CS 201

Project 3: Liveness Analysis

Report

Huy Tran & Rajat M. Jain

Algorithm’s Summary

The approach to perform Liveness Analysis on a Basic Block is as follow:

1. Our first task is to iterate through all the basic blocks and fetch each basic block in the CFG.
2. From each basic block, we not only store the basic block but we also store it’s name and blockId for tracking purposes.
3. Now that we receive information about the basic block, our next task to iterate through the instructions executed in the basic block.
4. We then store the 2 operands used in the instruction.
5. This is followed by checking the nature of our instruction which could be load, store, addition, subtraction, multiplication, or division.
6. For these instructions we add the operands on the right side of the ‘=’ to our Upward Exposed Variables and the variable on the left of ‘=’ to our Variable Kill Set.
7. After constructing our UEVar, VarKill we can proceed to compute the LiveOut based on the formula:



1. To compute the LiveOut we iterate through all the blocks we saved for our reference and use the successors of the block to apply the formula.
2. Finally, we print out by iterating through the structure we created to keep track of the UEVar, VarKill and LiveOut of each block.

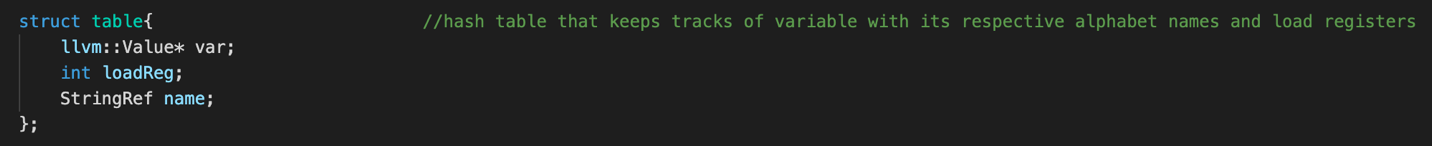
Data Structures used:

1. blockContent : A structure defined to store the BasicBlock itself and important variables like liveOut, ueVar and varKill in the form of StringRef. We also give each block a blockId and a list of its successor basic blocks.

Text

Description automatically generated

1. table: This is a hash table used to keep track of the variable with its respective loading register used for operations.



1. Indexing and tracking variables: We define a few more variables to assist our algorithm in indexing and condition checking.

Text

Description automatically generated

Explanation:

1. Implementation of computing UEVar and VarKill: We loop through the basic blocks and first increment the blockCount and then use blockCon of type blockContent to store the basic block itself and it’s name and blockId which is the blockCount. The inner loop then iterates through all the instructions in the basic block and fetches the operands for us.

Text

Description automatically generated

The statements in the block when the instruction is a load instruction is to store variable address, name and loading register in hash table. Loading register will always start %0 and go on regard to the number of load instruction. We want to keep track of the load register because operation instruction only use these registers as operands.

The if statement in the block when the instruction is a store instruction is for storing after operations. For example, c = a + b where first we load a and b, after that we add a and b, and finally the result which is c will be stored, else statement is for a special condition where for example, a = b which is load and then store.

We then insert the UEVar to their respective basic block by accessing blockCon[blockCount] when the instruction is a binary operation.

Text

Description automatically generated

1. Impementation of LiveOut: To compute the LiveOut of a block we follow the designated formula by first taking the difference between the LiveOut(X) and VarKill(X) and then union it with UEVar(X). We keep a track of the successors to implement the formula.

Text

Description automatically generated

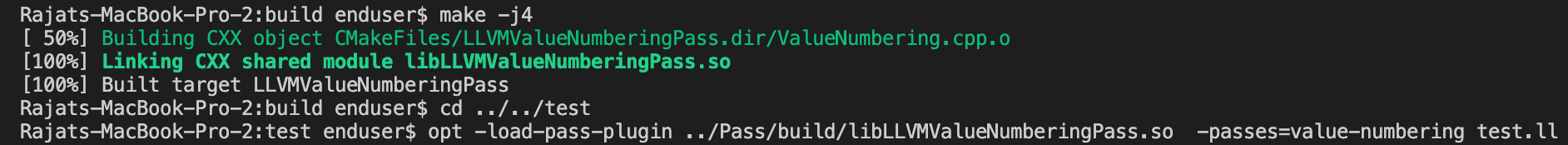
1. Printing the output: Ultimately we use errs() to print the output by iterating through blockCon and accessing its ueVar, varKill and liveOut property.

Text

Description automatically generated

Commands:

The following are the commands to run the algorithm on test.c

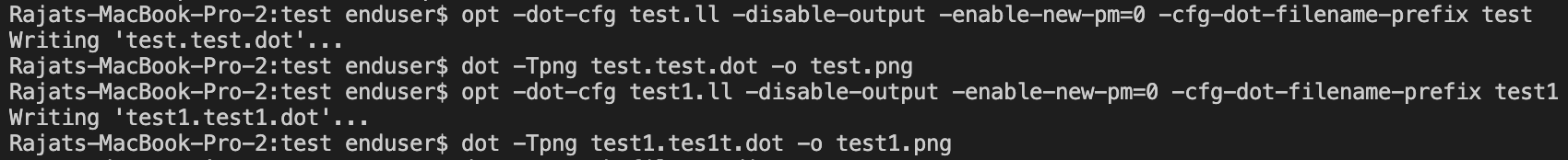


Output:

A picture containing text

Description automatically generated

We also generate the CFG for better reference by using the following commands:



Text

Description automatically generated

CFG for test.c:

Timeline

Description automatically generated