Applied Static Analysis

Software Technology Group
Department of Computer Science
Technische Universität Darmstadt
Dr. Michael Eichberg

Monotone Framework

You should use `MyOPALProject` as a template. That project is preconfigured to use the latest snapshot version of OPAL. You can clone the project using:
`git clone --depth 1 https://bitbucket.org/OPAL-Project/myopalproject Project`

> ⚠️ Always ensure that you use the latest snapshot version. You can clean the latest (snapshot) version that you have downloaded using the command `sbt cleanCache cleanLocal` in your project's root folder.

An integrated JavaDoc of the latest snapshot version of OPAL that spans all sub-projects can be found at: www.opal-project.de

For further details regarding the development of static analysis using OPAL see the OPAL tutorial.

You should develop the following analysis on top of the 3-address code representation (TACAI) offered by OPAL. Use the `l1.DefaultDomainWithCFGAndDefUse` domain and the `ProjectInformationKey` `ComputeTACAIKey` as the foundation for your analysis.

Closeables

Develop an **intraprocedural** analysis using the monotone framework which finds violations of the following rule taken from The CERT Oracle Secure Coding Standard for Java:

> FIO04-J: Close resources when they are no longer needed.

Here, we consider as a resource every instance of the class `java.lang.AutoCloseable`.

Non-compliant example:

```
public int processFile(String fileName) throws Exception {
  FileInputStream stream = new FileInputStream(fileName);
  BufferedReader bufRead = new BufferedReader(new InputStreamReader(stream));
  String line;
  while ((line = bufRead.readLine()) != null) {
    sendLine(line);
  }
  return 1;
}
```

Compliant example:

```
try (FileInputStream stream = new FileInputStream(fileName);
     InputStreamReader reader = new InputStreamReader(stream);
     BufferedReader bufRead = new BufferedReader(reader)) {
  String line;
  while ((line = bufRead.readLine()) != null) {
    sendLine(line);
  }
} catch (IOException e) {
```

```
    log(e);
}
```

> Do not forget that this analysis requires that the class hierarchy is reasonably complete. Hence, you should load the JDK (at least) as a library: `run -cp=<Path>/Closeables.class -libcp=<PATH>/jdk1.8.0_191.jdk/Contents/Home/jre/lib/rt.jar`

To reduce the number of false positives, only apply this check if the resource object is created inside the same method and ...:

- not passed to some other method, not returned and not stored in a field (i.e., only apply this check if the resource object is subject to garbage collection at the end of the method), or
- the resource is closed at least on one path.

For example, ignore the following cases where the stream is passed in as a parameter:

```
public int processFile(FileInputStream stream) throws Exception {
  BufferedReader bufRead = new BufferedReader(new InputStreamReader(stream));
  String line;
  while ((line = bufRead.readLine()) != null) {
    sendLine(line);
  }
  return 1;
}
```

Tasks

1. Test your analysis using the class `Closeables`. It should not produce any false positives.
2. Run your analysis against the JDK.