CS264: Homework 1 (Pointer Analysis)

Assigned: Tuesday, Mar 1, 2005
Due: Tuesday, Mar 8, 2005, 7pm.

Submission

Please make sure your answers are brief and easy to understand. You can work on the problems with fellow students, but you are responsible for preparing and submitting your own solution. Submit your homeworks by email to `cs264@imail.eecs.berkeley.edu`.

1 Points-to Analysis vs. Alias Analysis

The pointer analysis we discussed in class is called *points-to* analysis. Points-to analysis approximates *alias* analysis, which computes whether pairs of expressions may be aliased. Expressions e_1, e_2 are aliased at a node n iff there exists a control-flow path from the program start node to the node n such that executing statements along the path makes e_1 and e_2 point to the same memory location. In flow-insensitive alias analysis, e_1 and e_2 are aliased iff one can construct a control-flow graph in which e_1 and e_2 are aliased at some node n.

In this problem, consider a simple language that includes only simple assignments with C-like right-hand-side expressions:

$$\begin{array}{rcl} stmt & \rightarrow & ID := exp \\ exp & \rightarrow & \&ID \\ & | & exp2 \\ exp2 & \rightarrow & ID \\ & | & *exp2 \end{array}$$

Note that expressions in this language may contain an arbitrary level of pointer dereferencing, for example, the assignment $a := *****b$ is a legal statement in this language.

1. Describe how to use Andersen's points-to analysis to answer whether arbitrary expressions exp_1, exp_2 are aliased. For example, how to determine if $***a$ and $*b$ are aliased?
2. Construct a program in which Andersen's points-to analysis computes an imprecise solution, that is, it answers that exp_1 and exp_2 are aliased even when there is no sequence of statements that would make them point to the same location. Explain the reasons for the loss of precision.

2 Andersen's Analysis for Java using CFL Reachability

Section 4.4 in the paper "Program analysis via graph reachability" describes a CFL-reachability-based formulation of points-to analysis. The paper presents the analysis for the C language; your task is to reformulate the analysis for Java, again using CFL reachability. Show the Java statements that your analysis will handle (you can ignore arrays) and give a suitable context-free grammar for the reachability problem. You can develop the Java analysis either by directly rewriting the grammar for C or by starting from scratch and taking advantage of properties unique to Java. For the latter option, a hint is that pointer semantics in Java leads to a balanced-parentheses grammar, which is not the case for C.