Project3 Report

Zhuocheng Shang 862188698 and Zizhuo Wang 862325178

1 Algorithm

Algorithm 1: Compute

```
1 VARKILL(N) = $\emptyset$
2 UEVar(N) = $\emptyset$
3 FOR i = 1 to k
4     assume $i_i$ is x $\leftarrow$ y op z
5     IF y $\notin$ VARKILL(N)
6         UEVar(N) = UEVAR(N) $\cup$ y
7     IF z $\notin$ VARKILL(N)
8         UEVar(N) = UEVAR(N) $\cup$ z
9     VARKILL(N) = VARKILL(N) $\cup$ x
```

Algorithm 2: Iterative

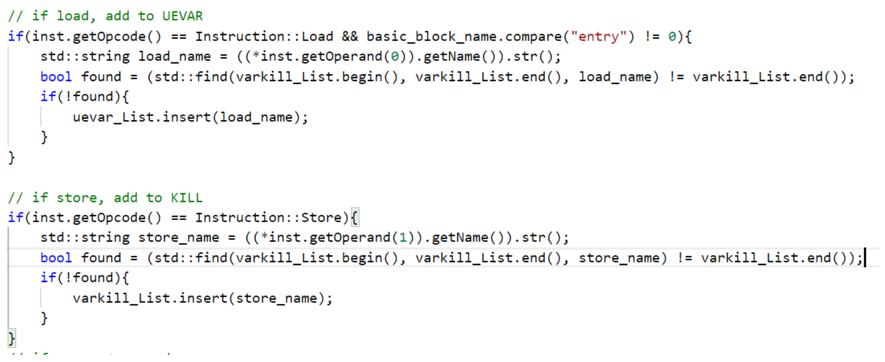
```
1 FOR block N in CFG
2     LIVEOUT(N) = $\emptyset$
3 UEVar and Varkill        *continue* = true
4 WHILE(*continue*)
5     *continue* = false
6     FOR block N in CFG
7         LIVEOUT(N) = $\cup_{X \in SUCC(N)}$ (LIVEOUT(X)-VARKILL(X) $\cup$ UEVar(X))
8         IF LIVEOUT(N) changed
9             *continue* = true
```

2 Data structures

There are three major map structures that are used: UEVAR_table,VARKILL_table and LIVEOUT_table. In each table, the keys are the names of the basic blocks and the values are the UEVar sets, the VARkill sets or the Liveout sets of the corresponding basic blocks.

