**Project 3: Liveness Analysis Report**

Group member: Zixu Cao**,** Chuanye Xiong

* **Code summary**

Basically, the algorithm can be divided into three major parts. The first part is to calculate the VarKill and UEVar for every block in the CFG. The second part is iterating for the liveout variables for every block in the CFG until the liveout converge (won’t change any more) for every block. The third part is based on the data we already have and print the UEVAR, VARKILL and LIVEOUT in the right format.

* **Basic data structure**

The major data structure we are using is three hash tables:

1. block\_UEVar, in which key is the block name and value is UEVar.
2. block\_VarKill, in which key is the block name and value is VarKill.
3. block\_LiveOut, in which key is the block name and value is LiveOut.

Each hash table correlating the block name (key is string) with the upward exposed variables (string vector) in the single block, variables killed in the single block(string vector) and Live out variables (string vector) from the single block.

* **Implementation details**

**A screenshot of a computer

Description automatically generated with medium confidence**

Figure 1. code part1

The code part 1 (figure 1) aiming at calculate VarKill and UEVar for every block. We traverse all the block, and in each block, we traverse each instruction.

If the instruction is loading instruction: then we following the algorithm(Figure 2), provided in the ppt. if the var not in the VarKill set and also not in the UEVar set. We insert it in the UEVar set.

Text

Description automatically generated

Figure 2. the algorithm for compute UEVar and VarKill set

Text

Description automatically generated

Figure 3. code part2

As is shown in Figure3, If the instruction is store, var1, var2 all has chance to be the real variable. So we follow the logic above (in Figure 2) to insert the variable to the UEVar set.

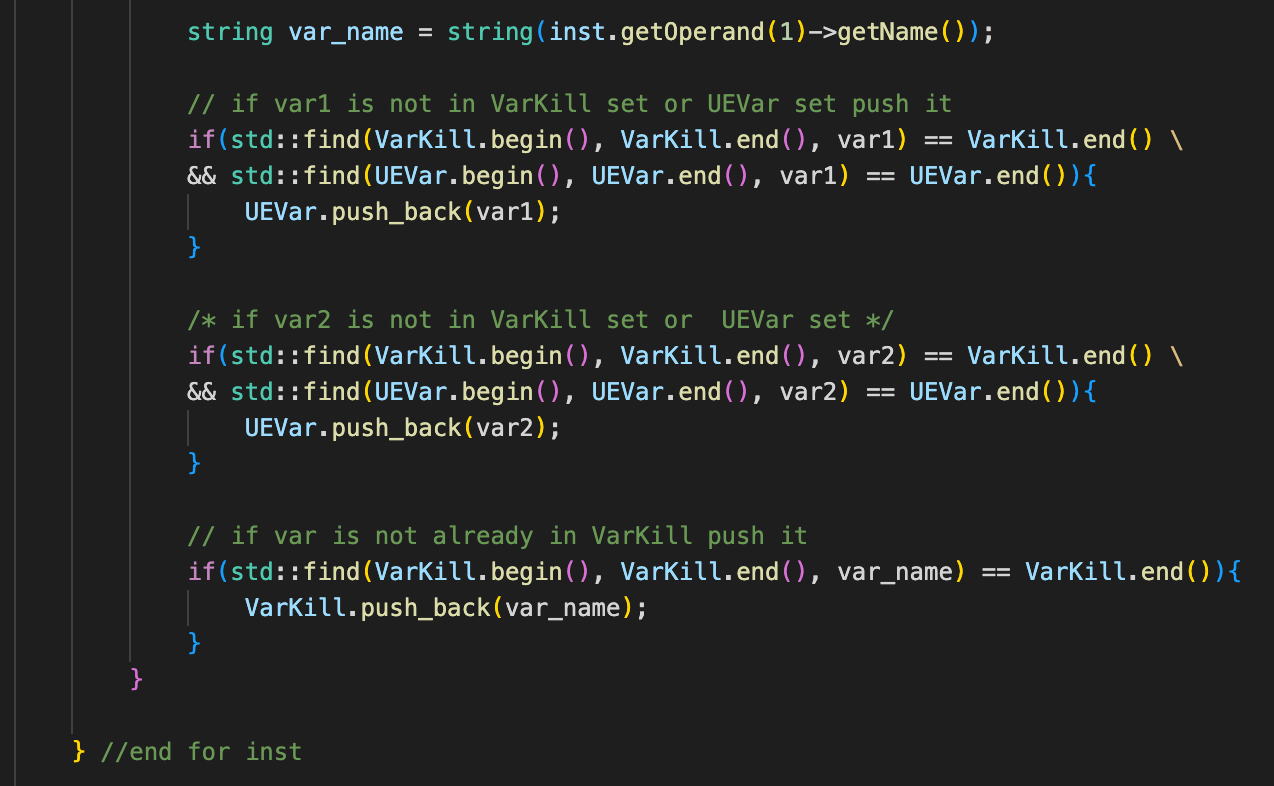


Figure 4. code part3

As we can see in the Figure 4, var\_name is the element that we must killed. So we check if it’s not in the VarKill set, we push it into the VarKill set. For example: as for the instruction x← y op z. var1 is the y, var2 is the z and var\_name is x.

Text

Description automatically generated

Figure 5. code part4

For each basic block, we get the data of the UEVar set and the VarKill set and save them to the corresponding hash table. (block\_UEVar and block\_VarKill). Inside the hash table, the key is the block name and value is the UEVar set or VarKill set.

Next we will do the iterations to get the LiveOut set for every block based on the VarKill and UEVar we already have.

Graphical user interface, text, website

Description automatically generated

Figure 6. code part5

Before the iteration, we initialize the LiveOut set for every basic block to be empty set, as is shown in Figure 6.

