Exercise 4: Liveness Analysis

Perform liveness analysis on the following piece of code

```
1: var  x,y,z;
2: x = a + b
3: y = x + 1
4: z = x
5: if (x > 0) {
6: z = x + 1
} else {
7: x = 2
}
8: y = z + x
9: output  y;
```

- drawing its control flow graph,
- computing the constraints for each block, and
- computing the least solution of the constraints.

[*Optional*] Do the results of liveness analysis on this code enable any optimization opportunities? If so, describe the optimization.

LANGUAGE BASED TECHNOLOGY FOR SECURITY
July 19 2024

Duration 40 minutes – Answers can be written either in English or in Italian.

Exercise 1: Dynamic Information Flow

Consider the following function under an information flow security policy over the lattice $L \leq H$.

```
bool function f(x) {
 y = true;
 z = true;
 if x then y = false;
 if (!y) then z = false;
 return !z;
 }
```

We assume to consider a dynamic information flow mechanism where the typed security policy Γ is *dynamic*. We also assume that the first two assignments to variables y and z are low-level assignments (namely, the corresponding types and values are at the low level L).

- Discuss the dynamic information flow policies associated to the execution of the function f above. Consider two cases:
  1. Function f is called with the true value at the security level H;
  2. Function f is called with the false value at the security level H;

Exercise 2: Static Taint Analysis

Consider the following program

```
x = getInput();
a = x;
b = a*a;
d = 10;
if d < a then y = 0 else y = a+1;
if d > b then y = y+1 else y = b-1;
print(y);
```

- Discuss how static taint analysis is applied to discover whether the previous program fragments presents taint flow. Specifically define and illustrate the constraints generated by the static taint analysis.

Exercise 3: Security Polices

Consider an intermediate programming language designed to enable near-native code execution. We assume that the language includes standard instructions and is equipped with new feature to enable data flow reasoning about tainted data.

1. Design the *Security automaton* specifying the taint security policy associated to the *jump instruction* jmp e where e is an expression.
2. Exploit the security automata defined in the previous question to instrument the code below:

```
x = getInput(0);
y = x + 8;
jmp y;
```