Input Space Partitioning

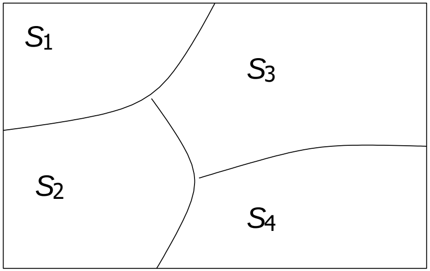
Meenakshi D'Souza

International Institute of Information Technology
Bangalore.

Partitions of a set

- Given a set S, a partition of S is a set $\{S_1, S_2, \ldots, S_n\}$ of subsets of S such that
  - The subsets S_i of S are pair-wise disjoint, i.e., $S_i \cap S_j = \emptyset$The
  - union of the subsets S_i is the entire set S, i.e., $\cup_i S_i = S$.

S



Partitions and Testing

- The set that is split into partitions while doing testing is the input domain.
- Input domain can be several different sets, one for each input. We may or may not consider all the inputs while doing partitioning.
- Each partition represents one characteristic of the input domain, the program under test will behave in the same way for any element from the partitions.
- There is an underlying equivalence relation that influences the partitions, so input space partitioning is popularly known as equivalence (domain) partitioning.

Partitions and Characteristics

- Each partition is based on some characteristic of the program P that is being tested.
- Examples of characteristics:
  - Input x is null.
  - Order of file F (sorted, inverse sorted, arbitrary)
- Each characteristic allows a tester to define one partition.

Partitions and Characteristics

Characteristics that define partitions must ensure that the partition satisfies two properties:

- completeness: The partitions must cover the entire domain.
- disjoint: The partitions must not overlap.

For e.g., consider the characteristic "Order of file F".

- Order of file F:
    - S_1: Sorted in ascending order.
    - S_2: Sorted in descending order.
    - S_3: Arbitrary order.

  This is *not* a valid partitioning. If a file is of length 0 or 1, then the file will belong to all the three partitions.

Input Space Partitioning: Example

Consider a software system that computes income tax based on adjusted gross income (AGI) according to the following rules:

- If AGI is between $1 and $29,500, the tax due is 22% of AGI.
- If AGI is between $29,501 and $58,500, the tax due is 27% of AGI.
- If AGI is between $58,501 and $100 billion, the tax due is 30% of AGI.

Input Space Partitioning: Example, contd.

We get five partitions as below:

- $1 \leq AGI \leq 29{,}500$: Valid input.
- $AG1 < 1$: Invalid input.
- $29{,}501 \leq AGI \leq 58{,}500$: Valid input.
- $58{,}501 \leq AGI \leq 100$ billion: Valid input.
- $AGI > 1$ billion: Invalid input.

Five test cases, each containing one number for AGI in the above range will suffice for testing the tax requirement based on AGI.

Input domain modelling

The following are the steps in input domain modelling.

1. Identification of testable functions.
2. Identify all the parameters that can affect he behaviour of a given testable function. These parameters together form the input domain of the function under test.
3. Modelling of the input domain identified in the previous step: Identify the characteristics and partition for each characteristic.
4. Get the test inputs: A test input is a tuple of values, one for each parameter. The test input should belong to exactly one block from each characteristic.

Input domain modelling: Two approaches

- Input domain can be modelled in several different ways, needs extensive knowledge of the *domain*.
- Both valid and invalid inputs need to be considered.
- Two broad approaches available:
    - Interface-based approach
    - Functionality-based approach

Interface-based input domain modelling

This method considers each parameter in isolation.

- Strengths: It is easy to identify the characteristics, hence easy to model the input domain.
- Weaknesses:
  - Not all information that is available to the test engineer will be reflected in the interface domain model, the model can be incomplete.
  - Some parts of the functionality may depend on the combinations of specific values of several interface parameters. Analysing in isolation will miss the combinations.

