.
- ***Expected result***: indicate whether the test cases should be either accepted or rejected
  - ***Accept*** term: specify that the IUT should process the test case inputs and produce the indicated output.
  - ***Reject*** term: specify that the test case inputs should not be processed, and an appropriate error response should be produced by the IUT.

-(In a column-like matrix) each column of values is a test case.

- Only **on** or **off** point appears in a test case; these values fall on the diagonal of the matrix.
- In addition of the **on/off** points recommended by the 1×1 selection criteria, **in** points should be generated for all other variables in each test case.
- **In** points are chosen after the **on** and **off** points are determined. They can be developed by guessing, by analyzing the situation, or by using a pseudorandom algorithm.
- Try to avoid repeating **in** point values, as they will increase the chance of revealing an unexpected bug.

Exercise 6.6: Consider a banking application, in which corporate credit card accounts are subject to a business policy that guarantees, on one hand, a credit limit ranging between \$1000 and \$100,000, and requires, on the other hand, that the balance always remains within the credit limits. The bank also requires that its corporate customers protect their loan by purchasing a special insurance policy.

-Consider the following Java class representing a customer account:

```
public class Account {
        double balance;
        double creditLimit;
        boolean insured; //true when the customer has an insurance and false otherwise.

        boolean withdraw (double amount) {                    -bug
                boolean result = false;
                if (balance-amount+creditLimit >0) {
                        balance = balance – amount;
                        result = true;
                }
                return result;
        }

        //…more methods…
}
```


-The class invariant derived from the business policy is defined as:

```
assert (
        (balance + creditLimit >= 0) &&
        (1000.00<=creditLimit && creditLimit <= 100000.00) &&
        (insured = true)
        );
```

- Generate test cases using the domain matrix testing technique.

Variable	Condition	Type	Test cases							
			1	2	3	4	5	6	7	8
creditLimit	>=1000	On								
		Off								
	<= 100000	On								
		Off								
	Typical	In								
balance	>= -creditLimit	On								
		Off								
	Typical	In								
insured	= true	On								
		Off								
	Typical	In								
amount	Typical	In								
Expected Result										
(Expected) New balance										
(Expected) Transaction result										

Variable	Condition	Type	Test cases							
			1	2	3	4	5	6	7	8
creditLimit	>=1000	On	1000							
		Off								
	<= 100000	On								
		Off								
	Typical	In								
balance	>= -creditLimit	On								
		Off								
	Typical	In	2000							
insured	= true	On								
		Off								
	Typical	In	True							
amount	Typical	In	500							
Expected Result			Accept							
(Expected) New balance			1500							
(Expected) Transaction result			true							

Variable	Condition	Type	Test cases							
			1	2	3	4	5	6	7	8
creditLimit	>=1000	On	1000							
		Off		500						
	<= 100000	On								
		Off								
	Typical	In								
balance	>= -creditLimit	On								
		Off								
	Typical	In	2000	0						
insured	= true	On								
		Off								
	Typical	In	True	True						
amount	Typical	In	500	100						
Expected Result			Accept	Reject						
(Expected) New balance			1500	0						
(Expected) Transaction result			true	false						

Variable	Condition	Type	Test cases							
			1	2	3	4	5	6	7	8
creditLimit	>=1000	On	1000							
		Off		500						
	<= 100000	On			100000					
		Off								
	Typical	In								
balance	>= -creditLimit	On								
		Off								
	Typical	In	2000	0	1000					
insured	= true	On								
		Off								
	Typical	In	True	True	True					
amount	Typical	In	500	100	2					
Expected Result			Accept	Reject	Accept					
(Expected) New balance			1500	0	998					
(Expected) Transaction result			true	false	true					

25

Variable	Condition	Type	Test cases							
			1	2	3	4	5	6	7	8
creditLimit	>=1000	On	1000							
		Off		500						
	<= 100000	On			100000					
		Off				1000000				
	Typical	In								
balance	>= -creditLimit	On								
		Off								
	Typical	In	2000	0	1000	1500				
insured	= true	On								
		Off								
	Typical	In	True	True	True	True				
amount	Typical	In	500	100	2	1200				
Expected Result			Accept	Reject	Accept	Reject				
(Expected) New balance			1500	0	998	1500				
(Expected) Transaction result			true	false	true	false				

