

## ENEE759C Spring 2005

### Quiz

Name: \_\_\_\_\_ *Answer key*  
Student ID: \_\_\_\_\_

**Pages: 6 printed sides**  
Time allotted: **1 hour 15 min**  
Maximum score: **55 points**

University rules dictate strict penalties for any form of academic dishonesty. Looking sideways will be penalized. Look at only your own exam at all times.

There are 7 questions, some with subparts. **Read them *carefully* to avoid throwing away points!!** Write your answer in the space provided. *Closed book, closed notes. Calculators are allowed.*

***Partial credit rule: Must show your intermediate steps clearly for partial credit!***

---

1. Fill in the blanks with one, two, or at most three words:

**(1 point \* 15 = 15 points)**

- (a) Compiler usually represents the program internally using architecture-independent instructions whose specification is called its \_\_\_\_\_ intermediate form \_\_\_\_\_.
- (b) The linker \_\_\_\_\_ is a software that runs after the compiler that allows multiple object files to be combined into a single executable.
- (c) Before register allocation is run, the operands of register-only instructions are \_\_\_\_\_ virtual registers \_\_\_\_\_ which are infinite in number.
- (d) A control-flow graph represents control flow within a \_\_\_\_\_ procedure \_\_\_\_\_.
- (e) In a \_\_\_\_\_ reducible \_\_\_\_\_ flow graph, all the cycles in the graph are in loops.
- (f) The dominator tree \_\_\_\_\_ is a data structure that succinctly represents the information about dominance between flow-graph nodes.
- (g) A tree is a graph in which each node has only one \_\_\_\_\_ parent \_\_\_\_\_.
- (h) A/An induction \_\_\_\_\_ variable is one whose only update in a loop is an increment by a constant value.
- (i) Elementary statements in the language for traditional dataflow analysis are called quadruples \_\_\_\_\_.
- (j) In compiler terminology, a/an \_\_\_\_\_ affine \_\_\_\_\_ function is a linear combination of enclosing loop induction variables.
- (k) Dead-code elimination is most naturally performed using the results of the \_\_\_\_\_ liveness analysis \_\_\_\_\_, which is a type of traditional dataflow analysis.
- (l) \_\_\_\_\_ Constant folding \_\_\_\_\_ is the name of an optimization that executes program instructions at compile-time if their source operands are known.
- (m) In SSA form, the number of arguments of a  $\phi$  function inserted in a node is equal to the number of \_\_\_\_\_ predecessors \_\_\_\_\_ of that node.
- (n) For \_\_\_\_\_ backward \_\_\_\_\_ dataflow analysis, an end node must be inserted into the flow graph for proper iterative computation of the dataflow results.

(o) The copy propagation optimization can eliminate some variable-to-variable copy instructions from the program.

2. For each subpart (i) to (iii) below, circle **all** correct answers from among the four given - note that **more than one** answer may be correct!

(2 points \* 3 = 6 points)

(i) Traditional dataflow analysis

- (a) needs the concept of control-flow graphs to compute its results.
- (b) is linear in runtime in the number of instructions in the compiled program.
- (c) can enable strength reduction of induction variable computations.
- (d) is a collection of analyses mechanisms rather than a single such mechanism.

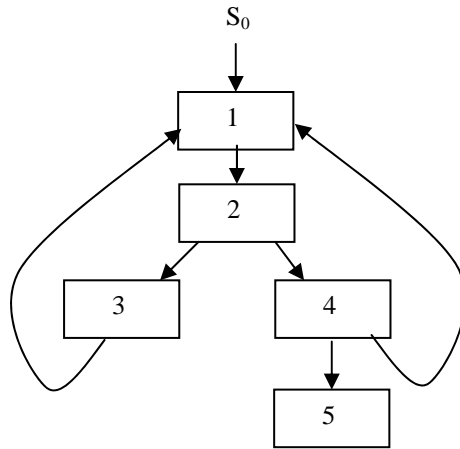
(ii) Alias analysis

- (a) is a form of dataflow analysis.
- (b) can be done in a limited way by using the address-taken information for each variable.
- (c) can affect the results of dataflow analysis.
- (d) can represent its information as a bi-partite graph.

(iii) Induction variable analysis

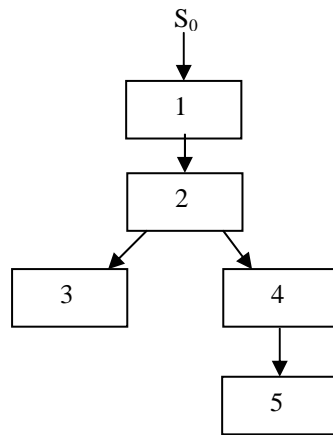
- (a) helps identify affine function accesses.
- (b) produces results that are used by dead code elimination.
- (c) is essential for correctly compiling programs into executable form.
- (d) enables hoisting in some cases.

3. Consider the following control flow graph:



(3 + 2 + 4 = 9 points)

(a) Using intuitive reasoning, draw the dominator tree of the above CFG.



(b) State a formal definition of a back edge. What are the back edges in the CFG?

An edge  $(n,h)$  is a back edge if  $h \text{ dom } n$ ,

Back edges in above CFG:  $(3,1)$  and  $(4,1)$ .

(c) How many natural loops are present in the above CFG? For each, list the member nodes and the header.

There are two natural loops, since each back edge induces a natural loop.