Functionality based input domain modelling

This method identifies characteristics based on the overall functionality of the system/function under test, rather than using the actual interface.

- Strengths:
  - There is a wide spread belief that this approach yields better results than interface-based approach.
  - Since this is based on the requirements, test case design can start early in the development lifecycle.
- Weaknesses:
  - Since it is based on functionality, identifying characteristics is far from trivial.
  - This, in turn, makes test case design difficult.

Example

Consider the following method (code not given):

```
public boolean findElement(List list, Object element)
// Effects: If list or element is null throw NullPointerException
// else returns true if element is in the list, false otherwise
```

Characteristics of interface-based approach

- Characteristics in this approach are easy, directly based on the individual inputs.
- Inputs can be obtained from specifications, hence, this is black-box testing.
- For the list example:
  - list is null: b_1= True, b_2= False.
  - list is empty: b_1 = True, b_2 = False.

Characteristics of functionality-based approach

- Pre-conditions, post-conditions are typical sources for identifying functionality-based characteristics.
- Implicit and explicit relationships between variables are another good source.
- Missing functionality is another characteristic.
- Domain knowledge is needed.
- For the list example:
    - Number of occurrences of element in list: b_1 = 0, b_2 = 1, b_3 = More than 1.
    - Element occurs first in list: b_1 = True, b_2 = False.

Choosing partitions

- This is a key step in input space partitioning.
- There should be a balance between the number of partitions and their effectiveness.
- For each characteristic with n partitions, the total number of combinations increases by a factor of n in functionality based approach.
- Lesser combinations might result in testing that is less effective, more combinations is likely to find more faults.

Identifying values

Some strategies for identifying values are:

- *Valid values*: Include at least one set of valid values.
- *Sub-partitions*: A range of valid values can be further partitioned such that each sub-partition exercises a different part of the functionality.
- *Boundaries*: Include values at and close to the boundaries (BVA).
- *Invalid values*: Include at least one set of invalid values.
- *Balance*: It might be cheap or free to add more partitions to characteristics that have less partitions.
- *Missing partitions*: Union of all the partitions should be the complete input space for that characteristic.
- *Overlapping partitions*: There should be no overlapping partitions.

Multiple partitions of the input domain

- Typically, input domain has several different inputs, each of which can be partitioned.
- In addition, each input domain can be partitioned in several different ways.
- For interface-based ISP, inputs are considered separately.
- For functionality-based ISP, input combinations cutting across partitions need to be considered.
- There are various criteria that deal with how to consider *combinations amongst multiple partitions*.

Combination strategies criteria

The list of various coverage criteria that we will be considering over input domains and their partitions are:

- All Combinations Coverage (ACoC)
- Each Choice Coverage (ECC)
- Pair-Wise Coverage (PWC)
- T-Wise Coverage (TWC)
- Base Choice Coverage (BCC)
- Multiple Base Choices (MBCC)

The above coverage criteria will cover most of the different ways in which we can partition the inputs to test the software artifact.

All Combinations Coverage (ACoC)

- All Combinations Coverage (ACoC): All combinations of blocks from all characteristics must be used.
- Example: If we have three partitions as [A,B], [1,2,3] and [x,y], then ACoC will have the following twelve tests:

  (A,1,x) (B,1,x)
  (A,1,y) (B,1,y)
  (A,2,x) (B,2,x)
  (A,2,y) (B,2,y)
  (A,3,x) (B,3,x)
  (A,3,y) (B,3,y)
- A test suite for ACoC will have a unique test for each combination.
- Total number of tests will be $\Pi_{i=1}^{n} B_i$, B_i is the number of blocks for each partition, n is the number of partitions.

ACoC, contd.

- ACoC is just an exhaustive testing of considering all possible partitions of the input domain and testing each combination of partitions.
- Apart from the partitions themselves, ACoC has no other advantages, it is like exhaustive testing with respect to the partitions.
- ACoC might not be necessary all the time.
- The various coverage criteria that we will define now will describe how to consider different possible combinations of partitions.

