NAME: Sunil ROLL NO: 18111076

CS738: Advanced Compiler Optimizations
End Semester Examination, 2018-19 I

Max Time: 3 Hours Max Marks: **180**

NOTE:

- There are total **6** questions on **5 pages.**
- Write your name and roll number on the question paper and the answer book.
- Presenting your answers properly is your responsibility. You lose credit if you can not present your ideas clearly, and in proper form. Please DO NOT come back for re-evaluation saying, "What I actually meant was ...".
- Be precise and write clearly. Remember that somebody has to read it to evaluate!
- **IMPORTANT: Some questions have to be answered on the sheet provided at the end of the question paper. Write you Name and Roll Number, and return it along with the answer book.**

Notations

- CFG stands for control flow graph.
- IN(S) denotes the program point before the statement S. OUT(S) denotes the program point after the statement of S.
- PRED(S) denotes the set of predecessors, and SUCC(S) denotes the set of successors of S.

1. [15 + 20] **Always Visible Variable.** A variable v is *always visible* at a program point π if at least **one** of the following conditions hold:

   - There is an access (definition or use) of v on every path from Entry to π

     OR

   - There is an access of v on every path from π to Exit.

   Figure 1 shows the CFG for an example program having variables A, B and C.

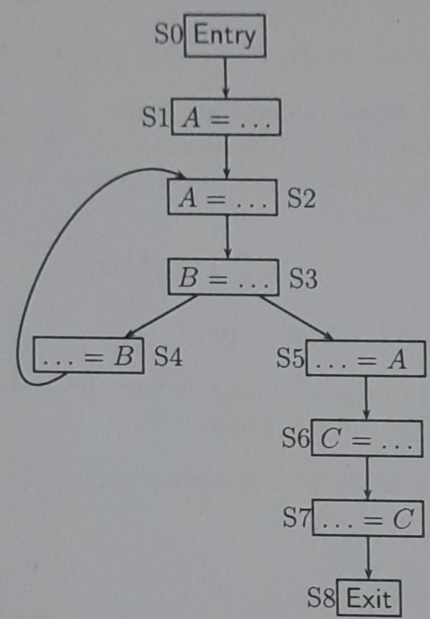


Figure 1: An Example Program

   (a) Complete the Table 1 (on the last page) to show the program points where the variables A, B and C are visible for the CFG in Figure 1.

   (b) **Define data flow equations to compute** VsbleIN(S, v) **and** VsbleOUT(S, v), where

   - VsbleIN(S, v) is true if IN(S) has variable v as visible.
   - VsbleOUT(S, v) is true if OUT(S) has variable v as visible.

2. [15 + 10] **Visibility of a Set of Variables:** A set of variables, φ, is visible at a program point π if at least **one** the following conditions hold:

   - There is an access to some variable $v \in \varphi$ on every path from Entry to π

     OR

- There is an access to some variable $v' \in \varphi$ on every path from π to Exit.

Note that v and v' may or may not be the same variable in φ. Also, different variables from the φ may be accessed along different paths.

(a) Complete the Table 2 (on the last page) to show the program points where the given sets of variables are visible for the CFG in Figure 1.

(b) **Define equations to compute $\mathsf{VsblSetIN}(S, \varphi)$ and $\mathsf{VsblSetOUT}(S, \varphi)$, where**
- $\mathsf{VsblSetIN}(S, \varphi)$ is true if $\mathsf{IN}(S)$ has the set of variable φ visible.
- $\mathsf{VsblSetOUT}(S, \varphi)$ is defined analogously.

You can reuse any quantities defined in Question 1.

