Exercise: Data FlowAnalysis

1 Available Expressions [38 points]

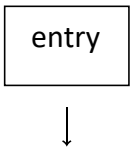
Consider the following program in a toy language with syntax inspired by Python. Assume all variables are integers and operators have the obvious semantics.

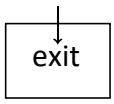
```
1  x = a - 3
2  y = a + 3
3  if x > a + 3:
4          a = a * 3
5  else:
6          x = a + 3
7  end
8  y = a - 3
```

Your task is to perform the *Available Expressions* data flow analysis, as presented in the lecture. Complete the following subtasks.

1.1 Control-Flow Graph [8 points]

Complete the following drawing of the control-flow graph (CFG) of the given program. As in the lecture, nodes are individual statements (and not basic blocks, as in other definitions of CFGs). Label each statement/node in the graph with a unique number, e.g., the line number in the source program, to help with the following subtasks. Do not include elseor endin the CFG.





Check your result: The number of nodes (including entry and exit nodes) plus the number of edges (including edges from/to entry/exit nodes) should be 16.

1.2 Transfer Function [10 points]

First, list all *non-trivial, arithmetic expressions* in the program, i.e., expressions involving the operators: +, -, and *.

Expressions: *{* *}*

Next, fill the following table with the *gen* and *kill* sets of each statement in the program. For the first column, use the numbers from the CFG for identifiying statements.

Statements	$gen(s)$	$kill(s)$

1.3 Solving Data Flow Equations [16 points]

Now, use the iterative algorithm from the lecture to solve the data flow equations for each statement in the program. You can iteratively fill up the second and third column of the table below during solving.

Statements	$AE_{entry}(s)$	$AE_{exit}(s)$

1.4 Understanding [4 points]

Finally, to show your wider understanding of the applications of this data flow analysis, answer the following question: How could a compiler use the results of *Available Expressions* analysis to optimize the original program? (1-2 sentences as answersuffices.)

2 Live Variables [38 points]

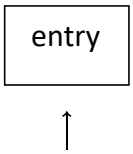
Consider the following program in a toy language with syntax inspired by Python. Assume all variables are integers and operators have the obvious semantics.

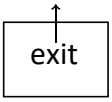
```
1  x =5
2  y =0
3  while x > 0:
4          x = x - 1
5          while y < 10:
6                  y = x + y
7  end
8  y =3
```

Your task is to perform the *Live Variables* data flow analysis, as presented in the lecture. Complete the following subtasks.

2.1 Reversed Control-Flow Graph [8 points]

Complete the following drawing of the *reversed* CFG of the given program, i.e., where all edges are inverted compared with the regular CFG. As in the lecture, nodes are individual statements (and not basic blocks, as in other definitions of CFGs). Label each statement/node in the graph with a unique number, e.g., the line number in the source program, to help with the following subtasks. Do not include end in theCFG.





Check your result: The number of nodes (including entry and exit nodes) plus the number of edges (including edges from/to entry/exit nodes) should be 19.

2.2 Transfer Function [10 points]

First, write down the domain of the *Live Variables* analysis for the given program, i.e., complete the following set.

Domain: $\{$ $\}$

Next, fill the following table with the *gen* and *kill* sets of each statement in the program. For the first column, use the numbers from the CFG for identifiying statements.

Statements	$gen(s)$	$kill(s)$

2.3 Solving Data Flow Equations [16 points]

Now, use the iterative algorithm from the lecture to solve the data flow equations for each statement in the program. You do not need to write down the data flow equations themselves here; you can iteratively fill up the second and third column of the table below during solving.

Statements	$LV_{entry}(s)$	$LV_{exit}(s)$

2.4 Understanding [4 points]

Finally, to show your wider understanding of the applications of this data flow analysis, answer the following question: How could a compiler use the results of *Live Variables* analysis to optimize the original program? (1-2 sentences as answersuffices.)