For this assignment, you will get experience using race-free Java (aka rccjava), a typechecker that checks for race conditions in Java programs.

Written Exercises

1. In the RACEFREEJAVA formalism, there is no mention of Java’s `wait` and `notify` operations. Add syntax for these to the formal language, and add the appropriate type rules (see the Appendix of the paper).

2. In class, we briefly considered how one might adjust the Flanagan and Abadi calculus to be flow sensitive. That is, we would replace the `sync e` syntactic form with three new forms:

   \[ e ::= \ldots \mid \text{acquire } e \mid \text{release } e \mid e;e \]

   The first acquires the lock, the second releases it, and the third is sequential composition: it evaluates the first expression, throws away its result, and then evaluates the second. Typing judgments of the form \( E;p|e : t \) would have to change to be of the form \( E;p|e : t;p' \). That is, they now have an input permission \( p \) and an output permission \( p' \).

   Carry this change through to completion, adjusting the syntax and type rules of the monomorphic system (Figures 1 and 2 in the paper). Describe the changes that would also have to be made to the operational semantics (Figure 3 in the paper) to support this. For extra credit, present modified operational rules to support your proposed change.

   You are encouraged to do this using LaTeX. Try the semantic.sty package (Google it, or search for it in http://www.ctan.org). If you feel uncomfortable doing this, write the rules on paper, and e-mail me a scanned copy.
Checking for Race Conditions with rccjava

Installation

The two directory trees that implement the checker are installed on junkfood in /fs/unsupported/Javafe and /fs/unsupported/Rcc. To run the checker, I’ve provided two scripts: my-rcc and my-rccn. You can get these from http://www.cs.umd.edu/class/spring2004/projects/. They invoke scripts rcc and rccn in the Rcc directory, and set a bunch of environment variables to point at the right place. Since the checker is written in Java, you should be able to run them from any machine. However, I couldn’t run these from the Linux machines because the default Java doesn’t seem to work right. Therefore, I hard-coded them to point at /fs/unsupported/jdk1.4/bin/java. I can change this if someone tells me a reasonable way how.

If you want to run things on your own machines, you should be able to cp -r the Rcc and Javafe directories to your own machine, and adjust the environment variables in the above-mentioned scripts.

A final note: the scripts are set up to look first at the Java source for the standard libraries, and if a source file is not present, to look at the classfiles. However, this only seems to work for Java 1.3 source; this is what I have hard-coded in the scripts. To use it with Java 1.4 source, I found that you can create a “null” Java class for each class it complains it can’t find. For example, it complains for 1.4 that it cannot find sun.nio.ch.Interruptible (or something like that). Therefore, in the src/sun/nio/ch/ directory I created the Java file Interruptible.java with contents:

```java
package sun.nio.ch;
public class Interruptible { }
```

This seemed to make the checker happy. If you use the 1.3 sources, you shouldn’t have to do this.

Usage

The my-rcc script runs the checker on the Java files provided on the command line, while my-rccn runs the checker with some default flags (look at Rcc/rccn to see what they are). There is a brief man-page on the checker in the Rcc directory. From junkfood, you can view it by doing

```
man -M /fs/unsupported/Rcc rccjava
```

Proving the Absence of Race Conditions

I have provided two sample applications for you to try this on. They are based on projects done in CMSC 433 over the last two Falls. The first (java-server) is a parameterizable web server that uses multiple threads. If you want to understand how it works, take a look at http://www.cs.umd.edu/class/fall2002/cmsc433-0201/hw3/hw3.html. The second is a simulation of elevators carrying

In both cases, you should be able to go into the directory of the code and simply do

\texttt{my-rccn *.java}

This will run the checker using the standard flags on all of the files. Since you have not annotated these files, you will immediately get complaints from the checker. Your task is to add annotations to the files so that the checker doesn’t complain any more. This may not be possible. That is, you may actually have to change the programs in some (hopefully small) way to make the checker happy.

As you do this, time yourself. How long is it taking to make the changes? How might you automate this process? How many total annotations did you write, in how much time? How hard was it to make the changes? What could the tool have done better to make the task easier. Turn in a writeup with the answers to all these questions.