COMS W4115: Programming Assignment 6

Eliminate Dead Code

Logistics

1. **Announcement Date:** November 19^{th}, 2019
2. **Due Date:** December 15^{th}, 2019 by 11:59pm. No extension!!
3. **Total Points:** 100

LLVM Pass for Eliminate Dead Function

1. Generate LLVM IR for example1.c

   Run the following command to compile example1.c to LLVM bytecode object example1.bc.

   ```
   ./llvm-project/build/bin/clang -O0 -emit-llvm -c example1.c
   ./llvm-project/build/bin/llvm-dis example1.bc
   ```

2. Create an LLVM pass to eliminate functions without any call site.

   In this assignment, you are required to create an LLVM pass to eliminate any function definition in IR if there is no call site for this function.
   Note1: main function should not be eliminated.
   Note2: After eliminating some functions, some other functions may become having 0 call site, so you need to iterate this process until no more functions need to be eliminated.

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

   ```
   ./bin/opt  -load ./lib/LLVMprog6.so -efunc < example1.bc > optimized.bc
   /bin/llvm-dis optimized.bc
   ```

   We will look at *optimized.ll* to see if the dead functions are removed correctly to grade this part. 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 besides example1.c.

LLVM Pass for Eliminate Dead Instructions

1. Generate LLVM IR for example2.c

   Run the following command to compile example2.c to LLVM bytecode object example2.bc.

   ```
   ./bin/clang -O0 -Xclang -disable-O0-optnone -emit-llvm -c ./example2.c

   ./bin/llvm-dis example2.bc
   ```

2. Generate SSA(Single Static Assignment).

   ```
   llvm-project/build/bin/opt -mem2reg example2.bc -o ssa.bc
   ./bin/llvm-dis ssa.bc
   ```

3. Create an LLVM pass to eliminate dead instructions.

   In this assignment, you are required to create an LLVM pass to eliminate any instructions in SSA IR if the defined variable in this instruction becomes dead after this instruction. (Hint: defined variable not in living-out set.)You should leverage the live variable analysis in previous assignment.

   Note1: call instructions, terminators, return instructions or PHI instructions should not be removed.
   Note2: After eliminating some instructions, some other instructions may become dead, so you need to iterate this process until no more instructions need to be eliminated.
   Note3: If you use API *"isInstructionTriviallyDead"* to solve this problem, you can only get 15 points(total 50 points).

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

   ```
   ./bin/opt  -load ./lib/LLVMprog6.so -einst < ssa.bc > optimized.bc
   /llvm-project/build/bin/llvm-dis optimized.bc
   ./bin/llvm-dis optimized.bc
   ```

   We will look at *optimized.ll* to see if the dead instructions are removed correctly to grade this part. 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 besides example2.c.

Submission Guide

You can either work in pair or work by yourself.
Please submit the followings:

1. You are required to submit **{UNI}-functions.cpp**(50 points) and **{UNI}-instructions.cpp**(50 points). {UNI} means your UNI number.
2. Submit an extra file **contribution.txt** describing each of your contribution.