Final Exam, CS6013

Maximum marks = 60, Time: 3hrs

15-Nov-2021

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=64 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 (and write your roll number on each page – both sides of the sheet, that is) Think about the question before you start writing and write briefly.

1. [4+2+2+4] **Control flow analysis and Optimizations**
   (a) Give an algorithm to eliminate unreachable basic-blocks from a given CFG.
   (b) Give the conditions under which two consecutive basic-blocks can be merged into a single basic-block.
   (c) Given a canonical for-loop, how will you perform loop-inversion?
   (d) Recall that a basic-block is defined as a maximal sequence of single-entry-single-exit instructions. Given a function with `N` (≥ 1) three-address-code instructions, what is the minimum and maximum possible number of basic-blocks (excluding the ENTRY and END basic-blocks)? Justify your answers with an example each.
2. [3+4+5] **Data Flow Analysis**
   (a) Given a may-points-to analysis algorithm, state what changes will you make in the analysis algorithm to derive the "must"-points-to analysis algorithm.
   For the remaining of the question, assume that both these analysis are available in appropriate data structures.
   (b) Say, we are performing constant propagation and our goal is to evaluate the boolean expressions of the form (`x == y`) and (`x != y`), where `x` and `y` reference type variables. State how you will compute the (constant) value for such expressions.
   (c) State how you will extend the constant propagation scheme (studied in the class) to additionally handle field accesses. Say the field accesses are of the form: `x = y.f` and `y.f = x`, where `x` is a scalar type (integer) variable and `y` is a reference type variable.
3. [8+4] **Dependence Analysis**
   (a) Write an example loop-nest in C, with four statements `S1`. `S2`, `S3`, and `S4` such that
      (i) there is a dependence from `S1` to `S2`, which can be represented as by the direction vector $< \geq, =, = >$. [4 marks]
      (ii) there is a dependence from `S3` to `S4`, which can be represented by the distance vector $< 1, 0, 1 >$. [4 marks]
   (b) Complete the below given loop-nest (in C language) such that GCD test will identify that there there is no loop-carried dependence between `S1` and `S2` and there is an intra-iteration dependence between `S2` and `S3`.

```
for (i=0;i<n;++i)
  for (j=0;j<n;++j) {
     S1: --------------
     S2: x [2*i, i+j] = i + j;
     S3: --------------
  }
```

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

(a) Give example C codes with loops where the following optimizations CANNOT be applied safely. (i) Loop distribution [3], (ii) Loop unswitching [3], (iii) Vectorization [3].

(b) Given two loops L1 and L2 in sequence, give the conditions under which L1 and L2 can be interchanged. That is, if in the original code L1 executes before L2, then after the transformation, L2 executes before L1.

5. [3+3] **SSA computation**

(a) Given a CFG with N nodes, what is the maximum and minimum number of blocks in which SSA ϕ nodes may have to be inserted using the pruned-SSA scheme?

(b) Write a C code where the SSA based sparse conditional constant algorithm, studied in the class, can identify more constants than the traditional conditional constant propagation algorithm.

6 [1×10] **Miscellaneous**
State true or false.

(a) A function containing at least a single if-then-else statement must have a ϕ node in the SSA form.

(b) A function containing a single if-then-else statement (with some code before and after) can at most have a single ϕ node after the if-then-else statement in the SSA form.

(c) "Uninitialized variable" is an error thrown by the parser.

(d) The debugger needs compiler support to preserve the variable names.

(e) Precise points-to analysis requires thread-escape analysis information.

(f) In a subset of minijava language with no loops, no conditional-statements, we can compute the points-to analysis fully preciesly.

(g) The points-to lattice has finite height, but infinite elements.

(h) Since we don't perform fixed point computation, forward analysis using structural analysis is less precise than the iterative data-flow analysis.

(i) The flow functions for Structural analysis are applied on the nodes of CFG.