Prof. Dr. Oscar Nierstrasz
Pascal Gadient, Pooja Rani

Assignment 09 — 11.11.2020 – v1.2
Static Program Analysis

Please submit this exercise by email to pascal.gadient@inf.unibe.ch before 18 November 2020, 10:15am.

You must submit your code as editable text, i.e., use plain text file(s).

For the second part of this exercise, we use the latest GT release from here. If you already imported some Moose (MSE) models from the previous assignments into your GT environment, please start with a fresh copy of GT to reduce the strains on your computer's random access memory.

First, we have to download and extract the *Weka* dataset, and second, we need to import it into GT. We can perform both tasks using GT's Playground. Be warned: this process will take several minutes depending on your device's CPU and internet connection. We strongly advise you to save the image when the process succeeded to avoid redoing these steps.

The datasets can be downloaded and extracted with the following script:

```
targetFolder := (FileLocator imageDirectory asFileReference / 'models')
ensureCreateDirectory.
archiveFileName := 'weka-3-8.zip'.
archiveUrl := 'https://dl.feenk.com/moose-tutorial/weka/'.
ZnClient new
  url: archiveUrl, archiveFileName;
  signalProgress: true;
  downloadTo: targetFolder.
(ZipArchive new
  readFrom: targetFolder / archiveFileName)
  extractAllTo: targetFolder.
```

The sample dataset can be imported with the following script:

```
modelFile := (FileLocator imageDirectory asFileReference / 'models')
  / 'weka-3-8'
  / 'weka-3-8.mse'.
modelWeka := MooseModel new
  importMSEFromFile: modelFile.
```

Exercise 1: Theory (3.5 pts)

a) What is the difference between static and dynamic code analysis?

b) Suppose you want to analyze the code that a method downloads arbitrarily from the internet. Can you perform such an inspection with *static analyses*? If yes, how, if not, why not?

c) Suppose you want to analyze the code that a method downloads arbitrarily from the internet. Can you perform such an inspection with *dynamic analyses*? If yes, how, if not, why not?

d) Choose a static analysis scenario where it is crucial to have no type 1 errors (false positives), but type 2 errors (false negatives) can be accepted. Explain for your scenario why you can accept type 2 errors, but no type 1 errors.

e) Choose a statically-typed language, and briefly describe what makes it statically-typed.

f) Choose a dynamically-typed language, and briefly describe what makes it dynamically-typed.

g) Is the collection of variable name declarations in the method body of Java's `String.println(String s)` intraprocedural or interprocedural? Explain why.

Exercise 2: Control flow graphs (6.5 pts)

a) Draw a CFG by hand (or with a flow chart tool) for the following code block: (1.5 pts)

```
int a = 2;
int b = 3;
if (a == 2) {
  b = a + b;
} else {
  b = a++;
}
return a + b;
```

b) Draw a CFG by hand (or with a flow chart tool) for the following code block: (2 pts)

```
public boolean addBalances(int money) {
  if (money > 0) {
    acquire(lock);
    int newMoneyValue = this.money + money;
    if (newMoneyValue < MAX_MONEY) {
      this.money = newMoneyValue;
      release(lock);
      return true;
    } else {
      release(lock);
      return false;
    }
  } else {
    return false;
  }
}
```

c) What is the cyclomatic complexity of both code blocks, i.e., from task a) and task b)? You should use the formula from A08. (1 pt)

d) Create the interval CFG for both code blocks, i.e., from task a) and task b). You can find an example CFG at the bottom of slide 20 (page 32) in the slide deck of lecture 09 which is available here. (2 pts)

Exercise 3: Template methods (6 pts BONUS)

Find all template methods in `modelWeka` with the help of `#gtASTNodes`, and plot them with `GtMondrian`. Please follow the steps below, one after another. The resulting plot should look similar to Figure 1.

NB: Template methods are abstract methods (in abstract classes).

Step 1: Select all methods that are in the namespace scope `weka::core`.

Step 2: Of those methods, find those that are abstract. Use `gtASTNode` for the AST traversal.

In this step, GT will build the AST of the relevant entities. This requires several minutes and around 6 GB of RAM on your PC. If you have less RAM the computation will slowdown massively.

Step 3: Visualize the found methods in *Step 2* with `GtMondrian`. Use the following parameters:

- shape type: `BlElement` with a size that represents `#children` of each method
- shape geometry: `BlCircle`
- shape background: all methods that have more than three elements in `children` should be in red, the others in gray

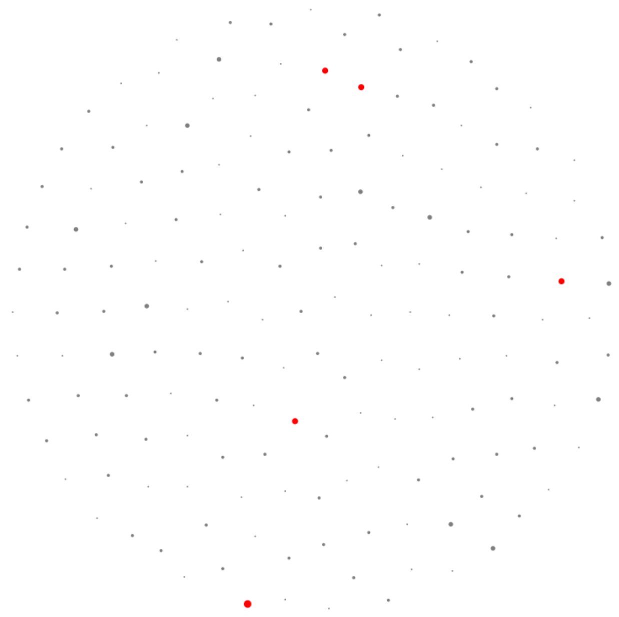

Figure 1: The resulting GtMondrian visualization built with GT