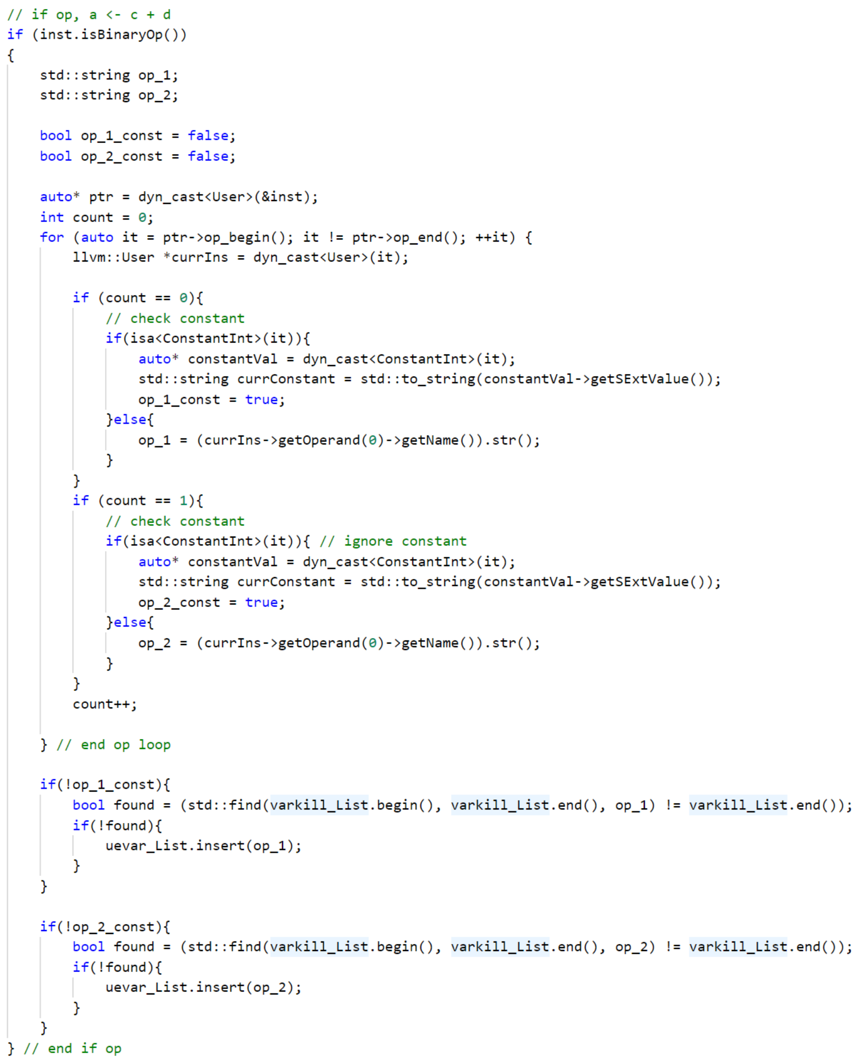
3 Implementation details

First, we compute UEVar and Varkill for each basic block. If an instruction is a load instruction, then we add the operand to UEVar. If an instruction is a store instruction, then we add the operand to Varkill.



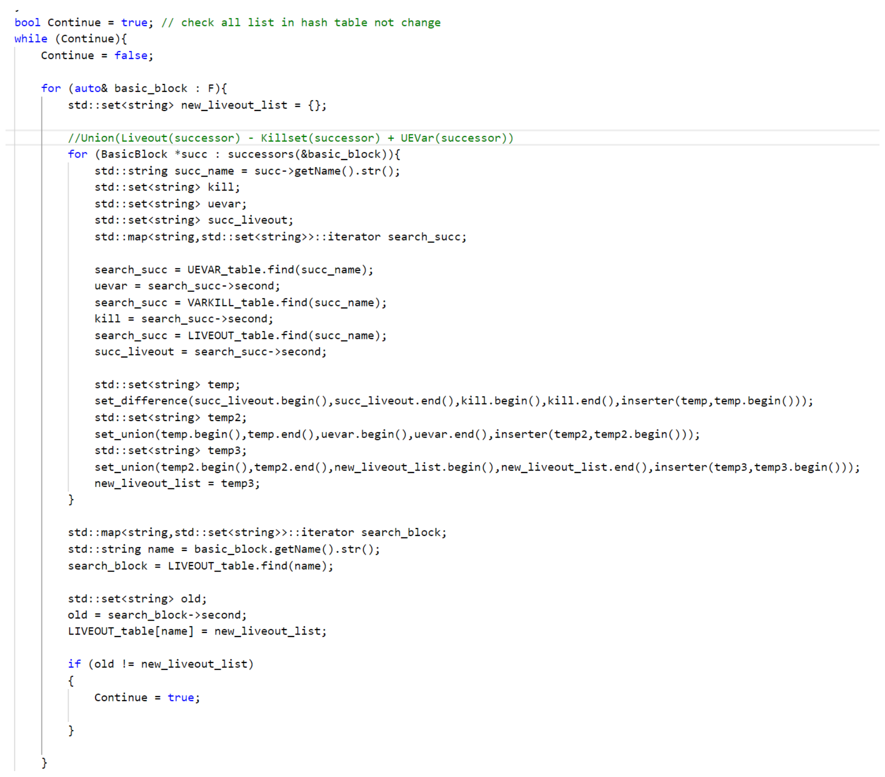
```
// if load, add to UEVAR
if(inst.getOpcode() == Instruction::Load && basic_block_name.compare("entry") != 0){
    std::string load_name = ((*inst.getOperand(0)).getName()).str();
    bool found = (std::find(varkill_List.begin(), varkill_List.end(), load_name) != varkill_List.end());
    if(!found){
        uevar_List.insert(load_name);
    }
}

// if store, add to KILL
if(inst.getOpcode() == Instruction::Store){
    std::string store_name = ((*inst.getOperand(1)).getName()).str();
    bool found = (std::find(varkill_List.begin(), varkill_List.end(), store_name) != varkill_List.end());
    if(!found){
        varkill_List.insert(store_name);
    }
}
```

If an instruction is a computing instruction, we firsr check if the operands is a constant. If not, check if they are in the Varkill list. If they are also not in it, we will add them into the UEVar list.



