Piacente Cristian 866020

Assignment A5 – Dynamic Symbolic Execution

Software Quality, Academic Year 2023-2024, University of Milan – Bicocca

Identify the path of getFraction that gets executed when the three inputs are all equal to 1, and build with symbolic execution the path condition that characterises the path.

Then, working in the style of dynamic symbolic execution, identify the path conditions of the alternative subpaths, considering only the subpaths that do not lead to exceptions, and use these path conditions to identify test cases to execute those subpaths.

```
1 public static Fraction getFraction(int pwhole, int pnumerator, int pdenominator) {
2   int whole = pwhole;
3   int numerator = pnumerator;
4   int denominator = pdenominator;
5   if (denominator <= 0 || numerator < 0) {
6     throw new ArithmeticException("The denominator must be positive, "+
       "and the numerator must be non-negative");
    }
7   long numeratorValue;
8   if (whole < 0) {
9     numeratorValue = whole * (long)denominator - numerator;
    } else {
10    numeratorValue = whole * (long)denominator + numerator;
    }
11 if (numeratorValue < Integer.MIN_VALUE || numeratorValue > Integer.MAX_VALUE)  {
12    throw new ArithmeticException("Numerator: " + numeratorValue +
       " too large to represent as an Integer.");
    }
13 return new Fraction((int) numeratorValue, denominator);
  }
```

Notation:

- describe the identified paths and subpaths as a sequence of statement number, referring to the numbering of the code (e.g., 1 2 3 4 5 6 indicates the (sub) path from the beginning of the program to the execution of the then branch of the first if statement in the program)
- describe the path conditions by using the following symbols for the symbolic values of the parameters: PW as initial symbolic value of pwhole, PN as initial symbolic value of pnumerator, PD as initial symbolic value of pdenominator

If all the three input parameters equal to 1, the following path gets executed: **1 2 3 4 5 7 8 10 11 13**.

To compute the path condition, we can symbolically execute the path:

After 1: pwhole=PW, pnumerator=PN, pdenominator=PD, PC=true
After 2: pwhole=PW, pnumerator=PN, pdenominator=PD, whole=PW, PC=true
After 3: pwhole=PW, pnumerator=PN, pdenominator=PD, whole=PW, numerator=PN, PC=true
After 4: pwhole=PW, pnumerator=PN, pdenominator=PD, whole=PW, numerator=PN, denominator=PD, PC=true
After 5 (jump to 7): pwhole=PW, pnumerator=PN, pdenominator=PD, whole=PW, numerator=PN, denominator=PD, PC=PD > 0 && PN >= 0
After 7: pwhole=PW, pnumerator=PN, pdenominator=PD, whole=PW, numerator=PN, denominator=PD, numeratorValue=0L, PC=PD > 0 && PN >= 0
After 8 (jump to 10): pwhole=PW, pnumerator=PN, pdenominator=PD, whole=PW, numerator=PN, denominator=PD, numeratorValue=0L, PC=PD > 0 && PN >= 0 && PW >= 0
After 10: pwhole=PW, pnumerator=PN, pdenominator=PD, whole=PW, numerator=PN, denominator=PD, numeratorValue=PW * (long)PD + PN, PC=PD > 0 && PN >= 0 && PW >= 0
After 11 (jump to 13): pwhole=PW, pnumerator=PN, pdenominator=PD, whole=PW, numerator=PN, denominator=PD, numeratorValue=PW * (long)PD + PN, PC=PD > 0 && PN >= 0 && PW >= 0 && PW * (long)PD + PN >= Integer.MIN_VALUE && PW * (long)PD + PN <= Integer.MAX_VALUE

For simplicity, we will **omit the long cast** for our analysis and also we will **replace the constants** Integer.MIN_VALUE and Integer.MAX_VALUE with their values, obtaining the following path condition:

