17-654/17-754: Analysis of Software Artifacts

Jonathan Aldrich

Assignment 1 (Programming): Simple Static Analysis

Due Monday, January 23, 5pm

Turn in a zip file 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) format)
UnreadScreenshot.xxx (in any common graphics format)
Unread.java
MyAnalysis.java
MyAnalysisTest.java
MyAnalysisTestOutput.xxx (either graphical or text output)

80 points

Assignment objectives:

- Understand Abstract Syntax Tree representations more practically
- Write a simple Abstract Syntax Tree walker analysis
- Become familiar with the Crystal analysis infrastructure

Part 1 (40 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) (5 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/help31/topic/org.eclipse.jdt.doc.isv/reference/api/org/eclipse/jdt/core/dom/package-tree.html. Which classes (sometimes more than one) represent:

b) (1 point) a variable declaration?
c) (1 point) a variable access?
d) (1 point) a field declaration?
e) (1 point) a field access?
f) (1 point) 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 PrettyPrintAnalysis 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.

Part 2 (40 points)

For this part of the assignment, design another simple visitor-based analysis to find some kind of bug (not just print out useful information, like PrintMethods). You may choose what kind of bug on which to focus. The analysis can be extremely simple (as in the previous part), and the bug can be shallow. However, exceptionally creative or well-executed analyses may receive up to 10 points of extra credit.

a) (10 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) (20 points) Turn in your analysis code.