Each Choice Coverage (ECC)

- Each Choice Coverage: One value from each block for each characteristic must be used in at least one test case.
- Example: If we have three partitions as [A,B], [1,2,3] and [x,y], then ECC will have only three tests: (A,1,x), (B,2,y) and (A,3,x).
- If the program under test has n parameters $q_1, q_2, \ldots, q_n$, and each parameter q_i has B_i blocks, then, a test suite for ECC will have at least $Max_{i=1}^{n} B_i$ values.
- ECC is a *weak criterion*, there is a lot of choice available to the tester to choose the values, no combinations are considered.
- ECC will not be effective for arbitrary choice of test values.

Pair-wise Coverage

- Pair-wise coverage: A value from each block for each characteristic must be combined with a value from every block for each other characteristic.
- A test suite that satisfies PWC will pair each value with each other value or have at least $(Max_{i=1}^{n} B_i)^2$ values.

PWC, contd.

- For our e.g., with partitions [A,B], [1,2,3] and [x,y], PWC will need sixteen tests to cover the following combinations:

(A,1)	(B,1)	(1,x)
(A,2)	(B,2)	(1,y)
(A,3)	(B,3)	(2,x)
(A,x)	(B,x)	(2,y)
(A,y)	(B,y)	(3,x)
		(3,y)

- PWC allows the same test case to cover more than one unique pair of values. The above combinations can be combined in several ways. One way is:

(A,1,x)	(B,1,y)
(A,2,x)	(B,2,y)
(A,3,x)	(B,3,y)
(A,-,y)	(B,-,x)

T-wise Coverage

- A natural extension of PWC is to require t values instead of pairs.
- T-Wise Coverage: A value from each block for each group of t characteristics must be combined.
- If the value for T is chosen to be the number of partitions, then TWC is the same as ACoC.
- A test suite that satisfies TWC will have at least $(Max_{i=1}^{n} B_i)^t$ values.
- TWC is expensive in terms of the number of tests, empirical studies indicate that going beyond PWC is mostly not useful.

Ways of considering combinations

- ACoC considers all combinations, ECC considers each combination.
- PWC and TWC considers combinations *blindly* without regard for which values are being combined.
- The two criteria we are going to define consider the notion of *important* block for each partition. Such a block is called a base choice.

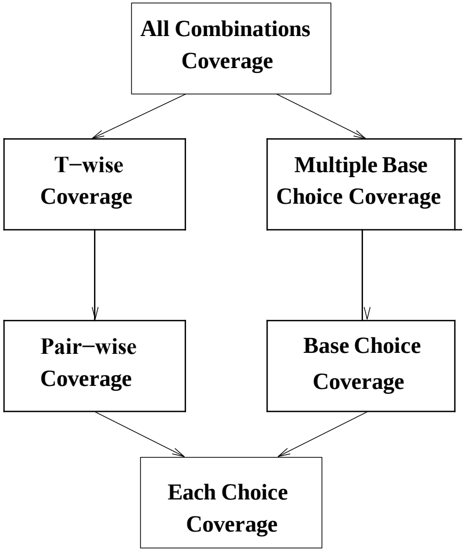
Base Choice Coverage

- Base Choice Coverage: A base choice is chosen for each characteristic, and a base test is formed by using the base choice for each characteristic. Subsequent tests are chosen by holding all but one base choice constant and using each non-base choice in each other characteristic.
- For our e.g., suppose the base choices blocks are A, 1 and x. Then, the base choice test is (A,1,x) and the following additional tests are needed: (B,1,x), (A,2,x), (A,3,x), (A,1,y).
- A test suite that satisfies BCC will have one base test, plus one test for each remaining block for each partition. Totally, $1 + \Sigma_{i=1}^{n}(B_i - 1)$.

