Aimee Valladares

CEN 4072

15 April 2022

# Grammar-based Testing Exercises

**Q1)** Consider the Following BNF:

*P ::= IDY | IYD | DIY | DYI | YID | YDI*

*I ::= “j” | “s”*

*D ::= “9” | “21”*

*Y ::= “0” | “4”*

1. How many nonterminal symbols are in the grammar?

There are 4 nonterminal symbols: P, I, D, and Y.

1. How many terminal symbols are in the grammar?

There are 6 terminal symbols: j, s, 9, 21, 0, and 4.

1. Write two strings that are valid according to the BNF?
2. P = IDY = j214
3. P = DIY = 9j0
4. For each of your two strings, give two valid mutants of the string
5. P = j214, two valid mutants are j210 and s214.
6. P = 9j4, two valid mutants are 9s4 and 9j0
7. For each of your two strings, give two invalid mutants of the string
8. P = j214, two invalid mutants are 21214 (DDY) and jj4 (IIY)
9. P = 9j4, two invalid mutants are 999 (DDD) and 9j49 (DIYD)

**Q2)**

/\*\*

\* Find last index of element

\*

\* @param numbers array to search

\* @param val value to look for

\* @return last index of val in numbers; -1 if absent

\* @throws NullPointerException if numbers is null

\*/

1. public static int findVal(int numbers[], int val)
2. {
3. int findVal = -1;
4. for (int i=0; i<numbers.length; i++)
5. // for (int i=(0+1); i<numbers.length; i++)
6. if (numbers [i] == val)
7. findVal = i;
8. return (findVal);
9. }
10. If possible, find test inputs that do not reach the mutant.

It is not possible to find the test input that does not reach the mutant because any input even if the array is empty will reach the mutant in the for loop. Additionally, it is unclear if the input numbers = null will not reach the mutant because we are not sure when the exception is thrown (It can be thrown after reaching the mutant).

1. If possible, find test inputs that satisfy reachability but not infection for the mutant.

It is not possible to satisfy reachability but not infection because when you reach the for loop you are at the infection and encounter the wrong internal state. Specifically, it is not possible to not encounter the wrong internal state when encountering the for loop, initializing I to 1 will always cause infection.

1. If possible, find test inputs that satisfy reachability and infection, but not propagation for the mutant.

The following are test inputs that satisfy reachability and infection, but not propagation:

* When the element the function is looking for at index 0 of the array position is followed by the same element.
  + For instance, numbers = (5, 5, 5) and val = 5
* When the element the function is looking for that is in any array index except for index 0 of the array position.
  + For instance, numbers = (2, 3, 5) and val = 5 or numbers = (2, 3, 5) and val = 3
* When the element the function is looking is not in the numbers array.
  + For instance, numbers = (0,0,0) and val = 6

1. If possible, find test inputs that strongly kill the mutants.

The test inputs that would strongly kill the mutants is having the element the function is looking for at index 0 of the array and that element is not repeated in other array indices.

* For instance, numbers = (2, 3, 5) and val = 2

Q3) Define 10 mutants for the following method power() using the effective mutation operators discussed in class. Use any mutation operator at most two times. Indicate which operator you are using and the original statement/line.

1. public static int cal (int month1, int day1, int month2, int day2, int year)
2. {
3. //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
4. // Calculate the number of Days between the two given days in the same year.
5. // Preconditions: day1 and day2 must be in same year
6. // 1 <= month1, month <=12
7. // 1 <= day1, day2 <= 31
8. // month1 <= month2
9. //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
10. int numDays;
11. if (month2 == month1) // in the same month
12. numDays = day2 – day1;
13. else
14. {
15. // Skip month 0
16. int daysIn[] = {0, 31, 0, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
17. // Are we in a leap year?
18. int m4 = year % 4;
19. int m100 = year % 100;
20. int m400 = year % 400;
21. if ((m4 !=0 ) || ((m100 == 0) && (m400 != 0)))
22. daysIn[2] = 28;
23. else
24. daysIn[2] = 29;
25. // start with days in the two months
26. numDays = day2 + (daysIn[month1] - day1);
28. // add the days in the intervening months
29. for (int i = month1 + 1; i <= month2 – 1; i++)
30. numDays = daysIn[i] + numDays;
31. }
32. return (numDays);
33. }

