Equivalence partitioning and boundary-value analysis

Mark Timmer

This document describes how equivalence partitioning and boundary-value analysis can be applied. Although a cookbook is hard to give, as creativity is also heavily involved, we at least provide basic rules for coming up with equivalence classes and determining boundaries of equivalence classes defined by intervals.

1 Equivalence partitioning

For equivalence partitioning, the possible values of the inputs are divided into appropriate equivalence classes. The equivalence partitioning should at least take into account the values each input is allowed to have. Also, if applicable it should partition based on relations between the inputs. Finally, when appropriate, we test for the number of inputs and their types.

Below, we consider each of these categories of equivalence classes.

Input ranges The values each input is allowed to have obviously depend on the specification. For instance, we can imagine an application with five inputs k, l, m, n, x, for which the following restrictions hold:

Condition	**Valid equiv. classes**	**Invalid equiv. classes**
Value of k	$k > 0$ (1)	$k \leq 0$ (2), $k >$ MAXINT(3)
Value of l	$l < 0$ (4), $l > 10$ (5)	$0 \leq l \leq 10$ (6), $l <$ MININT (7), $l >$ MAXINT (8)
Value of m	$0 \leq m \leq 10$ (9)	$m < 0$ (10), $m > 10$ (11)
Value of n	$3 < n < 8$ (12)	$n \leq 3$ (13), $n \geq 8$ (14)
Value of x	$0 \leq x < 100$ (15)	$x < 0$ (16), $x \geq 100$ (17)

Note that equivalence classes are often defined by intervals, either with open or closed bounds. Also note that we take into account the minimum and maximum values integers can have (similarly, for real-valued variables we should use MINFLOAT and MAXFLOAT). Although these values depend on the compiler and operation system that are used, they are relevant for testing purposes.

Also note that we often just write something like $k > 0$, while we actually mean $0 < k <$ MAXINT. Although this is the way often seen in practice, it couldn't hurt to explicitly also mention the upper bound of MAXINT and the lower bound of MININT for the valid equivalence classes, where applicable.

Input relations Sometimes, a program behaves differently based on a relation between inputs. For instance, we might have the following additional equivalence classes:

Condition	**Valid equiv. classes**	**Invalid equiv. classes**
Relation	$m < n$ (18)	$m \geq n$ (19)
Relation	$2 \cdot m + l \geq 13 \cdot k$ (20)	$2 \cdot m + l < 13 \cdot k$ (21)
Relation	$m \geq 5 \wedge n \geq 5$ (22), $m < 5 \vee n < 5$ (23)	

Number of inputs and types Finally, we need to take care of the amount of inputs and the types of these inputs. For instance:

Condition	Valid equiv. classes	Invalid equiv. classes
Nr. of inputs	5 (24)	< 5 (25), > 5 (26)
Types	int/int/int/int/float (27)	non-int/int/int/int/float (28)
		int/non-int/int/int/float (29)
		int/int/non-int/int/float (30)
		int/int/int/non-int/float (31)
		int/int/int/int/non-float (32)

Static type checking In case of static type checking (as for instance for a Java function), the compiler may already enforce some of the restrictions above. In that case, the equivalence classes related to the number of inputs and their types can be omitted. Also, the equivalence classes of the form $n <$ MININT and $n >$ MAXINT might not be needed.

However, when working with command line tools, or input from a textfield for instance, these kind of equivalence classes are relevant.

1.1 Testing based on equivalence classes

Once the equivalence classes have been found, it is not hard anymore to construct a test case based on them.

First, we test all the valid equivalence classes. Try to combine as many as possible, to keep the test suite as small as possible. Second, we test all the invalid equivalence classes. Here, only test one invalid equivalence class at the same time. After all, when inputs are contained in several invalid equivalence classes, we are still not sure whether input that is 'less invalid' is also handled correctly by the program.

2 Boundary-value analysis

The rules for extending a test suite based on boundary-value analysis are based on the equivalence classes: for each class, additional test cases are created. We categorise the rules based on the format of the equivalence class. First, we only consider equivalence classes specifying the range one specific input is supposed to be in.

In the remainder of this document we use the convention of representing arbitrary constants by $a, b, c, \ldots$, and real-valued (floating point) variables by v, w, x, y, z (exactly as in for instance the well-known formula $ax^2 + bx + c = 0$). Moreover, we use $k, l, m, n, \ldots$ to represent integer-valued variables.

Single input – Bounded interval Basically, for bounded intervals for a single input we test the lower and the upper bound.

In case the variable under consideration is an integer, this is easy: for the equivalence class $a \leq n \leq b$ we test $n = a$ and $n = b$. If one or both bounds are open (that is, n is not allowed to be equal to the bound), we first rewrite to a closed form: $a < n \leq b$ is easily rewritten to $a + 1 \leq n \leq b$, so $n = a + 1$ and $n = b$ are tested.

For floating point values, we still test $x = a$ and $x = b$ for the equivalence class $a \leq x \leq b$. However, for open bounds we cannot rewrite to a closed form anymore. Therefore, in that case we test the bound itself and a value on the valid side that is very close (let's call the distance ϵ) to the boundary. For instance, given the interval $a < x < b$ we test $x = a$, $x = b$, $x = a + 0.001$ and $x = b - 0.001$ (in this case having chosen $\epsilon = 0.001$).

Single input – Unbounded interval For equivalence classes for a single input defined by unbounded intervals, a distinction is made between valid and invalid classes.

