

Assignment 2: Concolic Testing

Milan Lagae

Institution/University Name:

Faculty:

Course:

Vrije Universiteit Brussel
Sciences and Bioengineering Sciences
Software Quality Analysis

Contents

1	Intro	1
2	Discussion Point 1	2
2.1	Legend	2
2.2	Iteration 1	2
2.3	Iteration 2	2
3	Discussion Point 2	3
4	Discussion Point 3	3
4.1	Bread-First Search Priority	3
4.2	Random Priority	3
5	Discussion Point 4	3
5.1	Overview	3
6	Appendix	4
6.1	Discussion Point 1	4
6.2	Discussion Point 2	4
6.3	Discussion Point 3	4
6.4	Discussion Point 4	4
	Bibliography	5

1 Intro

All complete code can be found in the Appendix section §6.

2 Discussion Point 1

2.1 Legend

TODO Add table symbol explanations.

2.2 Iteration 1

Code	Concrete Store	Symbolic Store	Path Conditions
main () {			
var x , y , z ;			
x = input ;	[x -> 0] (input = 0)	[x -> a]	
z = input ;	[x -> 0, z -> 0] (input = 0)	[x -> a, z -> b]	
y = &x ;	[x -> 0, z -> 0, y -> x]	[x -> a, z -> b, y -> x]	
if (x > 0) {	[x -> 0, z -> 0, y -> x]	[x -> a, z -> b, y -> x]	[!(a > 0)]
y = &z ;	/	/	
} else {			
*y = input ;	[x -> 0, z -> 0, y -> x] (input = 0)	[x -> a, z -> b, y -> x]	
}			
*y = *y + 7 ;	[x -> 7, z -> 0, y -> x] (7 + 0)	[x -> a, z -> b, y -> x]	
if (2 > z) {			[!(a > 0) and !(2 > (b + 7))]
if (*y == 2647) {	/	/	/
error 1 ;	/	/	/
}	/	/	/
}	/	/	/
return *y ;	7		
}	-	-	[!(a > 0) and !(2 > (b + 7))]

2.3 Iteration 2

3 Discussion Point 2

4 Discussion Point 3

As described in the assignment, two priorities were implemented. Both strategies were implemented in the `ConcolicEngine.scala` file.

4.1 Bread-First Search Priority

The first priority based BFS strategy is implemented by the method: `nextExplorationTargetBFS`. The search starts from the the `root` of the execution tree. The complete method implementation for the BFS Strategy can be found in **ADD-REFERENCE**

The first step is creating a `mutable.PriorityQueue`, by default the queue retrieves the elements with the highest priority, the algorithm requires the element with the lowest priority (shortest distance from root), for this the ordering is set reversed on queue creation.

After the empty queue is created, the children of the root are iterated and for each a tuple, is created containing the child node: `(child, 1)`, with the initial distance set to 1.

The next step in the algorithm is to iterate the queue until it is empty. On entering the queue, the first element highest priority/shortest distance from root is removed.

Using the below match case statement from the existing `nextExplorationTarget` method, the `true` & `false` branches are checked for the status of amount of explored branches.

ADD LISTING

If the count of explored branches of either value is 0, the existing node is returned. In case both branches have already been explored, the `None` value is returned.

The final step in the algorithm, is the `if` case, which can be found in the listing below: **REFERENCE-LISTING**.

ADD LISTING

If there is a value defined the chosen node is used as the `nextTarget`. If no value is defined, all children of the node are added to the queue, with the distance increased by one: `distance + 1`.

4.2 Random Priority

The second search strategy is based on assigning a random priority to the node, the implementation method can be found in method: `nextExplorationTargetRNDM` and in listing: **ADD**.

The only change required for this strategy is importing the `scala.util.Random` package. When populating the queue with the initial values of the `root`, use the `rand.nextInt(Int.MaxValue)` to assign a random value.

The method will proceed as before, the only remaining difference is the case when the node has already been explored. When appending the children of the node, instead of increment the distance value as before, a random value is assigned with the function described as before.

5 Discussion Point 4

This section will discuss the implementation of the different search strategies and the effect on the number of runs & resulting errors cases.

5.1 Overview

The results of the concolic testing can be summarized and are visible in **LISTING**

Strategy	Runs	Failures
DFS	21	ADD
BFS	21	ADD
Random	21	ADD

Table 1: Concolic Testing - Search Strategies

6 Appendix

6.1 Discussion Point 1

6.2 Discussion Point 2

6.3 Discussion Point 3

6.4 Discussion Point 4

Bibliography