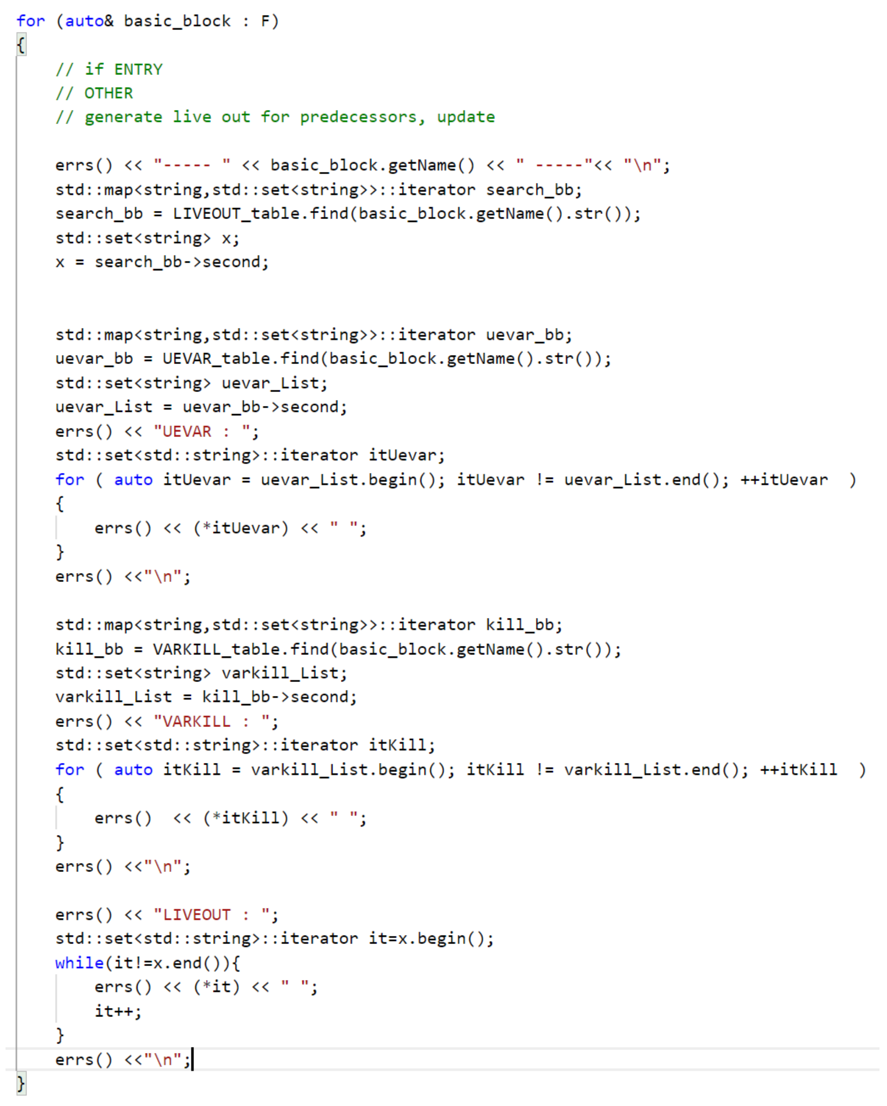
```
// if op, a <- c + d
if (inst.isBinaryOp())
{
    std::string op_1;
    std::string op_2;

    bool op_1_const = false;
    bool op_2_const = false;

    auto* ptr = dyn_cast<User>(&inst);
    int count = 0;
    for (auto it = ptr->op_begin(); it != ptr->op_end(); ++it) {
        llvm::User *currIns = dyn_cast<User>(it);

        if (count == 0){
            // check constant
            if(isa<ConstantInt>(it)){
                auto* constantVal = dyn_cast<ConstantInt>(it);
                std::string currConstant = std::to_string(constantVal->getSExtValue());
                op_1_const = true;
            }else{
                op_1 = (currIns->getOperand(0)->getName()).str();
            }
        }
        if (count == 1){
            // check constant
            if(isa<ConstantInt>(it)){ // ignore constant
                auto* constantVal = dyn_cast<ConstantInt>(it);
                std::string currConstant = std::to_string(constantVal->getSExtValue());
                op_2_const = true;
            }else{
                op_2 = (currIns->getOperand(0)->getName()).str();
            }
        }
        count++;

    } // end op loop

    if(!op_1_const){
        bool found = (std::find(varkill_List.begin(), varkill_List.end(), op_1) != varkill_List.end());
        if(!found){
            uevar_List.insert(op_1);
        }
    }

    if(!op_2_const){
        bool found = (std::find(varkill_List.begin(), varkill_List.end(), op_2) != varkill_List.end());
        if(!found){
            uevar_List.insert(op_2);
        }
    }
} // end if op
```

