Program Monitoring

Lecture 5 :

Parameterized State Machines and Regular Expressions

Monday May 18, 2009

This fifth lecture introduces parameterized state machines and regular expressions and how they are supported by the JavaMOP system. After this lecture you should be able to write state machines and extended regular expression specifications in JavaMOP, and generate aspects from these, which can then be compiled together with your application program. This should make it easier to express properties than using only AspectJ.

Installation

JavaMOP is operated purely via the following website:

`http://fsl.cs.uiuc.edu/index.php/Special:JavaMOP2.0`

The idea is that state machine/regular expression specs are entered on this website, and aspects are generated, which can then be copied and pasted into an aspect in your application.

In order to run the aspects generated by JavaMOP, you need to make the apache commons collections package visible to your application:

`http://commons.apache.org/collections`

Reading

Same as lecture 4. Note, however, that most can be gained from playing around with the website.

Assignment 2

To be submitted to `havelund@gmail.com` before Monday May 25, at 11:59 pm.

Consider the Java API `java.net.URLConnection` documented as part of the Java API:

`http://java.sun.com/j2se/1.5.0/docs/api/`

This is a package for accessing URLs. The following example program accesses the course website, using the `URLConnection` package:

```
package exercise2;

import java.io.*;
import java.net.*;

public class Test {
  public static void main(String[] args) {
    try {
      URL url =
        new URL("http://www.runtime-verification.org/course09");
      URLConnection connection = url.openConnection();
      InputStream is = connection.getInputStream();
      System.out.println("begin");
      int input;
      do {
        input = is.read();
        if (input != -1) {
          System.out.print((char)input);
        }
      } while (input != -1);
      System.out.println("end");
      connection.setAllowUserInteraction(false);
      // is.close() -- we are lazy
    } catch(Exception e) {}
  }
}
```

The main method first creates a URL object, then opens a connection on that URL, gets the input stream and reads from it, printing it to standard output, character by character. The program contains two errors:

1. it calls the `URLConnection.setAllowUserInteraction(boolean)` method after the `getInputStream()` has been called (not allowed, see policy below).

2. the inputstream is not closed before program termination.

The documentation of the API requests a policy (not very clearly stated) for using a URLConnection, specifically referring to the first kind of error. Using JavaMOP, write two specifications using either regular expressions (JavaERE) or state machines (JavaFSM) that check the following two policies:

1. *it is not allowed to call a* `set`*-method on a URLConnection object after a* `get`*-method has been called. In this case the* `get` *method is* `getInputStream()`.

2. *an input stream that has been retrieved from a URLConnection object using the* `getInputStream()` *method should be closed before the main program terminates.*

Ignore the situation where the input stream is used to create some other input object. Assume further that the program is single-threaded, hence the only thread is the main program executing the `main` method, as in the test program. This means that the program terminates **after** the program executes the `Main.main(..)` method. The execution of this method is caught by the following pointcut:

```
execution(static void Test.main(..))
```