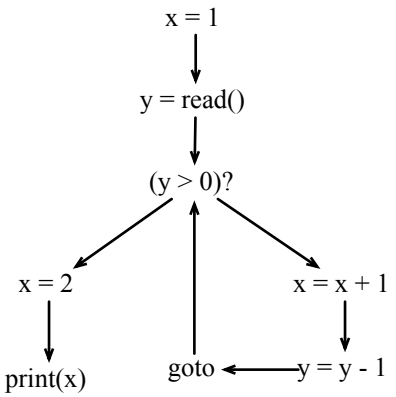
DCC888 – Data Flow Analyses

Nome: _____ Matrícula: _____

1. Show the result of liveness analysis for the program below:

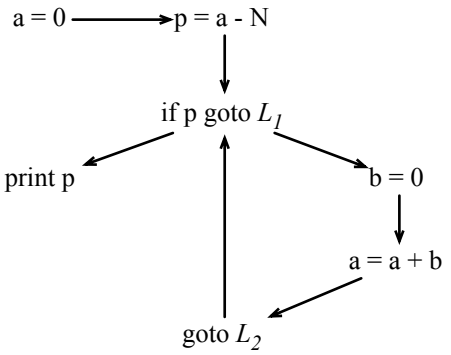


2. Consider the following program: `x = 1; x = x - 1; x = 2`. Variable `x` is dead at the exit of the second instruction. On the other hand, `x` is alive after the first instruction, even though its value is only used to compute a dead variable. Under such circumstances, we say that `x` is a *quasi-dead* variable. In other words, a variable is *quasi-dead* either if it is dead, or if it is only used to compute a quasi-dead variable. In our example, `x` is quasi-dead in the OUT set of each one of the instructions. If a variable is not quasi-dead, then we shall call it *vigorous*. Design a data-flow analysis to detect vigorous variables.

3. There exists a very simple analysis to detect the arithmetic sign of the numeric variables in a program. This analysis associates each variable with a subset in the set $\{+, -, 0\}$. For example, if a variable can only assume the values 0, 1, 2 and 3 during the execution of a program, then its abstract state is $\{0, +\}$. Design a set of transfer functions to compute this analysis. Assume that your underlying programming language has the following syntactic categories of instructions:

   (a) $a = n, n \in N$
   (b) $a = b, \{a, b\} \subset Var$
   (c) $a = b - c, \{a, b, c\} \subset Var$
   (d) $a = b + c, \{a, b, c\} \subset Var$
   (e) $a = b \times c, \{a, b, c\} \subset Var$
   (f) if a goto L_i, $a \in Var, L_i \in Label$
   (g) goto L_i, $L_i \in Label$
   (h) print a, $a \in Var$

