15-819 M: Program Analysis

Jonathan Aldrich

Assignment 1 (Programming): Simple Static Analysis

Due Wednesday, January 20, 1:30pm

Turn in a zip file named electronically in the Blackboard drop box. The zip file should contain the following files:

answers.xxx (with the answers to text questions in txt, pdf, or Word (doc/docx) format). At the top of the document, state your name and Andrew id.
UnreadScreenshot.xxx (in any common graphics format)
UnreadProject.zip (a zipped-up Eclipse project with your Unread analysis)
MyAnalysisProject.zip (a zipped-up Eclipse project with your second analysis)
MyAnalysisTest.java (the test case you ran your second analysis on)
MyAnalysisTestOutput.xxx (either graphical or text output for the test case)

100 points

Assignment objectives:

- Set up the Crystal analysis infrastructure in your environment and successfully compile and run an analysis
- Understand Abstract Syntax Tree representations more practically
- Write a simple Abstract Syntax Tree walker analysis
- Become familiar with the Crystal analysis infrastructure

Part I - Setup

Download and install the Java Development Kit 6 (if you don't already have it) from:
http://java.sun.com/javase/downloads/widget/jdk6.jsp
Select your platform and click Continue
Install by executing the downloaded file

Download and install Eclipse Classic 3.5.1 from:
http://www.eclipse.org/downloads/
For Windows, click on **Eclipse Classic 3.5.1**
For other platforms, click your platform to the right of Eclipse Classic
Install by unzipping the downloaded file
The first time you run it (via the eclipse executable in the eclipse/ directory), it will ask for a workspace location. The default is generally fine for most people.
Note: Eclipse for RCP/Plug-in Developers, or Eclipse Modeling Tools will also work. But the standard Eclipse IDE doesn't have the plugin development environment (PDE) and it won't work out of the box—but if you want you

can install the PDE manually. See the Crystal "getting started" page for more information

Install Crystal following the instructions at:
http://code.google.com/p/crystalsaf/wiki/Installation

Create a Crystal analysis plugin by following the instructions on the Crystal getting started page:
http://code.google.com/p/crystalsaf/wiki/GettingStarted

Make sure that you can run your analysis plugin (i.e. it appears in the Crystal menu when you run your plugin in an Eclipse child window). Your analysis, of course, may not do anything yet—that's Part II.

Part II (50 points)

In this part of the assignment, you will design a simple, visitor-based analysis that identifies variables and fields that are declared in the program but are never read. To be more precise, a read is any use of a variable or field that is not a write. A variable write occurs when a variable is directly on the left-hand side of an assignment expression, and a field write occurs when a field dereference is the outermost expression on the left hand side of an assignment expression. For example, x = y is a read of y and a write of x; x[5] = z reads both x and z, and x.y.z = 5 is a read of x and y and a write of z.

First, consider some simple design questions:

a) (10 points) What data structures will you use to keep track of what variables and fields exist, and which of those have been read? *Hint: consider the case where the visitor visits a use of a field before visiting its declaration*

Browse the Eclipse AST, which you can find at:
http://help.eclipse.org/galileo/topic/org.eclipse.jdt.doc.isv/reference/api/org/eclipse/jdt/core/dom/package-summary.html

(10 points) Which classes (sometimes more than one) represent:

b) a variable declaration?
c) a variable access?
d) a field declaration?
e) a field access?
f) an assignment?

Based on your design above, implement a simple tree-walker analysis within Crystal that finds variables and fields that are never read, and outputs a warning message to the Eclipse problems view for each one. Hint: to catch field declarations, you will need to analyze entire compilation units, not just methods, so use

AbstractCompilationUnitAnalysis (see PrintNodesAnalysis from the demonstration analysis on the course web page for an example). Run your analysis on the sample code distributed with this assignment, Countdown.java.

g) (10 points) Turn in a screenshot of the problems view when your analysis is run on the sample program. When capturing the screenshot, resize the window if necessary to show all the errors.
h) (20 points) Turn in your analysis code as a zipped Eclipse project.

Part III (50 points)

For this part of the assignment, design another simple visitor-based analysis to find some kind of bug. You may choose what kind of bug on which to focus; the list of bugs covered by FindBugs may give you some ideas:
http://findbugs.sourceforge.net/bugDescriptions.html

The analysis can be extremely simple (as in the previous part), and the bug can be shallow.

a) (15 points) Describe precisely under what conditions the analysis should report a bug. Your description should be at least as precise as the description of the unread variable and field analysis given above.

Implement your analysis within Crystal. Test it on a sample program that has at least 3 different instances of the bug, as well as 2 instances of code that is similar to code that would trigger the bug, but which is correct. The sample code given above, Countdown.java, is a test program that would fulfill these requirements for the unread variable and field analysis.

b) (10 points) Turn in your sample program, and the output from running your analysis on the sample program. The output can be a screenshot, as described above, or it can be text output from the Crystal output window.
c) (25 points) Turn in your analysis code as a zipped Eclipse project.