Liveness Analysis Project Report

The project breaks down to two pars, one for finding VarKill and UEVar, another is to find LiveOut.

First we create a table to store VarKill, UEVar and LiveOut for each BasicBlock.

```
map<BasicBlock*, map<string, set<Value*>>> table = {};
```

The table uses BasicBlock pointer as key and contains another map for storing VarKill, UEVar, and LiveOut. We choose set structure because it's easy to do union and subtraction operations.

Then we initialize the table with empty sets.

```
for(auto& basic_block : F){
  map<string, set<Value*>> blockInfo;
  set<Value*> UEVar;
  set<Value*> VarKill;
  set<Value*> LiveOut;
  blockInfo.insert(make_pair(“UEVar”, UEVar));
  blockInfo.insert(make_pair("VarKill", VarKill));
  blockInfo.insert(make_pair(“VarKill”, LiveOut));
  table.insert(make_pair(&basic_block, blockInfo));
}
```

After the table is initialized, we compute the VarKill and UEVar for each basic block.

```
for(auto& basic_block: F){
  findVarKillAndUEVar(table, basic_block);
}
```

Inside the function we process the instruction one by one.

```
void findVarKillAndUEVar(map<BasicBlock*, map<string, set<Value*>>> &table,
BasicBlock &bb){
    for(auto& inst: bb){
      map<string, set<Value*>> *bbInfo = &table[&bb];
      set<Value*> *VarKill = &(*bbInfo)[“VarKill”];
      set<Value*> *UEVar = &(*bbInfo)["UEVar"];

      // load instruction
      if(inst.getOpcode()==31){
        Value* v = inst.getOperand(0);
        if(VarKill->find(v) == VarKill->end()){
          UEVar->insert(v);
        }
      }

      // set instruction
      if(inst.getOpcode()==32){
        VarKill->insert(inst.getOperand(1));
      }
    }
  }
```

When we found a load instruction, we look if it is in VarKill, if not, add to UEVar, and we put every variables that are set into VarKill.

After this part is done, we find LiveOut by calling `findLiveout(table, F);`.

```

void findLiveout(map<BasicBlock*, map<string, set<Value*>>> &table, Function
&F){
    bool flag = true;
    while(flag){
      flag = false;
      for(auto& bb: F){

        set<Value*> dest, dest1;

        map<string, set<Value*>> *bbInfo = &table[&bb];
        set<Value*> prev = (*bbInfo)[“LiveOut”];
        set<Value*> *VarKill = &(*bbInfo)[“VarKill”];
        set<Value*> *UEVar = &(*bbInfo)[“UEVar”];
        set<Value*> *LiveOut = &(*bbInfo)[“LiveOut”];
        for(BasicBlock* Succ: successors(&bb)){
          map<string, set<Value*>> *SuccBbInfo = &table[Succ];
          set<Value*> *SuccVarKill = &(*SuccBbInfo)["VarKill"];
          set<Value*> *SuccUEVar = &(*SuccBbInfo)["UEVar"];
          set<Value*> *SuccLiveOut = &(*SuccBbInfo)["LiveOut"];

          set_difference(SuccLiveOut->begin(), SuccLiveOut->end(), SuccVarKill-
>begin(), SuccVarKill->end(), inserter(dest, dest.begin()));
          set_union(dest.begin(), dest.end(), SuccUEVar->begin(), SuccUEVar-
>end(), inserter(dest1, dest1.begin()));
          set_union(dest1.begin(), dest1.end(), LiveOut->begin(), LiveOut-
>end(), inserter(*LiveOut, LiveOut->begin()));
          dest.clear(); dest1.clear();
        }
        if(!std::equal(prev.begin(), prev.end(),LiveOut->begin(), LiveOut-
>end())){
          flag = true;
        }
      }
    }
  }
```

For every iteration, we check the flag, the flag will be true if there are any changes happened to LiveOut, false otherwise. Then we loop over each BasicBlock. We compute LiveOut by using formula:

$$\textit{LiveOut}(n) = \cup_{x \in succ(n)}(\textit{LiveOut}(x) - \textit{VarKill}(x) \cup \textit{UEVar}(x))$$

Which is in this part of loop:

```
for(BasicBlock* Succ: successors(&bb)){
          map<string, set<Value*>> *SuccBbInfo = &table[Succ];
          set<Value*> *SuccVarKill = &(*SuccBbInfo)[“VarKill”];
          set<Value*> *SuccUEVar = &(*SuccBbInfo)[“UEVar"];
          set<Value*> *SuccLiveOut = &(*SuccBbInfo)[“LiveOut”];

          set_difference(SuccLiveOut->begin(), SuccLiveOut->end(), SuccVarKill-
>begin(), SuccVarKill->end(), inserter(dest, dest.begin()));
          set_union(dest.begin(), dest.end(), SuccUEVar->begin(), SuccUEVar-
>end(), inserter(dest1, dest1.begin()));
          set_union(dest1.begin(), dest1.end(), LiveOut->begin(), LiveOut-
>end(), inserter(*LiveOut, LiveOut->begin()));
          dest.clear(); dest1.clear();
        }
```

After we are done with each successors of the current BasicBlock, we check if there are any changes.

```
if(!std::equal(prev.begin(), prev.end(),LiveOut->begin(), LiveOut->end())){
          flag = true;
        }
```

Then we are done.

The last part is to output the results:

```
for(auto& basic_block: F){
      errs() << "—— ";
      basic_block.printAsOperand(errs(), false);
      errs() << " ——\n";
      map<string, set<Value*>> bbInfo = table[&basic_block];
      set<Value*> VarKill = bbInfo["VarKill"];
      set<Value*> UEVar = bbInfo["UEVar"];
      set<Value*> LiveOut = bbInfo["LiveOut"];

      errs() << "UEVar: ";
      for(auto it = UEVar.begin();it != UEVar.end();++ it){
        errs() << (**it).getName() << " ";
      }
      errs() << "\n";

      errs() << "VarKill: ";
      for(auto it = VarKill.begin();it != VarKill.end();++ it){
        errs() << (**it).getName() << " ";
      }
      errs() << "\n";



      errs() << "LiveOut: ";
      for(auto it = LiveOut.begin();it != LiveOut.end();++ it){
        errs() << (**it).getName() << " ";
      }
      errs() << "\n";

    }

    return false;
  }
```