Loop for  $(3,1) = \{ 1,2,3 \}$ ; header = 1.

Loop for  $(4,1) = \{ 1,2,4 \}$ ; header = 1.

4. It is sometimes useful to know if the value of a variable is input dependent. For this reason, in this question we will define a new traditional data flow analysis, called **input-dependent variables**. In this analysis, the  $in[n]$  and  $out[n]$  sets for each instruction  $n$  in the program contain the set of variables whose values are input-dependent. A variable is said to be input-dependent if its value could be affected by the values of the external inputs to the program. Assume that the  $get\_input()$  library routine, which may be called from the program, is the only way to obtain an external input value (for example, from a file).

(2 + 5 + 1 = 8 points)

- (a) For this forward analysis, write down the formulation for  $in[n]$  and  $out[n]$  for each statement  $n$  in the program.

$$in[n] = \bigcup_{p \in pred[n]} out[p]$$

$$out[n] = gen[n] \cup (in[n] - kill[n])$$

- (b) Write down the  $gen$  and  $kill$  sets for the **input-dependent variables** analysis in the table below. Assume that a function  $ID(x)$  is available for each read operand  $x$  of each statement  $n$ , which returns TRUE if  $x \in in[n]$  and FALSE otherwise (*i.e.*,  $ID(x) = x$  is input-dependent). You can use  $ID(x)$  in writing  $gen[s]$  and  $kill[s]$ . Assume that  $ID(f)$  is also available for each function  $f$  where  $ID(f)$  is TRUE if function  $f$  returns an input-dependent value for any one of its return statements. Although  $ID(f)$  is itself computed using dataflow equations, you do not need to show its computation.

Statement $s$	$gen[s]$	$kill[n]$
$d: t \leftarrow b \oplus c$	If ( $ID(b)$ or $ID(c)$ ) then $\{t\}$ else $\{\}$	$\{t\}$
$d: t \leftarrow M[b]$	$\{t\}$	$\{t\}$
$M[a] \leftarrow b$	$\{\}$	$\{\}$
If $a$ relop $b$ goto $L1$ else goto $L2$	$\{\}$	$\{\}$
Goto $L$	$\{\}$	$\{\}$
$L:$	$\{\}$	$\{\}$
$f(a_1, \dots, a_n)$	$\{\}$	$\{\}$
$d: t \leftarrow f(a_1, \dots, a_n)$	If ( $ID(f)$ ) then $\{t\}$ else $\{\}$	$\{t\}$

- (c) What should be the initial values of  $in[n]$  and  $out[n]$  to correctly solve the equations in the previous parts using a fixed point iteration?

$in[n] = \text{NULL set (for all } n\text{)}$ .

$out[n] = \text{NULL set (for all } n\text{)}$ .

5. Consider the following program fragment:

(3 + 5 = 8 points)

```
a = 0
While (sum < x) {
    sum = sum + a
    a = 5
}
```

- (a) The assignment  $a = 5$  cannot be hoisted out of its enclosing while loop. State an intuitive reason why. Also state the formal rule whose being false prevents hoisting in this case.

Intuitive reason: The value of  $a$  in  $sum = sum + a$  in the first iteration of the loop is defined outside the loop, and thus hoisting makes the code incorrect.

Formal rule whose being false prevents hoisting in this case:  $a$  should not be live-out of the loop pre-header.

- (b) Describe a compiler optimization in words that will allow the hoisting of the  $a = 5$  statement out of its enclosing loop. Show the resulting code after applying this optimization and hoisting to the code above. Do not apply constant propagation or folding.

Compiler optimization: Do the first iteration in an if statement, and subsequent iterations thereafter in a while loop inside of the if statement.

Code after optimization and hoisting:

```
a = 0
if (sum < x) {
    sum = sum + a
    a = 5
    While (sum < x) {
        sum = sum + a
    }
}
```

6. Prove by contradiction that if  $a \text{ dom } b$  and  $b \text{ dom } c$ , then  $a \text{ dom } c$ .

(4 points)

Assume for the sake of contradiction that  $a \not\text{dom } c$ .

$\Rightarrow \exists$  Path  $P$  from  $S_0$  to  $c$  not containing  $a$ . (1)

We know that  $P$  must contain  $b$  since  $b \text{ dom } c$ .

*// from definition of dominance.*

We also know that portion of  $P$  from  $S_0$  to  $b$  must contain  $a$ . *// from definition of dominance.*

$\Rightarrow P$  contains  $a$ . (2)

(1) and (2) are a contradiction.

$\Rightarrow a \text{ dom } c$ .



7. Pointer analysis is run on the following C program without optimizing the program in any way:

(2 + 2 + 1 = 5 points)

```
main() {
    int a=5, b=3, c = 7; *p, **q;
    q = &p;
    if (foo(a)) {
        p = &a;
    } else {
        p = &b;
    }
    a = *p + 2;    // Statement 1
    *q = &c;
    b = *p + 4;    // Statement 2
}
foo(int x) { return x > 0; }
```

(a) For a flow-sensitive pointer analysis, what is the set of variables that p can point to in statement 1?

{a, b}

(b) For a flow-insensitive pointer analysis, what is the set of variables that p can point to in statement 1?

{a, b, c}

(c) For a flow-sensitive analysis, at the end of statement 2, state the value (TRUE or FALSE) of (i) \*p may-alias a; and (ii) p may alias &q.

\*p may-alias a = TRUE  
p may alias &q = FALSE