3. [10] Consider the semantics for the **if** expression in our simple language of arithmetic, where $\mathtt{t} \rightarrow \mathtt{t'}$ denotes "t evaluates to t′ in one step"

$$\texttt{if true then } \mathtt{t_2} \texttt{ else } \mathtt{t_3} \rightarrow \mathtt{t_2}$$

$$\texttt{if false then } \mathtt{t_2} \texttt{ else } \mathtt{t_3} \rightarrow \mathtt{t_3}$$

$$\frac{\mathtt{t_1} \rightarrow \mathtt{t_1'}}{\texttt{if } \mathtt{t_1} \texttt{ then } \mathtt{t_2} \texttt{ else } \mathtt{t_3} \rightarrow \texttt{if } \mathtt{t_1'} \texttt{ then } \mathtt{t_2} \texttt{ else } \mathtt{t_3}}$$

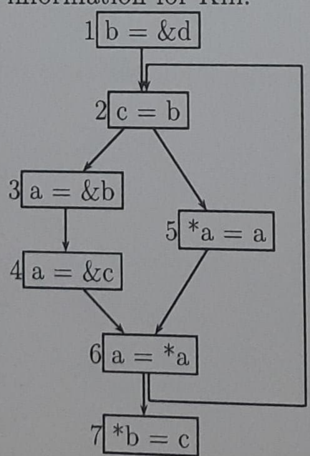
We want to change the evaluation semantics so that the **then** and **else** branches are evaluated (in that order) *before* the guard is evaluated. Show the modified semantics.

4. [10+25] Perform **Flow Sensitive May Points-to** analysis for the program below.

(a) Give Local Data Flow information for each block in terms of May sets. [Do not write the generic equation, but instantiate it for the current statement.]

(b) Show result of each iteration of the analysis (Maximum **3** iterations only).

Assume conservative *Must* information for Kill.



5. [10 + 20] Consider a simple programming language with only two primitive types: **int** and **bool**, and the corresponding array types **int[ ]**, **bool[ ]**. The language specification does not require a variable to be declared before use. However, the language requires that for a program to be valid, every variable must be assigned a *unique* type during compilation.

The following table describes the operations permitted by the language, the corresponding type restrictions on the argument variables and the result of the operation.

Operation	Example	Type Restrictions on Args	Result Type
==	x == y	x and y have to be of the same type (int or bool)	bool
+	x + y	x and y both are int	int
cint	cint(x)	cast to int: x can be of any type	int
[]	x[i]	i^{th} element in x of array type T[ ], i has to be integer	T

Note that arguments to binary operators can be array accesses, e.g., x[5] == y OR y[3] + z[4]. The language has following constants (each constant has a fixed type):

Const Class	Constants	Type
int_constant	$-\infty \ldots -2, -1, 0, 1, 2, \ldots \infty$	int
bool_constant	false, true	bool

The language has the following constructs and the restrictions:

- *assignment* statements: Both sides of an assignment must be of the *same* type.
- Conditionals (*if-else*): The condition of the *if-else* is a single variable of type bool.

Here are couple of sample programs, one valid and other invalid.

```
// PROGRAM 1
c = n[5] == e;
e = 5;
if (c) {
  d = e + 3;
} else {
  d = cint(c) - 5;
}
// c: bool
// d, e: int
// n: int[ ]
```

```
// PROGRAM 2
c = n==e;
e = false;
if (c[4]) {
  d  = c + 3;
} else {
  d  = cint(e) - 5;
}
// invalid program
// c has to be both bool[] and int!
// n, e: bool     d: int
```

(a) Design a simple (as-simple-as-possible for you) syntax (context free grammar) for the language described.

(b) Design a **flow-insensitive** type system to infer (or validate) types of variables for programs written in the above syntax.

6. [10 + 5 + 15 + 5] Perform *Interprocedural Available Expressions* analysis for the following program.

main() {	z() {	q() {
C_z: call z()	a * b	if(...) then
c * d	if(...) then	a = 5
C_q: call q()	C_q: call q()	else
a * b	else	b = 6
C_z: call z()	c = 7	}
}	}	

Show the following steps to arrive at the answer:

(a) Draw the inter-procedural flow graph (any one of the two representations discussed in class). Clearly label the nodes, and distinguish the E^0 (intra-procedural) and E^1 (inter-procedural) edges.

(b) For each procedure p, show the constraints for $\phi(r_p, n_p)$ for each node n_p in the CFG of the procedure. Recall that r_p is the entry node for procedure p. Assume the existence of flow functions f_0, f_1, id etc for the underlying data flow framework.

(c) Give the fixed-point solution for $\phi(r_p, n_p)$, for each procedure p.

(d) List the program points where expression `a * b` is available. List the program points where expression `c * d` is available.