After finding the UEVar and Varkill for all the basic blocks, we compute the liveout sets for each basic block using iteration. In each round, we update the liveout sets for each basic block with the formula LIVEOUT(N) = $\cup_{X \in SUCC(N)}$ (LIVEOUT(X)-VARKILL(X) $\cup$ UEVar(X)). When the liveout sets of each basic block no longer change, the iteration stops.



```cpp
bool Continue = true; // check all list in hash table not change
while (Continue){
    Continue = false;

    for (auto& basic_block : F){
        std::set<string> new_liveout_list = {};

        //Union(Liveout(successor) - Killset(successor) + UEVar(successor))
        for (BasicBlock *succ : successors(&basic_block)){
            std::string succ_name = succ->getName().str();
            std::set<string> kill;
            std::set<string> uevar;
            std::set<string> succ_liveout;
            std::map<string,std::set<string>>::iterator search_succ;

            search_succ = UEVAR_table.find(succ_name);
            uevar = search_succ->second;
            search_succ = VARKILL_table.find(succ_name);
            kill = search_succ->second;
            search_succ = LIVEOUT_table.find(succ_name);
            succ_liveout = search_succ->second;

            std::set<string> temp;
            set_difference(succ_liveout.begin(),succ_liveout.end(),kill.begin(),kill.end(),inserter(temp,temp.begin()));
            std::set<string> temp2;
            set_union(temp.begin(),temp.end(),uevar.begin(),uevar.end(),inserter(temp2,temp2.begin()));
            std::set<string> temp3;
            set_union(temp2.begin(),temp2.end(),new_liveout_list.begin(),new_liveout_list.end(),inserter(temp3,temp3.begin()));
            new_liveout_list = temp3;
        }

        std::map<string,std::set<string>>::iterator search_block;
        std::string name = basic_block.getName().str();
        search_block = LIVEOUT_table.find(name);

        std::set<string> old;
        old = search_block->second;
        LIVEOUT_table[name] = new_liveout_list;

        if (old != new_liveout_list)
        {
            Continue = true;

        }

    }
```

Finally, we output the result of each basic block one by one.



```
for (auto& basic_block : F)
{
    // if ENTRY
    // OTHER
    // generate live out for predecessors, update

    errs() << "----- " << basic_block.getName() << " -----"<< "\n";
    std::map<string,std::set<string>>::iterator search_bb;
    search_bb = LIVEOUT_table.find(basic_block.getName().str());
    std::set<string> x;
    x = search_bb->second;


    std::map<string,std::set<string>>::iterator uevar_bb;
    uevar_bb = UEVAR_table.find(basic_block.getName().str());
    std::set<string> uevar_List;
    uevar_List = uevar_bb->second;
    errs() << "UEVAR : ";
    std::set<std::string>::iterator itUevar;
    for ( auto itUevar = uevar_List.begin(); itUevar != uevar_List.end(); ++itUevar  )
    {
        errs() << (*itUevar) << " ";
    }
    errs() <<"\n";

    std::map<string,std::set<string>>::iterator kill_bb;
    kill_bb = VARKILL_table.find(basic_block.getName().str());
    std::set<string> varkill_List;
    varkill_List = kill_bb->second;
    errs() << "VARKILL : ";
    std::set<std::string>::iterator itKill;
    for ( auto itKill = varkill_List.begin(); itKill != varkill_List.end(); ++itKill  )
    {
        errs()  << (*itKill) << " ";
    }
    errs() <<"\n";

    errs() << "LIVEOUT : ";
    std::set<std::string>::iterator it=x.begin();
    while(it!=x.end()){
        errs() << (*it) << " ";
        it++;
    }
    errs() <<"\n";
}
```