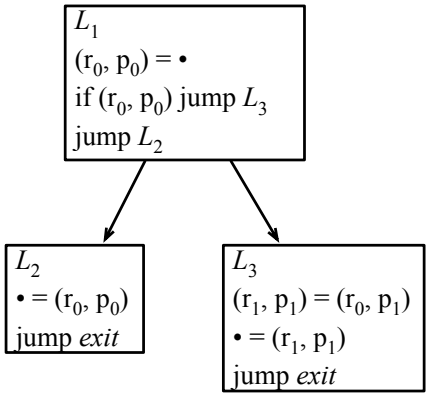
DCC888 – Correctness

Name: _____ ID: _____

1. This question refers to the program in C, seen below on the left, and to the program in FiRM on the right:

```
int* foo() {
  int* r0 = NULL;
  if (r0) {
    return r0;
  } else {
    float r1 = r0;
    return r1;
  }
}
```



(a) The C program above has a typing error. Which error is this one?

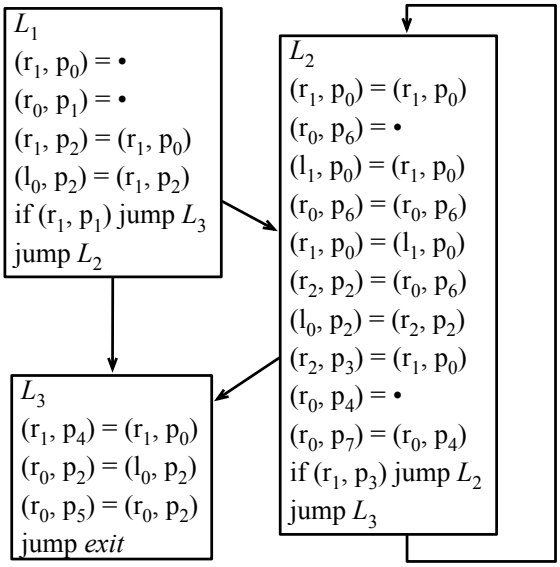
(b) The FiRM program on the right has an analogous typing error. In this case, what is the type of r_0 and r_1? What is the typing error?

(c) When does the C compiler catches the error on the C program on the left?

(d) When does our static validator catches the error on the FiRM program on the right?

(e) Can you define an analogy between type checking in C, and in FiRM?

2. This question refer to the program below, which represents an instance of register allocation. An instruction such as $(r_1, p_0) = (l_1, p_0)$ means that we are loading a value stored in memory location l_1 into register r_1. The register allocator has assigned, at that program point, register r_1 and memory location l_1 to the variable p_0. In other words, this instruction represents a load used to recover the spilled value of p_0.



   (a) Is this allocation correct? In other words, does this assignment of registers and memory locations to variables preserves the semantics of the original program?

   (b) Find a typing environment for each basic block to ensure that the basic block will execute soundly.

   (c) Determine the type of the second to last instruction in block L_2. This instruction is "if (r_1, p_3) jump L_2".

   (d) Now, determine the type of the last instruction in block L_2. This instruction is "jump L_3".

3. If we assume that the interference graph correctly represents the pattern of interferences in a given program, then we can easily provide a translator validator to a graph coloring register allocator. Describe what this translator validator must do to ensure a correct allocation.

4. Often times compilers must implement parallel copies, such as $(x, y, z) \leftarrow (x_1, y_1, z_1)$. This copy has the following semantics: first every value in the right side is read, and then every value in the left side is written, all in parallel. Often we only have two types of instructions to implement parallel copies: move instructions and swaps. If we assume that R, the bank of registers, is a function that maps registers to values, then these instructions have the following semantics:

$$(R, \text{copy}(a, b)) \to R[a \mapsto R[b]]$$

$$\frac{R[a] = x \qquad R[b] = y}{(R, \text{swap}(a, b)) \to R[a \mapsto y][b \mapsto x]}$$

The rest of this question refers to parallel copies, which must be implemented with sequences of the two instructions above.

(a) When do compilers have to implement parallel copies? Why do they surface in the design of a compiler?

(b) Implement the following parallel copies:

i. $(a, b, c) \leftarrow (b, c, d)$

ii. $(a, b, c) \leftarrow (a, a, b)$

iii. $(a, b, c) \leftarrow (b, c, a)$

(c) Let the *transfer graph* of a parallel copy p be a graph with a vertex v for each location in p, and an edge v, u for each copy of u into v that the parallel copy encodes. Draw the three transfer graphs for each of the parallel copies in the previous question.

(d) What is the general format of a *transfer graph*? Assume that all the locations on the left side of a parallel copy are different, but locations on its right side may be repeated.

(e) Design an algorithm to perform the translation validation of the implementation of a parallel copy. Your algorithm must receive two inputs. The first input is a parallel copy p, which is represented as two vectors of locations, e.g., $(x_1, x_2, \ldots, x_n) \leftarrow (y_1, y_2, \ldots, y_n)$. The second input is a sequence I of instructions, moves and swaps, that is meant to implement the parallel copy, e.g., swap(a, b), swap(b, c), copy(d, e). Your algorithm must ensure that I preserves the semantics of p.