COMS W4115: Programming Assignment 5

Data flow Analysis

Logistics

1. **Announcement Date:** November 11th, 2019
2. **Due Date:** Dec. 2nd, 2019 by 11:59pm. No extension!!
3. **Total Points:** 100

LLVM Pass for Liveness Analysis

1. Generate LLVM IR and SSA(single static assignment).

   (a) Generate LLVM IR for an C program.

   ```
   ./llvm-project/build/bin/clang -O -emit-llvm -c example.c
   ./llvm-project/build/bin/llvm-dis example.bc
   ```

   (b) Generate SSA(Single Static Assignment).

   ```
   llvm-project/build/bin/opt -mem2reg example.bc -o example.bc
   ```

2. Write a pass to perform liveness analysis by iterating through the flow graph backwards. You are provided with a starting file "liveness.cpp" and you should implement the liveness analysis algorithm in this cpp file. You may need to leverage a use/gen set, def/kill set, live-in set and live-out set to implement the analysis. The following equations are necessary for implementing the liveness analysis in the pass.

$$IN[n] = USE[n] \cup (OUT[n] - DEF[n])$$

$$OUT[n] = \bigcup_{s \in succ[n]} IN[s]$$

   The pass should take an IR in SSA form as input and output the **live-out set** after each instruction. There are two example inputs and outputs at the end of this file.

3. Notes:

   (a) Get used variable for an instruction

   ```
   User::op_iterator opnd = I.op_begin(), opE = I.op_end();
   for (; opnd != opE; ++opnd) {
       Value* val = *opnd;
       if (isa<Instruction>(val) || isa<Argument>(val)) {

       }
   }
   ```

(b) Get defined variable for an instruction

```
Instruction *pI = &I;
Value* p = cast<Value> (pI);
```

Return instruction and branch instruction should be handled specially.

(c) Φ instructions
Φ instructions are not real instructions and need to be handled specially by the liveness analysis. Each operand of a Φ instruction is only live along the edge from the corresponding predecessor block. In the attached two examples, the first one does not have Φ instructions while the second one have Φ instructions. You should try to answer what could be the output if Φ is not handled specially, and then try to come up with an approach to handle Φ instructions.

Please rename your cpp file to {UNI}-liveness.cpp and submit it. During grading, we will first rename your cpp file to liveness.cpp and run it as follows.

```
./build/bin/opt  -load ./build/lib/LLVMprog5.so -liveness < example.bc
```

Please define the created class name and registered LLVM pass name accordingly for full grade.

We may also test your program on other C programs.

Submission Guide

You can either work in pair or work by yourself.

Please submit the followings:

1. You are required to submit **{UNI}-liveness.cpp**(100 points). {UNI} means your UNI number.
2. Submit an extra file **contribution.txt** describing each of your contribution if you work in pair.

Example input and output

1. Example1

```
//C program
int g, h;
int test(int condition) {
  int x;
  if (condition==1)
     x = g;
  else
     x = h;
  return x;
}
//SSA
define dso_local i32 @test(i32 %condition) local_unnamed_addr #0 {
entry:
  %cmp = icmp eq i32 %condition, 1
  %g.val = load i32, i32* @g, align 4
  %h.val = load i32, i32* @h, align 4
  %x.0 = select i1 %cmp, i32 %g.val, i32 %h.val
  ret i32 %x.0
}
```

Expected output from your pass:

```
Instruction:   %cmp = icmp eq i32 %condition, 1 -->
 liveness OUT: {%cmp }
Instruction:   %g.val = load i32, i32* @g, align 4 -->
 liveness OUT: {%cmp %g.val }
Instruction:   %h.val = load i32, i32* @h, align 4 -->
 liveness OUT: {%cmp %g.val %h.val }
Instruction:   %x.0 = select i1 %cmp, i32 %g.val, i32 %h.val -->
 liveness OUT: {%x.0 }
Instruction:   ret i32 %x.0 -->
 liveness OUT: {}
```

2. Example2

```
//C program
int sum(int a, int e){
    int res = 0;
    while (a < e){
        int b = a + 1;
        a = b*2;
    }
    return res+a;
}

//SSA
define dso_local i32 @sum(i32 %a, i32 %e) local_unnamed_addr #0 {
entry:
  %cmp6 = icmp slt i32 %a, %e
  br i1 %cmp6, label %while.body, label %while.end

while.body:                                       ; preds = %entry, %while.body
  %a.addr.07 = phi i32 [ %mul, %while.body ], [ %a, %entry ]
  %add = shl i32 %a.addr.07, 1
  %mul = add i32 %add, 2
  %cmp = icmp slt i32 %mul, %e
  br i1 %cmp, label %while.body, label %while.end

while.end:                                        ; preds = %while.body, %entry
  %a.addr.0.lcssa = phi i32 [ %a, %entry ], [ %mul, %while.body ]
  ret i32 %a.addr.0.lcssa
}
```

Expected output from your pass:

```
Instruction:   %cmp6 = icmp slt i32 %a, %e -->
 liveness OUT: {%a %e %cmp6 }
Instruction:   br i1 %cmp6, label %while.body, label %while.end -->
 liveness OUT: {%a %e }
Instruction:   %a.addr.07 = phi i32 [ %mul, %while.body ], [ %a, %entry ] -->
 liveness OUT: {%e %a.addr.07 }
Instruction:   %add = shl i32 %a.addr.07, 1 -->
 liveness OUT: {%e %add }
Instruction:   %mul = add i32 %add, 2 -->
 liveness OUT: {%e %mul }
Instruction:   %cmp = icmp slt i32 %mul, %e -->
 liveness OUT: {%mul %cmp }
Instruction:   br i1 %cmp, label %while.body, label %while.end -->
 liveness OUT: {%mul }
Instruction:   %a.addr.0.lcssa = phi i32 [ %a, %entry ], [ %mul, %while.body ] -->
 liveness OUT: {%a.addr.0.lcssa }
Instruction:   ret i32 %a.addr.0.lcssa -->
 liveness OUT: {}
```