Domain Testing

(Naik & Tripathy, 2008)

Contents

[Input Domain 2](#_Toc443576034)

[Identifying Domains 2](#_Toc443576035)

[Closed Boundary 4](#_Toc443576036)

[Open Boundary 4](#_Toc443576037)

[Closed Domain 4](#_Toc443576038)

[Open Domain 4](#_Toc443576039)

[Extreme Point 4](#_Toc443576040)

[Adjacent Domains 4](#_Toc443576041)

[Domain Error 4](#_Toc443576042)

[Domain Error Types 4](#_Toc443576043)

[Closure Error 4](#_Toc443576044)

[Shifted-Boundary Error 5](#_Toc443576045)

[Tilted-Boundary Error 5](#_Toc443576046)

[Domain Testing 5](#_Toc443576047)

[ON Point 5](#_Toc443576048)

[OFF Point 5](#_Toc443576049)

[Test Selection Criteria 6](#_Toc443576050)

# Input Domain

The input domain of a program is the set of all input data to the program. The input domain may be sub-divided into a finite number of subdomains, where each distinct program path is mapped to a particular subdomain. A subdomain is a set of input values for which the program performs the same computation for every member of the set.

## Identifying Domains

1. Draw a CFG from the given source code
2. Find all possible interpretations of the predicates.
3. Analyze the interpreted predicates to identify domains

|  |
| --- |
| int CodeDomain(int x, int y)  {  int c = x + y;    int d;  if(c > 5) d = c - x / 2;  else d = c + x / 2;    int k;  if(d >= c + 2) k = x + d / 2;  else k = y + d / 4;    return k;  } |
|  |

|  |  |  |
| --- | --- | --- |
| **Node** | **Description** | **Interpretation** |
| **1** | <x, y> |  |
| **2** | c = x + y |  |
| **3** | c > 5 | x + y > 5 |
| **4 🡺 3(T)** | d = c – x / 2 | d = (x + y) – x / 2 |
| **5 🡺 3(F)** | d = c + x / 2 | d = (x + y) + x / 2 |
| **6** | d >= c + 2 | **3(T)**  (x + y) – x / 2 >= (x + y) + 2  - x / 2 >= 2  -x >= 4  x <= -4  **3(F)**  (x + y) + x / 2 >= (x + y) + 2  x / 2 >= 2  x >= 4 |

Node 3 has 1 interpretation: x + y > 5 (Open boundary at x + y = 5)

Node 6 as 2 interpretations:

* If c > 5 then x <= -4 (Closed boundary at x = -4)
* If c <= 5 then x >= 4 (Closed boundary at x = 4)

### Closed Boundary

A boundary is closed if the points on the boundary are included in the domain of interest.

### Open Boundary

A boundary is open if the points on the boundary do not belong to the domain of interest.

There are 4 subdomains:

1. **TT**: x + y > 5 and x <= -4 (Open boundary at x + y = 5, Closed boundary at x = -4)
2. **TF**: x + y > 5 and x > -4 (Open boundary at x + y = 5, Open boundary at x = -4)
3. **FT**: x + y <= 5 and x >= 4 (Closed boundary at x + y = 5, Closed boundary at x = 4)
4. **FF**: x + y <= 5 and x < 4 (Closed boundary at x + y = 5, Open boundary at x = 4)

### Closed Domain

A domain is closed if all of its boundaries are closed.

### Open Domain

A domain is open if some of its boundaries are open.

### Extreme Point

An extreme point is a point where two or more boundaries cross.

### Adjacent Domains

Two domains are adjacent if they have a boundary inequality in common.

# Domain Error

A domain error occurs when a specific input data causes the program to execute an unintended path in the program, in other words, the input data is placed in the wrong subdomain. This is typically caused by a fault in one or more conditional statements.

## Domain Error Types

### Closure Error

A closure error occurs if a boundary is open when the intention is to have a closed boundary.

* E.g. <= is implemented as <
* E.g. < is implemented as <=

### Shifted-Boundary Error

A shifted-boundary error occurs when the implemented boundary is parallel to the intended boundary.

* E.g. x + y > 5 is implemented as x + y > 4

### Tilted-Boundary Error

A titled-boundary error occurs when one of the constant coefficients of a variable used in a predicated expression is incorrect.

* E.g. (x + 1.1 \* y) > 5 is implemented as (x + 1.2 \* y) > 5

# Domain Testing

Domain errors are defined and then test data are selected to detect those errors.

## ON Point

Given a boundary, an **ON** point is a point on the boundary or ‘very close’ to the boundary.

Consider:

is an ON point

is an ON point but is not

Consider: (the boundary is defined by)

is an ON point (but not in the domain)

is an ON point but is not

## OFF Point

An OFF point of a boundary lies away from the boundary. If the domain is open w.r.t. the boundary then the OFF point is ***inside the domain*** within an ε-distance from the boundary. If the domain is closed w.r.t the boundary then the OFF point is ***outside the domain*** within an ε-distance from the boundary.

