Automated Testing and Verification

Project 2 – The Soot data flow analysis framework

Deadline: 20.12.2012

Part 1 – Running Soot

You will find all resources needed for this project in `https://www.st.cs.uni-saarland.de/edu/automatedtestingverification12/projects/project2.tar.gz`. This archive file contains:

- The project description (this document)
- The source-code for the `CoffeeMaker` project

(6 points) Exercise 1

Given the following Java program

```
#1: int a=5;
#2: int c=1;
#3: while (!(c>a)) {
#4:       c = c+c;
#5: }
#6: a = c-a;
#7: c=0;
```

Run a Soot analysis using the *Reaching Defs Tagger* and answer:

a) **(2 points)** Which definitions of variables `a` and `c` reach line #3?
b) **(2 points)** Which definitions of variables `a` and `c` reach line #6?
c) **(2 points)** Which definitions of variable `c` reach line #4?

(6 points) Exercise 2

Given the following Java method

```
public int exercise2(int a, int b) {
#1: int c = a+b;
#2: int d = a-b;
#3: int r;
#4: if (a<b) {
#5:       r=c;
#6: } else {
#7:       r=d;
#8: }
#9: return r;
}
```

Run a Soot analysis using the *Live Variables Tagger* and answer:

a) **(2 points)** What is the set of live variables at line #5?
b) **(2 points)** What is the set of live variables at line #7?
c) **(2 points)** What is the set of live variables at line #9?

(3 points) Exercise 3

Given the following Java code:

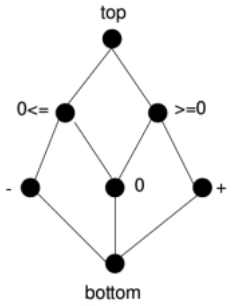
```
private static class Cell {
        int value;
}

public int exercise4(Cell c1, Cell c2) {
#1: c1.value =1;
#2: c2. value=2;
#3: return c1.value;
}
```

Run a Soot analysis using the *Null Pointer checker*. What abstract values for variables `c1` and `c2` may reach line #3?

Part 2 – Extending Soot

(40 points) Exercise 4

Implement a forward may data flow analysis for approximating if a given variable is zero, positive or negative. The abstract values for variables should be members of the following lattice:



The transfer function should handle at least the following operations:

- `x = constant;`
- `x = y;`
- `x = y+z;`
- `x = y-z;`
- `x = y*z;`

Given the following Java methods:

```
public int exercise4_1(int m, int n) {
#1: int x=0;
#2: int j = m/(x*n);
#3: return j;
}

public int exercise4_2(int m, int n) {
#1: int x = n-n;
#2: int i = x+1;
#3: int j=m/x;
#4: return j;
}

public int exercise4_3(int m, int n) {
```

```
#1: int x=0;
#2: if (m!=0)
#3:     x=m;
#4: else
#5:     x=1;
#6: int j = n/x;
#7: return j;
}

public int exercise4_4(int m, int n) {
#1: int x=0;
#2: int j=m/n;
#3: return j;
}
```

For each statement where a division is performed, print out the $in(n)$ set **Grading:** 10 points for printing out the correct $in(n)$ result for th analysis.

(45 points) Exercise 5

The goal of this exercise is to implement a dataflow analysis to compute the test obligations of the *all DU pairs* dataflow testing adequacy criterion, and then to design and implement a test suite satisfying this criterion.

The program under test is `CoffeeMaker`, a simple Java program that simulates a coffee machine.

Unzip CoffeeMaker.zip and compile the three Java classes under test (`Recipe.java`, `CoffeeMaker.java` and `Inventory.java`).

1. **(20 points)** Implement an intraprocedural analysis with Soot to compute the set of definition-use pairs in the three classes under test. For each method in each class compute the set of definition-use pairs and print the result of the analysis in the following format:

   ```
   --''METHODSIGNATURE''--
   ** Found ''N'' DU pairs **

   1) ''VARIABLENAME'' - DEF at ''LINENUMBER'', USE at ''LINENUMBER''
   2) ''VARIABLENAME'' - DEF at ''LINENUMBER'', USE at ''LINENUMBER''
   ...

   --''METHODSIGNATURE''--
   ** Found ''N'' DU pairs **

   3) ''VARIABLENAME'' - DEF at ''LINENUMBER'', USE at ''LINENUMBER''
   4) ''VARIABLENAME'' - DEF at ''LINENUMBER'', USE at ''LINENUMBER''

   ...
   ```

2. **(5 points)** Identify infeasible definition-use pairs, if there is any.

3. **(20 points)** Write a JUnit test suite that covers all the feasible definition-use pairs that your analysis identified. For each test case that you produce report which definition-use pairs it covers by reporting the definition-use ids in the comment of each test case.

Handout format

This project should be delivered before or during the handout date written at the very beginning of this document.

An email should be sent to the staff email (`atv12@st.cs.uni-saarland.de`) with the following material:

1. A file `src.zip` with the project source code. Code must be fully commented.
2. A file `readme.txt` with instructions on how to execute the delivered project.
3. A file `report.pdf` with a description of the resolution of all exercises, including a brief discussion on the most important design decision taken during the project.
4. A file `id.txt` containing the full names and matriculation numbers of all group members.

The e-mail subject should be:

```
[ATV-project2] name1 (matriculation1) / name2 (matriculation2)
```

where name1 and name2 should be lexically ordered. No printed material will be accepted.