Final Exam, CS6013

Maximum marks = 60, Time: 3hrs

03-May-2020

Read all the instructions and questions carefully. You can make any reasonable assumptions that you think are necessary; but state them clearly. It is your responsibility to write legibly. There are total six questions you can attempt (total=66 marks); there is a ceiling (of 60) on the maximum marks you can obtain in the test. Each twelve marks question will approximately require around 30 minutes to answer. For questions that have sub-parts, the division for the sub-parts is mentioned in square brackets.

Start each question on a new page. Think about the question before you start writing and write briefly. **The answer for any question (including all the sub-parts) should NOT cross more than two pages.** If the answer is spanning more than two pages, the spill-over text will be ignored. If you scratch/cross some part of the answer, you can get compensation space from the next page.

1. [6 + 6] **Control flow analysis**

(a) Give example codes (one for each question bit) which show that for a node n

(i) its immediate dominator can be different than its CFG predecessor [2]

(ii) its CFG predecessor need not be its dominator [2]

(iii) its strict dominators includes only the immediate dominator [2]

(b) Answer the following questions in (at most) one sentence [1×6]:

(i) How many leaves can be there in a control tree?

(ii) In an if-then-statement, if the flow function of the if-expression is F_e and the flow function of the then-statement is F_t then what is the flow function of the if-then-statement.

(iii) Can a CFG have a node with no successor?

(iv) Can a CFG have a node with multiple successors and predecessors?

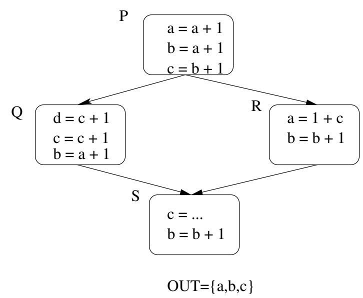
(v) True or False: Excluding the "Start" and "End" basic blocks, the number of basic-blocks is one more than the number of identified leaders.

(vi) True or False: If $Dom(n)$ denotes the set of dominators of a node n and $postDom(n)$ denotes the set of postdominators of n, then $Dom(n) \cap postDom(n) = \phi$.

2. [3+4.5+4.5] **Liveness analysis and register allocation**

(a) Write a piece of code (in C, using only integer type of variables), such that the resulting interference graph cannot be colored using 6 colors. Draw the interference graph to demonstrate the same.

(b) Compute and state the IN and OUT (related to liveness analysis) of the blocks P,Q,R in the CFG below. Assume that the OUT(S) = {a,b,c}.



(c) Answer the following questions in (at most) one sentence each [1 × 6]. During register allocation using iterated register coalescing

(i) State the advantage of moving a candidate variable to the stack instead of actually spilling - in the "(potential) spill" phase?

(ii) What is the need to rebuild the interference graph after the "actual spill" phase.

(iii) State the need to give up coalescing on one of the move-related nodes in the "Freeze" phase.

(iv) How many registers are kept aside for spilling? – not used for coloring.

(v) Can a variable be coalesced with a pre-colored node?

(vi) Can two precolored nodes be coalesced?

3. [8+4] **Points-to analysis**

(a) Write an example Java code snippet that shows the importance of flow sensitivity; the example should show how flow-sensitivity can improve the precision of points-to analysis. Repeat the same for context-sensitivity. [4 + 4]

(b) Answer the following question in the context of intra-procedural points-to/alias analysis of Java programs discussed in the class [1 × 4]:

(i) What is the size of the lattice?

(ii) What is the abstract stack (ρ) initialized to at the beginning of a function?

(iii) Between the abstract stack (ρ) and abstract heap (σ), in general, which one has more number of elements?

(iv) During the analysis of a function call, whose contents (ρ or σ) are guaranteed to not change, if any?

4.[9+3] **Loop optimizations**

(a) For each of the following three loop optimizations, ((i) Loop inversion, (ii) Loop unrolling, (iii) Loop tiling) briefly explain [3 × 3]

(I) about the transformation, using an example code snippet,

(II) feasibility: conditions under which the transformation is correct,

(III) profitability: when the transformation is profitable.

(b) Give an example code, where GCD test leads to identification of (i) precise dependence relation, (ii) conservative dependence relation [1.5 + 1.5].

5. [1+1+4+6] **SSA computation**

(a) What is the difference between the schemes of *minimal-SSA* and *pruned-SSA*?

(b) If we use the *minimal-SSA* scheme instead of *pruned-SSA* scheme then can it lead to incorrect SSA code or sub-optimal SSA code?

(c) Show an example program, where *pruned-SSA* leads to reduction of ϕ nodes (compared to *minimal-SSA*)?

(d) We use DF to denote dominance frontier and $DF+$ to denote iterated dominance frontier. Consider a set S of definitions for a variable t, in a program P. The SSA construction algorithm first builds $DF+(S)$ and then inserts ϕ nodes for the variable t in each of the nodes in $DF+(S)$. Why should we use $DF+(S)$, instead of $DF(S)$, to insert the ϕ nodes? Show an example program to establish/argue the insufficiency of using $DF(S)$ for inserting the ϕ nodes.

6 [1×6] **Miscellaneous** State true or false (answer any six). Only the first six answers will be evaluated.

(a) Lexical analysis can throw errors.

(b) Only lexical analysis, syntax analysis and semantic analysis passes can throw errors.

(c) LR parsers are less powerful than LL parsers.

(d) Using the three address codes discussed in the class, we can translate all the syntax of C language, except for the switch-case statements.

(e) There exist code snippets for which the conditional constant propagation algorithm will discover same number of constants as simple constant propagation algorithm.

(f) The constant propagation lattice has a finite height, but infinite number of elements.

(g) In languages like Java, call-graph construction process can improve the precision of the generated call-graph by taking into consideration points-to/alias information.

(h) Steensgaard's points-to analysis is as precise as Anderson's points-to analysis.

(i) A static analysis can accurately and precisely predict all the possible null-dereferences in Java applications.

(j) The abstract stack (ρ) and the abstract heap (σ) discussed in the class are examples of symbol tables.