Consider D1: (Closed boundary)

lies outside the domain so is an OFF point

Consider D2: (Open boundary, adjacent to D1)

lies inside the domain so is an OFF point

When testing a closed boundary, the ON points are in the domain under test, whereas OFF points are in an adjacent domain.

When testing an open boundary, the ON points are in an adjacent domain, whereas the OFF points are in the domain being tested.

## Test Selection Criteria

For each domain and for each boundary, select three points A, C, and B in an ON-OFF-ON sequence.

1. **TT**: x + y > 5 and x <= -4 (Open domain 🡺 OFF points are inside the domain)

* Open boundary at x + y = 5
  + ON: A => x = -4, y = 9
  + OFF: C => x = -4.1, y = 9.2
  + ON: B => x = -4.0001, y = 9.0002
* Closed boundary at x = -4
  + ON: A => x = -4
  + OFF: C => x = -4.1
  + ON: B => x = -4.0001

1. **TF**: x + y > 5 and x > -4 (Open domain 🡺 OFF points are inside the domain)

* Open boundary at x + y = 5
  + ON: A => x = -4, y = 9
  + OFF: C => x = -3.9, y = 9
  + ON: B => x = -3.999, y = 9
* Open boundary at x = -4
  + ON: A => -4
  + OFF: C => x = -3.9
  + ON: B => x = -3.999

1. **FT**: x + y <= 5 and x >= 4 (Closed domain 🡺 OFF points are outside the domain)

* Closed boundary at x + y = 5
  + ON: A => x = 4, y = 1
  + OFF: C => x = 3.9, y = 1.2
  + ON: B => x = 4.0001, y = 0.9998
* Closed boundary at x = 4
  + ON: A => x = 4
  + OFF: C => x = 3.9
  + ON: B => x = 4.0001

1. **FF**: x + y <= 5 and x < 4 (Open domain 🡺 OFF points are inside the domain)

* Closed boundary at x + y = 5
  + ON: A => x = 4, y = 1
  + OFF: C => x = 3.9, y = 1
  + ON: B => x = 3.999, y = 1
* Open boundary at x = 4
  + ON: A => x = 4
  + OFF: C => x = 3.9
  + ON: B => x = 3.999

# Setting Up the Test Cases in NUnit

(Nunit, 2015)

**TestCaseAttribute** serves the dual purpose of marking a method with parameters as a test method and providing inline data to be used when invoking that method.

|  |
| --- |
| **[TestCase(-4, 9, Description ="TT, TF")]**  **[TestCase(-4.1, 9.2, Description = "TT")]**  **[TestCase(-4.0001, 9.0002, Description = "TT")]**  **[TestCase(-3.9, 9, Description = "TF")]**  **[TestCase(-3.999, 9, Description = "TF")]**  **[TestCase(4, 1, Description = "FT, FF")]**  **[TestCase(3.9, 1.2, Description = "FT")]**  **[TestCase(4.0001, 0.9998, Description = "FT")]**  **[TestCase(3.9, 1, Description = "FF")]**  **[TestCase(3.999, 1, Description = "FF")]**  public void DomainTestingTestCaseAttr(double x, double y)  {  Assert.That(true);  } |
|  |

**TestCaseSourceAttribute** is used on a parameterized test method to identify the property, method or field that will provide the required arguments. The attribute has two public constructors.

* TestCaseSourceAttribute(Type sourceType, string sourceName);
* TestCaseSourceAttribute(string sourceName);

If sourceType is specified, it represents the class that provides the test cases. It must have a default constructor.

If sourceType is not specified, the class containing the test method is used. NUnit will construct it using either the default constructor or - if arguments are provided - the appropriate constructor for those arguments.

The sourceName argument represents the name of the source used to provide test cases. It has the following characteristics:

* It may be a field, property or method.
* It may be either an instance or a static member.
* It must return an IEnumerable or a type that implements IEnumerable.
* The individual items returned by the enumerator must be compatible with the signature of the method on which the attribute appears.

|  |
| --- |
| **static double[][] testData =**  **{**  **new double[] {-4, 9},**  **new double[] {-4.1, 9.2},**  **new double[] {-4.0001, 9.0002},**  **new double[] {-3.9, 9},**  **new double[] {-3.999, 9},**  **new double[] {4, 1},**  **new double[] {3.9, 1.2},**  **new double[] {4.0001, 0.9998},**  **new double[] {3.9, 1},**  **new double[] { 3.999, 1}**  **};**  [Test, **TestCaseSource("testData")**]  public void DomainTestingTestCaseSrc(double x, double y)  {  Assert.That(true);  } |
|  |