We follow the algorithm told in the class, as shown in the Figure 6 to iterate the LiveOut set for every basic block. Although go through all the blocks in the CFG is not the most efficient method for the liveout analysis, but it’s a useful and simple way to solve the task.

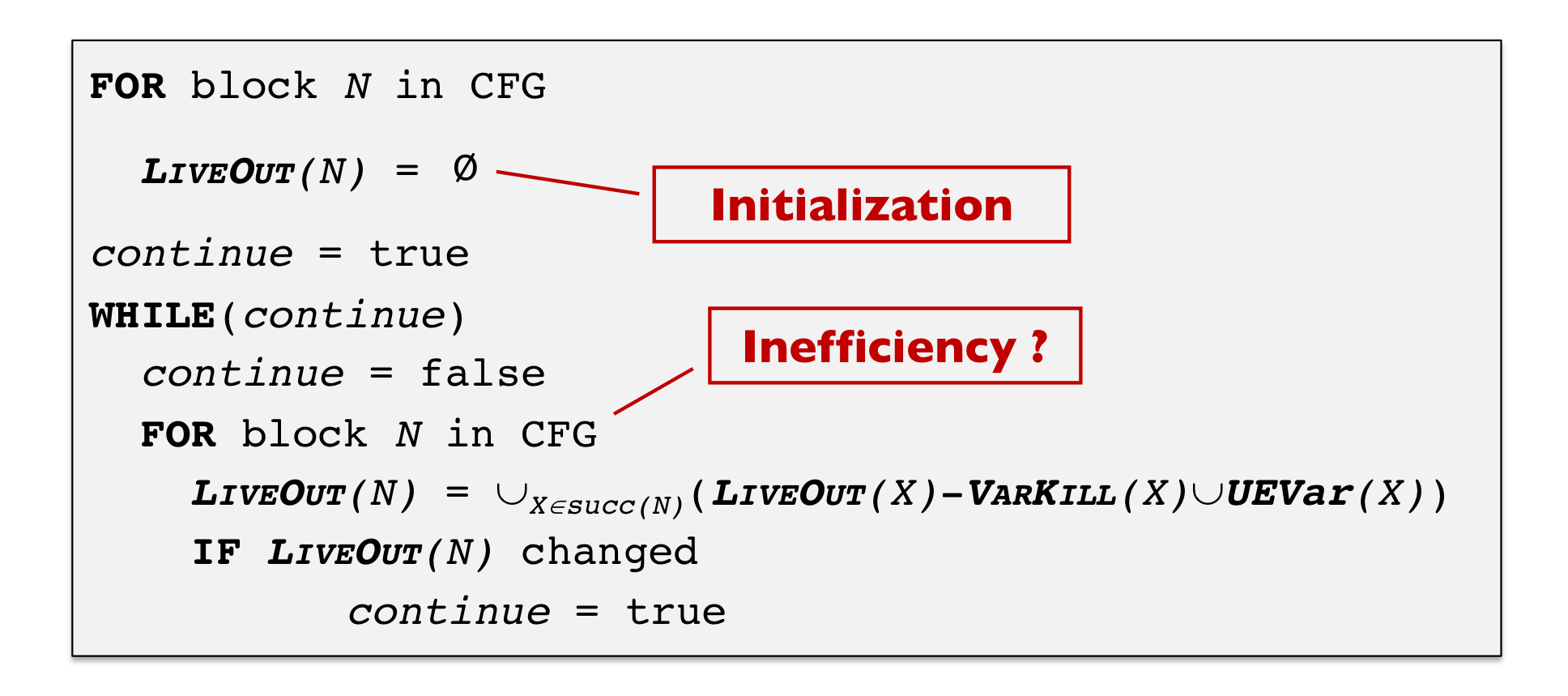


Figure 7. algorithm for iterating the Liveout set for each basic block

Text

Description automatically generated

Figure 8. code part 6

In the code part 6 above we first set continue iteration to be true. The ending iteration criterion is that the LiveOut set for all the block won’t change any more. the while(cont) is to tell the iteration proceed or not. The outer for loop is traverse for all the basic block in the CFG and the inner for loop is traverse for the all the successor blocks for a certain block in the CFG. For each successor, we compute the (LiveOut(X) – VarKill(X)∪UEVar(X)).

As is shown in Figure 9, After we traverse all the successors, we take the union UX∈successor (LiveOut(X) – VarKill(X)∪UEVar(X)), which would be the LiveOut set for certain block N in current iteration. We save that into the union\_succ in the code.

Text

Description automatically generated

Figure 9. code part 7

We get the previous iteration Liveout set data from the hash table (liveOut in the code), and compare with the current step LiveOut set data (union\_succ). If its not the same, we continue iteration while If it’s the same, we end the iteration. (Figure 10)

**Text

Description automatically generated**

Figure 10. code part 8

In the end, we update the current LiveOut set data and save it into the hash table, as is shown in the figure 11.

Graphical user interface, text

Description automatically generated

Figure 11. code part 9

When the iteration is done, we get the LiveOut set data for every block and saved that into the hash table.

The last part of our code is just retrieve the data from the hash table and print out the data. We traverse through all the block, for each block, we print out the block name, UEVAR set, VARKILL set and LIVEOUT set in the right format, as is shown in the Figure 12.

Text

Description automatically generated

Figure 11. code part 10

* **Implementation results**

1. For test1.c, we get the output:

Text

Description automatically generated

1. For test2.c, we get the output:

Text

Description automatically generated

1. For test3.c, we get the output:

Text

Description automatically generated

1. For test4.c, we get the output:

Text

Description automatically generated

In summary, we have successfully the Liveness analysis for all the test case.