Input Space Partitioning

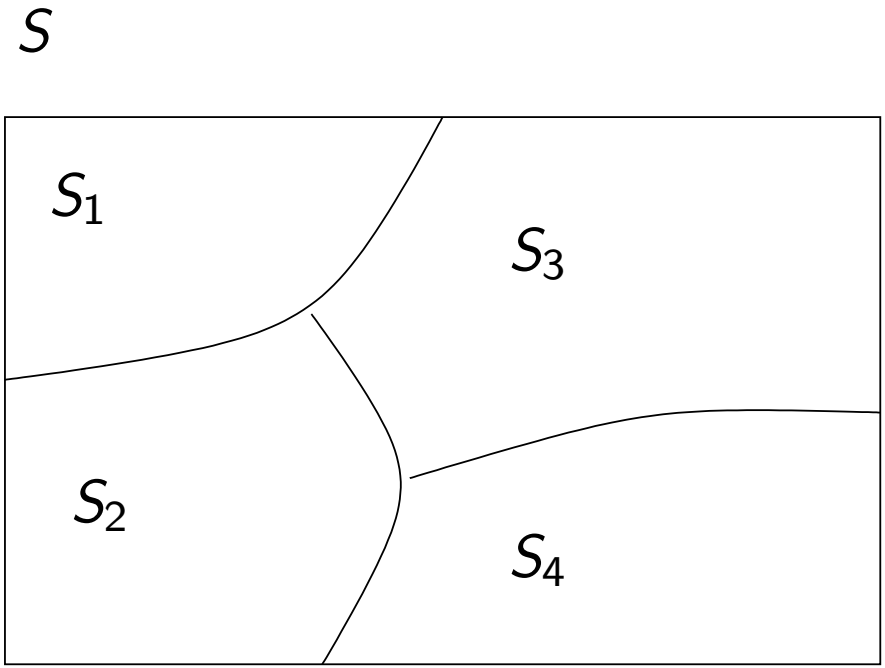
Meenakshi D'Souza

International Institute of Information Technology
Bangalore.

September 21, 2020

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., $\cup_i S_i = S$.



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:

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

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.
- Order of file F:
  - Sorted in ascending order: $S_1 = \text{True}$, $S_2 = \text{False}$.
  - Sorted in descending order: $S_1 = \text{True}$, $S_2 = \text{False}$.

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.