* ABS – Change line 12 to (numDays = **|**day2 – day1**|**)
* ABS – Change line 26 to (numDays = day2 + ( **|** daysIn[month1] - day1 **|** ))
* AOR - Change line 18 to m4 = year **+** 4.
* AOR - Change line 19 m100 = year **-** 100
* ROR – Change line 29 to for (int i = month1 + 1; i **<** month2 – 1; i++)
* COR – Change line 21 to if ((m4 !=0 ) **&&** ((m100 == 0) && (m400 != 0)))
* ASR – Change line 20 to m400 **+**= year % 400
* UOI – Change line 30 to **-**(numDays = daysIn[i] + numDays)
* SVR – Change line 26 to numDays = **day1** + (daysIn[month1] - day1)
* SVR – Change line 11 to month2 == **month2**

Q4) Write the predicate (only the predicate) to represent the requirement: \List all the wireless mice that either retail for more than $100 or for which the store has more than 20 items. Also list non-wireless mice that retail for more than $50." Write predicate (^ = and, v = or, ~ = not)

Q5)

1. Write the complete truth table for the predicate. Label your rows starting from 1. Row 1 should all clauses be true. You should include columns for the conditions under which each clause determines the predicate, and a column for the value of the predicate itself.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Row | a | b | c | P | Pa | Pb | Pc |
| 1 | T | T | T | T | \* |  | \* |
| 2 | T | T | F | F |  | \* | \* |
| 3 | T | F | T | T | \* |  |  |
| 4 | T | F | F | T | \* | \* |  |
| 5 | F | T | T | F | \* |  |  |
| 6 | F | T | F | F |  |  |  |
| 7 | F | F | T | F | \* |  |  |
| 8 | F | F | F | F | \* |  |  |

1. List all pairs of rows from your table that satisfy General Active Clause Coverage (GACC) with respect to each clause.

* When a is the active clause the rows (1,8) satisfy GACC.
* When b is the active clause the rows (2, 4) satisfy GACC.
* When c is the active clause the rows (1, 2) satisfy GACC.

1. List all pairs of rows from your table that satisfy Correlated Active Clause Coverage (CACC) with respect to each clause.

The pairs of rows that satisfy CACC is the same answer as GACC:

* When a is the active clause any combination of the following rows [(1, 3, 4) x (5, 7, 8)] are valid. For instance, the rows (1, 5) are valid
* When b is the active clause, rows (2, 4) are valid
* When c is the active clause, rows (1, 2) are valid

1. List all pairs of rows from your table that satisfy Restricted Active Clause Coverage (RACC) with respect to each clause.

* When a is the active clause the rows (1, 5), (3, 7), and (4, 8) satisfy RACC
* When b is the active clause the rows (2, 4) satisfy RACC
* When c is the active clause the rows (1, 2) satisfy RACC

Q6) Answer the following questions for the method twoPred() below:

1. public String twoPred (int x, int y)
2. {
3. boolean z;
4. if (x < y)
5. z = true;
6. else
7. z = false;
9. if (z && x+y == 10)
10. return “A”;
11. else
12. return “B”;
13. }

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Row | x < y (a) | x+y == 10 (b) | P | Pa | Pb |
| 1 | T | T | T | \* | \* |
| 2 | T | F | F |  | \* |
| 3 | F | T | F | \* |  |
| 4 | F | F | F |  |  |

1. List test inputs for twoPred() that achieve Restricted Active Clause Coverage (RACC).

The predicates that satisfy and achieve RACC when a is the active clause are rows (1, 3). Meanwhile when b is the active clause the rows that achieve RACC are rows (1, 2).

The test inputs would be:

* + twoPred(4, 6)
  + twoPred(4, 5)
  + twoPred(6, 4)

1. List test inputs for twoPred() that achieve Restricted Inactive Clause Coverage (RICC).

It is not feasible to utilize rows that satisfy the RICC requirements when the predicate is true so the inputs will primarily focus on inputs that achieve RICC when the predicate is false.

The test inputs would be:

* + twoPred(3, 6)
  + twoPred(9, 1)
  + twoPred(8, 1)