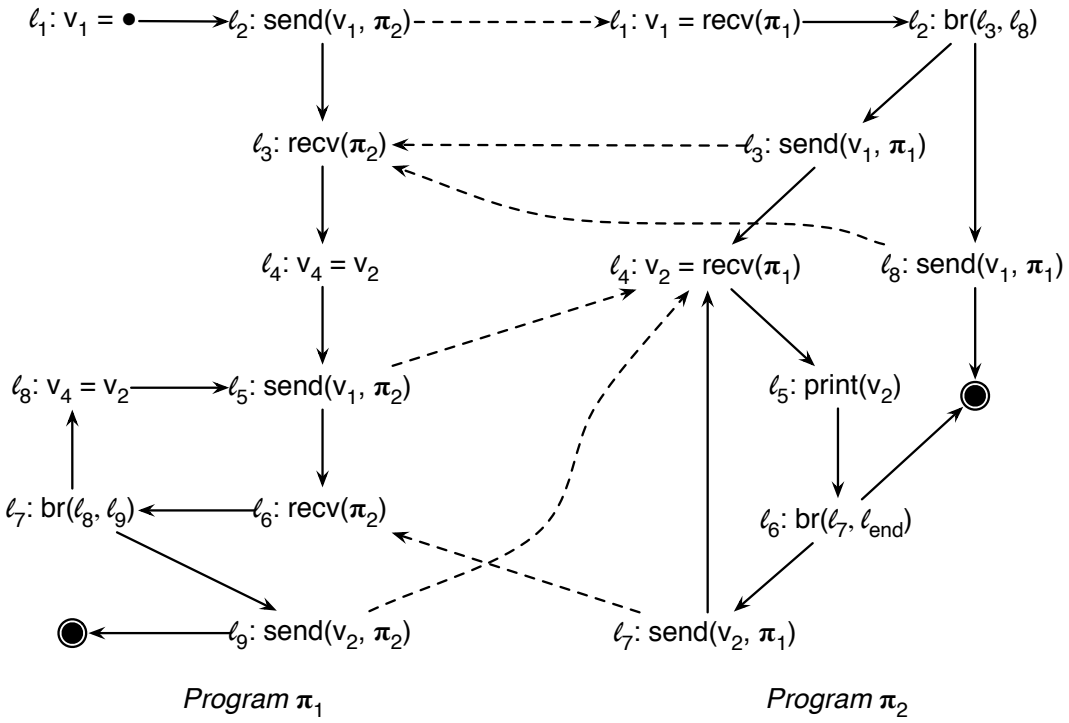
4. Show the result of your sign analysis to the program below:



5. Consider the following assembly-like programming language, which has only the seven instructions below:

Programs (P)	::=	$\ell_1 : I_1, \ell_2 : I_2, \ldots, \ell_n : I_n$
Labels (L)	::=	$\{\ell_1, \ell_2, \ldots\}$
Variables (V)	::=	$\{v_1, v_2, \ldots\}$
Instructions (I)	::=	
– Definition	|	$v = \bullet$
– Copy	|	$v_0 = v_1$
– Print v	|	$\texttt{print}(v)$
– Send v's value to π	|	$\texttt{send}(v, \pi)$
– Receive value from π	|	$v = \texttt{recv}(\pi)$
– Branch randomly	|	$\texttt{br}(\ell_0, \ell_1)$
– Unconditional branch	|	$\texttt{jmp}(\ell)$

Programs written in this language may exchange messages. Below we have an example of two programs that communicate:



(a) In the figure above, we have drawn arrows between pairs of sends and receives that can exchange messages. Notice that not every pair of such instructions can communicate. For instance, every message produced by $\ell_2 : \text{send}(v_1, \pi_2)$ in program π_1 will be consumed by $\ell_1 : v_1 = \text{recv}(\pi_1)$ before the execution flow of program π_2 reaches any other program point.

In this question you must design a program analysis that estimates the communication links between all the pairs of sends in a program (known as the *source*) and all the pairs of receives in another program (known as the destination). Your analysis must be *conservative*, i.e., it must correctly point out any possible exchange of messages. However, your analysis cannot be *trivial*, i.e., it cannot say that every exchange is possible for every pair of programs. You can rely on a few assumptions:

i. Our distributed system contains exactly two programs, π_1 and π_2.
ii. Every execution flow starts from label ℓ_1 of each program.
iii. Receive operations have a blocking semantics, e.g., once the program flow reaches a label containing a receive, it will block, until a message arrives.
iv. Send operations do not wait for confirmation. In other words, these operations do not block.

(b) Your algorithm must terminate for any pair of programs, assuming that we only deal with programs containing a finite number of labels. Provide an argument on why your algorithm always terminates.

(c) What is the complexity of your algorithm? Explain its complexity in terms of the number of labels in the programs **and/or** the number of sends **and/or** the number of receives **and/or** the number of variables.

(d) Explain one use of your analysis in a distributed system. You must tell in which scenario the information produced by your analysis can be useful. Explain to **whom** it is useful, and **how** it can be used. Allow yourself to be creative.