Project 3

Tsung-Wei, Liu
Yin-Yu, Chang

Link for video:
https://drive.google.com/file/d/1DE85qDhEKsBKuul5GACusQde9YaEl7uK/view?usp=share_link

Algorithm:
(1) VarKill(X)
Using a global vector as tmp to store the liveout from successor. When the element in the VarKill set appears in this liveout then kill it.

(2) UEVar(X)
Using a global vector as tmp to store the liveout from successor. When the element in the UEVar set appears in this liveout then kill it.

(3) LiveOut(X)
Using a global vector as tmp to store the liveout from successor. When the element in the liveout does not appear in the parent liveout then add it to the parent liveout.

(4) Liveness analysis
Initialize the liveout set and walk through all of liveout set. If there is change of liveout set, it would continue until it does not have any change.

Data Structure:

```
struct blockUnit{
  string blockName ;
  string address ; // used to store address of block
  bool changed ;
  vector <string> buffer ;
} ;
vector <blockUnit> UEVar, VarKill, LiveOut ;
```

Use the struct which is called blockUnit to store all of UEVar, VarKill and LiveOut.

Explanation for liveness analysis:

```
void setLiveOut( string blockName, string suc_address ) {
  // blockName current basic block name
  // suc_address successor address
  int posOfSuc = findPosOfSucBlock( suc_address ) ;
  // set the suceesor live out
  gSuc = LiveOut[posOfSuc].buffer ;
  killLiveOut( blockName, posOfSuc ) ; // kill
  unionUE( blockName, posOfSuc ) ;  // union uevar

  int posOfBlock = findPosOfBlock( blockName ) ;
  for ( int i = 0 ; i < gSuc.size() ; i++ ) {
    if ( !isInSet( LiveOut[posOfBlock].buffer, gSuc[i] ) ) {
      LiveOut[posOfBlock].buffer.push_back( gSuc[i] ) ;
      LiveOut[posOfBlock].changed = true ;
    } // if
  } // for

} // setLiveOut()
```

We use gSUc which is global vector of blockUnit to store the liveout from the successor. The killLiveout function and unionUE function do the operation of kill and union. As long as two functions finish their job, we can assign the set to the specific successor's liveout and set the boolean changed.

```
bool go = true ;
string blockName ;
while( go ) {
  go = false ;
  for ( auto& basic_block : F ) {
    StringRef bbName( basic_block.getName() ) ; // basic block name
    // errs() << bbName.str() << "\n" ;
    for(BasicBlock* child : successors(&basic_block)) {
      stringstream ss ;
      ss << child ;
      string address = ss.str() ; // get basic block string of address
      // showSuccessor( address, bbName.str() ) ;
      setLiveOut( bbName.str(), address ) ;

      if ( liveOutChanged() ) {
        go = true ;
        int posOfBlock = findPosOfBlock( bbName.str() ) ;
        LiveOut[posOfBlock].changed = false ;
      } // if
    } // for
  } // for

} // while()
```

The liveOutChanged is the boolean function that is used to check whether the liveout set has been changed. If it changed, it would set the boolean go as true and make the loop to be continued.

Explanation for UEVar and VarKill:

```
if ( inst.getOpcode() == Instruction::Load ) {
  // add it to the UEVar list
  // check for the VarKill list
  // if it is in the varkill list than dont add it to UEVar
  gFlag = true ;
  GetToken( ss, gBuffer_UEVar, gFlag ) ;
} // if
if ( inst.getOpcode() == Instruction::Store ) {
  // add it to the VarKill list
  gFlag = false ;
  GetToken( ss, gBuffer_VarKill, gFlag ) ;
} // if
```

```
if ( inst.isTerminator() ) {
  if ( inst.getOpcode() == Instruction::Br || inst.getOpcode() == Instruction::Ret ) {
    if ( UEVar.size() == 0 && VarKill.size() == 0 ) {
      // put entry in the buffer
      blockUnit uevar, varkill ;
      uevar.buffer = gBuffer_UEVar ;
      varkill.buffer = gBuffer_VarKill ;

      // set the Block name
      setBlockName( uevar, varkill, "entry" ) ;
      // push vector to each list
      UEVar.push_back( uevar ) ;
      VarKill.push_back( varkill ) ;

      gBuffer_UEVar.clear() ;
      gBuffer_VarKill.clear() ;
      GetToken( ss, gBuffer, gFlag ) ;
      // get the first branch instruction block name
      setQueue( gQ, gBuffer ) ;
      gBuffer.clear() ;
    } // if
```

When we meet Load and Store instructions, we call the GetToken function to get a string of instructions. Once we meet branching and return then we can set up our UEVar and VarKill. The image shows how we settle our two sets. Last but not least, we have a gQ vector to record the block name. That is because when we meet the branching, we will get the block name in the next two blocks or the future. Therefore, when we are in the next iteration, we have the actual block name and we can settle the block name to UEVar and VarKill.