PD > 0 && PN >= 0 && PW >= 0 && PW * PD + PN >= -2147483648 && PW * PD + PN <= 2147483647

Piacente Cristian 866020

Assignment A5 – Dynamic Symbolic Execution

Software Quality, Academic Year 2023-2024, University of Milan – Bicocca

Now that we have built the path condition from executing the initial test case, we can generate the alternative path conditions by negating the conditions singularly (we'll exclude those which lead to throwing an exception):

- ~~PD <= 0~~
  denominator must be positive
- ~~PD > 0 && PN < 0~~
  numerator must be non-negative
- PD > 0 && PN >= 0 && PW < 0
- ~~PD > 0 && PN >= 0 && PW >= 0 && PW * PD + PN < -2147483648~~
  numerator too large to represent as an integer
- ~~PD > 0 && PN >= 0 && PW >= 0 && PW * PD + PN >= -2147483648 && PW * PD + PN > 2147483647~~
  numerator too large to represent as an integer

We only have one possible alternative path condition that doesn't lead to generating an exception.

We can delve into the analysis of the alternative path condition:

- **PD > 0 && PN >= 0 && PW < 0**
  It is a satisfiable alternative (sub)path condition, so a constraint solver would find a possible combination of values for the parameters, e.g. PD=2, PN=1, PW=-1.
  A possible test case is: **getFraction(-1, 1, 2)**.
  We can perform symbolic execution to retrieve the full path condition:

  After 1: pwhole=PW, pnumerator=PN, pdenominator=PD, PC=true
  After 2: pwhole=PW, pnumerator=PN, pdenominator=PD, whole=PW, PC=true
  After 3: pwhole=PW, pnumerator=PN, pdenominator=PD, whole=PW, numerator=PN, PC=true
  After 4: pwhole=PW, pnumerator=PN, pdenominator=PD, whole=PW, numerator=PN, denominator=PD, PC=true
  After 5 (jump to 7): pwhole=PW, pnumerator=PN, pdenominator=PD, whole=PW, numerator=PN, denominator=PD, PC=PD > 0 && PN >= 0
  After 7: pwhole=PW, pnumerator=PN, pdenominator=PD, whole=PW, numerator=PN, denominator=PD, numeratorValue=0L, PC=PD > 0 && PN >= 0
  After 8 (jump to 9): pwhole=PW, pnumerator=PN, pdenominator=PD, whole=PW, numerator=PN, denominator=PD, numeratorValue=0L, PC=PD > 0 && PN >= 0 && PW < 0
  After 9: pwhole=PW, pnumerator=PN, pdenominator=PD, whole=PW, numerator=PN, denominator=PD, numeratorValue=PW * (long)PD - PN, PC=PD > 0 && PN >= 0 && PW < 0
  After 11 (jump to 13): pwhole=PW, pnumerator=PN, pdenominator=PD, whole=PW, numerator=PN, denominator=PD, numeratorValue=PW * (long)PD - PN, PC=PD > 0 && PN >= 0 && PW < 0 && PW * (long)PD - PN >= Integer.MIN_VALUE && PW * (long)PD - PN <= Integer.MAX_VALUE

  This means that the path is **1 2 3 4 5 7 8 9 11 13**; the (simplified) full path condition, retrieved after instrumenting and running the program with the test case shown before, is the following:

  PD > 0 && PN >= 0 && PW < 0 && PW * PD - PN >= -2147483648 && PW * PD - PN <= 2147483647

This concludes the dynamic symbolic execution for this exercise.

EXTRA
A tool based on the concolic execution technique wouldn't know in advance if an exception is raised for an alternative path condition. If the constraint solver can find a combination of values which solve the alternative path condition, then the tool would generate and run a test case with the values returned by the constraint solver, and it would find out about the exception only at run-time.
After retrieving a full path condition from a test case which didn't generate an exception, the tool would continue the DSE by generating new alternative (sub)path conditions and it would skip the redundant ones.