INF 212 Analysis of Programming Languages
Project 9 – Program Analysis
Due date: 6/10
Demo date: 6/11

Objectives

• Understand Abstract Syntax Tree representations in depth

• 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/Plugin 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.

Documentation

Crystal Notes

Tutorial

Part II (60 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) 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
b) 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
c) Which classes (sometimes more than one) represent:
   1) a variable declaration?
   2) a variable access?
   3) a field declaration?
   4) a field access?
   5) an assignment?

Task 1. Based on your design above, implement a simple tree-walker analysis that finds variables and fields that are never read, and outputs a warning message to the Eclipse problems view for each one

Task 2. 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.

Part III (40 points)

Face-to-face demonstration and Q&A. The discussion will revolve around your understanding of ASTs, their traversal, and the methods for gathering knowledge with and from them.