Flow Control

An Overflow Detection Addition to the Checker Framework

Kenji Nicholson (kenjilee)
Lauren Martini (lmartini)

CSE 403, Spring 2018

Problem and Motivation

- The Checker framework has an index checker
  - **BUT, does not account for possible integer overflow**

For example:

```
int size = Integer.MAX_VALUE;
makeArray(size);

public static void makeArray( @Positive(size) ) {
        size++;
        int[ ] array = new int[size];
}
```

An undetected overflow error occurs here. The index checker still believed that the @Positive tag applies to “size”, even though “size” will now have a negative value.

- Due to the high frequency of overflow errors, this problem is significant

Approach

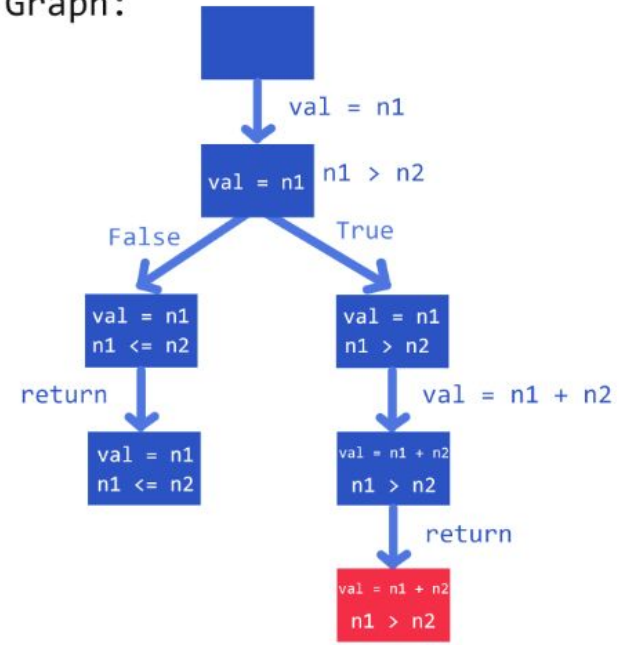
- Static Checker
- Create a graph of program states and look for states susceptible to overflow
- Define new tags or modify existing ones to help classify states into overflow cases
- Current approaches have high false-positive rates or are unsound/dynamic
    - False-positives created by things like:
        - Missing input constraints
        - Lack of global information
        - Imprecise symbolic execution

Example of program states graph with illegal state detection

Code:

```
public int sumIfFirstGreater(int n1, int n2, int val) {
    val = n1;
    if(n1 > n2) {
        val = n1 + n2;
    }
    return val;
}
```

Graph:



Challenges

- Reducing false positives
  - Capturing as many input constraints as possible
  - Being able to differentiate between safe and unsafe inputs
  - Correctly and robustly translates code into a program states graph

Risks

- Should not interfere with the existing index checker
  - Mitigate this by creating new tags and following pre-existing methods in the index checker.