Multiple Base Choices Coverage

- Sometimes, the tester may have trouble in choosing a single base choice; multiple base choices will be needed.
- Multiple Base Choice Coverage: At least one, and possibly more, base choice blocks are chosen for each characteristic, and base tests are formed by using each base choice for each characteristic, at least once. Subsequent tests are chosen by holding all but one base choice constant for each base test and using each non-base choice in each other characteristic.
- Assuming m_i base choices for each characteristic and a total of M base tests, MBCC requires $M + \Sigma_{i=1}^{n}(M * (B_i - m_i))$ tests.

ISP criteria: Subsumption relations



TriTyp example: ACoC criterion

- There are four partitions based on the relationship of length of a side to being greater than/equal to 1 and equal to/less than 0.
- There should be 64 different tests to satisfy All Combinations Criterion for this partitioning.
- Here are some of the tests from the data given in the table in an earlier slide:
  - (2,2,2), (2,2,1), (2,2,0, (2,2,-1), (2,1,2), etc. (totally 16 of them beginning with 2 as the length of side 1).
  - (1,2,2), (1,2,1), (1,2,0), (1,2,-1), (1,1,2) etc. (totally 16 of them beginning with 1 as the length of side 1).
  - (0,2,2), (0,2,1), (0,2,0), (0,2,-1), (0,1,2) etc. (totally 16 of them beginning with 0 as the length of side 1).
  - (-1,2,2), (-1,2,1), (-1,2,0), (-1,2,-1), (-1,1,2) etc. (totally 16 of them beginning with -1 as the length of side 1).

TriTyp example: PWC criterion

- There are four partitions based on the relationship of length of a side to being greater than/equal to 1 and equal to/less than 0.
- There should be 16 different tests to satisfy Pair-wise Criterion for this partitioning.
- Here is one possibility for the tests from the data given in the table in an earlier slide:
  {(2,2,2), (2,1,1), (2,0,0), (2,-1,-1), (1,2,1), (1,1,2), (1,0,-1), (1,-1,0), (0,2,0), (0,1,-1), (0,0,2), (0,-1,1), (-1,2,-1), (-1,1,0), (-1,0,1), (-1,-1,2) }

TriTyp example: MBCC criterion

- There are four partitions based on the relationship of length of a side to being greater than/equal to 1 and equal to/less than 0.
- We consider 2 and 1 to be base choices for side 1. This gives two base tests: (2,2,2) and (1,2,2).
- We get totally 2 (base) + 6 + 6 + 6 = 20 tests. Four of these are redundant. So, we get totally 16 tests.
- The tests are {(2,2,2), (1,2,2) (2,1,2), (2,0,2), (2,-1,2), (2,2,1), (2,2,0), (2,2,-1), (0,2,2), (-1,2,2), (1,2,1), (1,2,0), (1,2,-1), (1,1,2), (1,0,2), (1,-1,2)}.

Infeasible combinations of partitions

- Some combinations of partitions can be infeasible in the input domain model.
- For e.g., consider the `boolean findElement(list, element)` method with the following partitions:

	Partitions			
Characteristics	**1**	**2**	**3**	**4**
A: length and contents	one element	> than one, unsorted	> than one, sorted	> than one, all identical
B: match	element not found	element found once	element found more than once	–

- Invalid combinations are (**A1**,**B3**) and (**A4**, **B2**).

Constraints among partitions

- Constraints are relations between partitions from different characteristics.
- Two kinds of constraints:
  - A block from one characteristic *cannot* be combined with a block from another characteristic.
  - A block from one characteristic *must be* combined with a block from another characteristic.

ISP coverage criteria and constraints among partitions

- For ACoC, PWC and TWC, the only option is to drop the infeasible combinations from consideration.
- Constraints can be handled better with BCC and MBCC criteria. The base case(s) can be altered to handle infeasible constraints.

COURTESY:MEENAKSHI DSOUZA,IIIT ,BANGLORE