26

Variable	Condition	Type	Test cases							
			1	2	3	4	5	6	7	8
creditLimit	>=1000	On	1000							
		Off		500						
	<= 100000	On			100000					
		Off				1000000				
	Typical	In					15000			
balance	>= -creditLimit	On					-15000			
		Off								
	Typical	In	2000	0	1000	1500				
insured	= true	On								
		Off								
	Typical	In	True	True	True	True	True			
amount	Typical	In	500	100	2	1200	500			
Expected Result			Accept	Reject	Accept	Reject	Accept			
(Expected) New balance			1500	0	998	1500	-15000			
(Expected) Transaction result			true	false	true	false	false			

27

Variable	Condition	Type	Test cases							
			1	2	3	4	5	6	7	8
creditLimit	>=1000	On	1000							
		Off		500						
	<= 100000	On			100000					
		Off				1000000				
	Typical	In					15000	2000		
balance	>= -creditLimit	On					-15000			
		Off						-2001		
	Typical	In	2000	0	1000	1500				
insured	= true	On								
		Off								
	Typical	In	True	True	True	True	True	True		
amount	Typical	In	500	100	2	1200	500	1000		
Expected Result			Accept	Reject	Accept	Reject	Accept	Reject		
(Expected) New balance			1500	0	998	1500	-15000	-2001		
(Expected) Transaction result			true	false	true	false	false	false		

28

Variable	Condition	Type	Test cases							
			1	2	3	4	5	6	7	8
creditLimit	>=1000	On	1000							
		Off		500						
	<= 100000	On			100000					
		Off				1000000				
	Typical	In					15000	2000	10000	
balance	>= -creditLimit	On					-15000			
		Off						-2001		
	Typical	In	2000	0	1000	1500			1500	
insured	= true	On							True	
		Off								
	Typical	In	True	True	True	True	True	True		
amount	Typical	In	500	100	2	1200	500	1000	20000	
Expected Result			Accept	Reject	Accept	Reject	Accept	Reject	Accept	
(Expected) New balance			1500	0	998	1500	-15000	-2001	1500	
(Expected) Transaction result			true	false	true	false	false	false	false	

29

Variable	Condition	Type	Test cases							
			1	2	3	4	5	6	7	8
creditLimit	>=1000	On	1000							
		Off		500						
	<= 100000	On			100000					
		Off				1000000				
	Typical	In					15000	2000	10000	50000
balance	>= -creditLimit	On					-15000			
		Off						-2001		
	Typical	In	2000	0	1000	1500			1500	-10000
insured	= true	On							True	
		Off								False
	Typical	In	True	True	True	True	True	True		
amount	Typical	In	500	100	2	1200	500	1000	20000	40000
Expected Result			Accept	Reject	Accept	Reject	Accept	Reject	Accept	Reject
(Expected) New balance			1500	0	998	1500	-15000	-2001	1500	-10000
(Expected) Transaction result			true	false	true	false	false	false	false	false

30

Variable	Condition	Type	Test cases							
			1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
creditLimit	>=1000	On	**1000**							
		Off		**500**						
	<= 100000	On			**100000**					
		Off				**1000000**				
	Typical	In					**15000**	**2000**	**10000**	**50000**
balance	>= -creditLimit	On					**-15000**			
		Off						**-2001**		
	Typical	In	**2000**	**0**	**1000**	**1500**			**1500**	**-10000**
insured	= true	On							**True**	
		Off								**False**
	Typical	In	**True**	**True**	**True**	**True**	**True**	**True**		
amount	Typical	In	**500**	**100**	**2**	**1200**	**500**	**1000**	**20000**	**40000**
Expected Result			**Accept**	**Reject**	**Accept**	**Reject**	**Accept**	**Reject**	**Accept**	**Reject**
(Expected) New balance			**1500**	**0**	**998**	**1500**	**-15000**	**-2001**	**1500**	**-10000**
(Expected) Transaction result			**true**	**false**	**true**	**false**	**false**	**false**	**false**	**false**