A *valid* interval such a $n \leq a$ is interpreted as MININT $\leq n \leq a$. Therefore, we test $n =$ MININT and $n = a$. Similarly, $n < a$ is handled. For $x < a$ and $x \leq a$ we use MINFLOAT. Intervals with a $>$ or $\geq$ symbol are dealt with symmetrically, using MAXINT/MAXFLOAT instead of MININT/MINFLOAT.

For *invalid* intervals such a $x < a$ or $n > a$ we do not consider MINFLOAT or MAXINT. After all, these values are no boundary of the interval, as even smaller / larger values also are invalid.

Remark 1. *For programs reading from the command line or a text field, unbounded equivalence classes such as* $n < \text{MININT}$ *and* $n > \text{MAXINT}$*, or* $x < \text{MINFLOAT}$ *and* $x > \text{MAXFLOAT}$ *might be present. For these kind of intervals, we apply the rules as above. So, for* $n < \text{MININT}$ *we test* $n = \text{MININT} - 1$*, and for* $x > \text{MAXFLOAT}$ *we test* $x = \text{MAXFLOAT}$ *and* $x = \text{MAXFLOAT} + \epsilon$*.*

Relations between inputs If an equivalence class is given by a relation between inputs, we handle it in the same way as an invalid unbounded interval (no matter whether it indeed is valid or invalid). For instance, for the equivalence class $n < m$, relating two integers, we only test $n = m - 1$. For the equivalence class $x < y$, relating two floating points, we test $x = y$ and $x = y - \epsilon$ for some small ϵ.

The next page shows a summary of all the rules discussed to far.

More complicated equivalence classes For more complicated equivalence classes it is not that straight-forward to provide rigorous rules. Take for instance the equivalence class $\text{grade}_1 \geq 5 \wedge \text{grade}_2 \geq 5$. In this case, taking into account the considerations for the easier situations, it seems to make sense to test at least the following boundary situations:

- $\text{grade}_1 = 5 \quad \wedge\ \text{grade}_2 = 5$
- $\text{grade}_1 = 4.999 \ \wedge\ \text{grade}_2 = 5$
- $\text{grade}_1 = 5 \quad \wedge\ \text{grade}_2 = 4.999$

We will not provide rigorous rules for all possible situations, but ask you to extrapolate the existing rules in a sensible manner.

Boundaries of the output Finally, in addition to testing bounds for the equivalence classes of the inputs, for boundary-value analysis we test the boundaries of the output. So, if a program compares for instance the difference between two hours of a day, we provide inputs to test the minimal difference 0 and also the maximal difference 23.

3 Summary of boundary-value analysis

The following tables summarise the results for boundary-value analysis.

Bounded equivalence classes for a single input

Equivalence class	Test case 1	Test case 2	Test case 3	Test case 4
$a \leq n \leq b$	$n = a$	$n = b$		
$a < n \leq b$	$n = a + 1$	$n = b$		
$a < n < b$	$n = a + 1$	$n = b - 1$		
$a \leq n < b$	$n = a$	$n = b - 1$		
$a \leq x \leq b$	$x = a$	$x = b$		
$a < x \leq b$	$x = a$	$x = b$	$x = a + \epsilon$	
$a < x < b$	$x = a$	$x = b$	$x = a + \epsilon$	$x = b - \epsilon$
$a \leq x < b$	$x = a$	$x = b$		$x = b - \epsilon$

Unbounded equivalence classes for a single input

Equivalence class	Test case 1	Test case 2	Test case 3
$n < a$ (valid)	$n = a - 1$	$n =$ MININT	
$n \leq a$ (valid)	$n = a$	$n =$ MININT	
$n \geq a$ (valid)	$n = a$	$n =$ MAXINT	
$n > a$ (valid)	$n = a + 1$	$n =$ MAXINT	
$n < a$ (invalid)	$n = a - 1$		
$n \leq a$ (invalid)	$n = a$		
$n \geq a$ (invalid)	$n = a$		
$n > a$ (invalid)	$n = a + 1$		
$x < a$ (valid)	$x = a$	$x =$ MINFLOAT	$x = a - \epsilon$
$x \leq a$ (valid)	$x = a$	$x =$ MINFLOAT	
$x \geq a$ (valid)	$x = a$	$x =$ MAXFLOAT	
$x > a$ (valid)	$x = a$	$x =$ MAXFLOAT	$x = a + \epsilon$
$x < a$ (invalid)	$x = a$		$x = a - \epsilon$
$x \leq a$ (invalid)	$x = a$		
$x \geq a$ (invalid)	$x = a$		
$x > a$ (invalid)	$x = a$		$x = a + \epsilon$

(For the invalid classes, a can be MININT, MAXINT, MINFLOAT or MAXFLOAT.)

Relations between inputs

Equivalence class	Test case 1	Test case 2
$f(n) \leq f(m)$	$f(n) = f(m)$	
$f(n) < f(m)$	$f(n) = f(m) - 1$	
$f(n) > f(m)$	$f(n) = f(m) + 1$	
$f(n) \geq f(m)$	$f(n) = f(m)$	
$f(x) \leq f(y)$	$f(x) = f(y)$	
$f(x) < f(y)$	$f(x) = f(y)$	$f(x) = f(y) - \epsilon$
$f(x) > f(y)$	$f(x) = f(y)$	$f(x) = f(y) + \epsilon$
$f(x) \geq f(y)$	$f(x) = f(y)$	