J. Hoenicke
J. Christ

20.11.2012
Hand in solutions via email to
`christj@informatik.uni-freiburg.de`
until 27.11.2012 (only Java sources, JPF configuration files, and PDFs accepted).
Paper submissions possible after the lecture.

Tutorials for "Formal methods for Java"
Exercise sheet 5

Exercise 1: Installing JPF

Install a Mercurial client for your operating system. Get a copy of the JPF repository either with

```
hg clone http://babelfish.arc.nasa.gov/hg/jpf/jpf-core
```

or a similar command from your Mercurial client.
Compile the downloaded version, e.g., using ant from the cloned repository:

```
bin/ant
```

Instructions for Eclipse or NetBeans can be found on the JPF wiki at `http://babelfish.arc.nasa.gov/trac/jpf/wiki`
You don't have to submit anything for this exercise.

Exercise 2: Configuring JPF

Create the directory `.jpf` in your home directory. Inside this directory create the main configuration file `site.properties` containing only the lines

```
jpf.home=<where you cloned jpf-core to>
jpf-core=${jpf.home}/jpf-core

extensions=${jpf-core}
```

where the `jpf.home` is set appropriately.
You don't have to submit anything for this exercise.

Exercise 3: Create a JPF Project
Get a copy of the jpf-template project, e.g., with

```
hg clone http://babelfish.arc.nasa.gov/hg/jpf/jpf-template
```

and compile it.
Create a new JPF project with the following command line

```
<path-to-jpf-template>/bin/create_project <Project-Name>
```

where the parts within < and > are set appropriately.
This step creates a new folder containing your project. The folder contains a `bin` directory that you can use to run JPF, a `src` directory with many subdirectories that you can use for development and examples, and an ant build script to build your project.
You don't have to submit anything for this exercise.

Exercise 4: Writing a listener
Create a subfolder `src/main/exercises` in your new JPF project. Download the file `UsageChecker.java` from the website of the lecture and place it in this folder. This file contains a partially implemented listener to check a certain usage pattern. For a pair of functions f and g, at any point during execution, more calls to f than calls to g should be made, i. e., the sequence *ffg* is allowed, but the sequence *fgg* is not. The functions f and g should be configurable via the options `uc.up` for f and `uc.down` for g. Furthermore, the Boolean option `uc.rec` should be used to configure, if multiple calls to f are allowed before a call to g, i. e., if `uc.rec` is set to `false`, only sequences of the form $(fg)^*(f|\varepsilon)$ are allowed.
Your task is to implement this listener. To account for the backtracking search, you can annotate the `ElementInfo` with the methods `addObjectAttr(Object)`, `getObjectAttr(Class)`, and `replaceObjectAttr(Object old, Object new)` of class `ElementInfo`. Note that annotation objects need to be immutable since backtracking will not restore the value of annotations.
Write a small test program and a JPF configuration to test your listener. You will need two functions and a main method.
Hand in the implementation of the listener, the test class, and the JPF configuration.