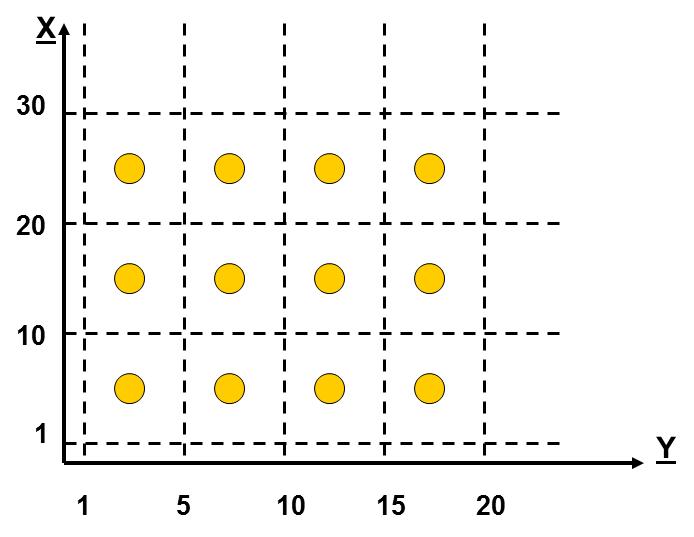
**Equivalence Class Testing**

**Introduction**

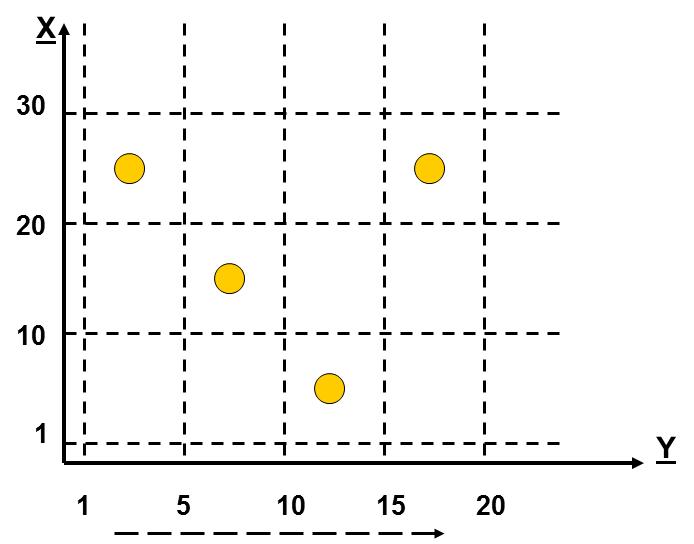
1. **Equivalence Class Testing (ECT) –** ECT is a good approach to black-box testing. It is an easier version of the technique we learned in CS 4321, Category-Partition testing. The general steps for ECT are:
2. Identify the input variables
3. Break the values for each variable into *equivalence classes.* All values in an equivalence class are *expected* to behave the same. Two input values are in the same equivalence class if they are treated in the same way according to the specification, *i.e.* they cause the same path to be executed.
4. Create the test specification by minimally making sure each equivalence class is covered. When possible we would consider every combination of equivalence classes.
5. Create the test cases.
6. **Example –** Consider a problem where we have two input variables, X and Y and the following equivalence classes result:

X: [1,10], [11,20], [21,30]

Y: [1,5], [6,10], [11,15], [16,20]

All combinations of equivalence classes results in 3\*4=12 tests. Obviously with such a small number of inputs, we could easily test all these, and would use some boundary values and error conditions also. In many situations it is not possible to consider all combinations.

A minimal ECT is one where at least one value is used from each class – each class is *covered.* Thus a minimal ECT could be:



|  |  |  |
| --- | --- | --- |
| Test | Specification | Test Case |
| 1 | X: [1,10], Y: [11,15] | X=6, Y=12 |
| 2 | X: [11,20], Y: [6,10] | X=14, Y=7 |
| 3 | X: [21,30], Y: [1,5] | X=25, Y=4 |
| 4 | X: [21,30], Y: [16,20] | X=26, Y=18 |

1. **Boundary Value Testing –** Test at the boundaries. **Off-by-one Testing** – include the values around boundaries.

|  |  |
| --- | --- |
| **X: [<1], [1,10], [11,20], [21,30], [>30]**  **Y: [<1], [1,5], [6,10], [11,15], [16,20], [>20]** | **X: [<1], [1,10], [11,20], [21,30], [>30]**  **Y: [<1], [1,5], [6,10], [11,15], [16,20], [>20]** |

1. **Example –** A health index that is computed depends on a person’s sex, age, and weight. The procedure is valid for males between 18 and 55 and females between 18 and 50; however, different procedures are used depending on the category of the age as shown in the table below. In all cases, the weight must be between 80 and 300 pounds.

|  |  |  |
| --- | --- | --- |
| Category | Male | Female |
| Young | 18-35 | 18-30 |
| Middle | 36-45 | 31-40 |
| Old | 46-55 | 41-50 |

|  |
| --- |
| **Equivalence Classes** |
| Sex: [male], [female]  Weight: [80, 300]  Age: [18,30], [31-35], [36-40], [41-45], [46-50], [51-55] |

|  |
| --- |
| **Minimal ECT** |
| |  |  |  |  | | --- | --- | --- | --- | | Test | Sex | Age | Weight | | 1 | Female | 20 | 100 | | 2 | Male | 32 | 200 | | 3 | Female | 38 | 90 | | 4 | Male | 43 | 160 | | 5 | Female | 48 | 150 | | 6 | Male | 53 | 250 | |

1. A system I developed was a small payroll management system. There, I identified 10 sets of equivalence classes.
2. Hours per day: (0,4), (4,8), (8,inf)
3. Hour per week: Regular (Reg) pay and overtime (OT) pay. (<=40 Reg, 0 OT), (<=40 Reg, >0 OT), (>40 Reg, 0 OT), (>40 Reg, >0 OT)
4. Cards Per Day: (0), (2), (4), (6,8,...)
5. Card Type: (R), (C), (O), (H), (V), (J), (B)
6. Production Code: (D), (I)
7. Lunch Time: (<30 min), (>=30 min)
8. Call Back: (work<4), (work>=4)
9. Date In/Out: (date in = date out), (date in < date out)
10. Number of employees: (1), (2 or more)
11. Accounting Method: (Local), (Corporate)

Total number of combinations of equivalence classes: 3\*4\*4\*7\*2\*2\*2\*2\*2\*2 = 21,504

1. In practice, the actual specification of the tests is harder than just supplying simple input values as in the example above. It often involves the careful construction of data files that can be considered as input to the system as well as painstakingly proving what the expected results are. Sometimes this involves developing another program to validate test cases, using a spread-sheet, or paper and pencil.
2. Sometimes we must design a system with testing in mind. For instance, if we are developing a game (with randomness) we don’t want to do all our testing by just playing the game! That would take forever and would be subject to the vagaries of randomness! So, for such a system, you might design a “back-door” so that the game could be played subject to an input file that specified moves). You should not design a system that requires a Gui to be able to test the system.