Part 2: Collecting Dynamic Instruction

Algorithm:

1. A basic block is a single-entry, single-exit section of code. So whether the program is running or not, the static analysis for a basic block is the same with the dynamic analysis for this block. The idea is to calculate the statistic for the basic block statically when the LLVM Pass optimize the target program and in the end of each basic block, the LLVM Pass we implement will insert a function call to a global function “merge”. When the program is running, every time the program comes across the inserted point, the program will call “merge” and the function will combine the statistic of this basic block to a global map. So probably a basic block can run several time, which means the statistic of this basic block will be merged into global map several time. Also in the termination of the program, we need to output the result. So when the LLVM Pass optimize the program, it also find the “main” function and then find the “return” statement and just before the “return” statement, LLVM Pass inserts a function call to “print” which can output the statistic result of this program. The two function “print” and “merge” are put in a file called merge.cpp.
2. At high level, one program and merge.cpp are passed to clang, and clang generates their LLVM IR. And then, the LLVM Pass will optimize the program’s LLVM IR. And then the compiler will link the modified program LLVM IR and merge LLVM IR together and then compile them into the executable.

Pseudocode for the LLVM Pass:

for each function in the module:

for each basic block in the module:

for each instruction in the basic block:

if the instruction is “return” and the function is “main”:

insert a function call to “print” the statistic of the program

increase the number of corresponding instruction

insert a function call to “merge” the statistic into global map in t he end of each basic block

We use FunctionType::get to construct the type of function we will insert. And Module::getOrInsertFunction to get the function from the function library of the target program and IRBuilder:: CreateCall to insert a function call. The big challenge we encountered is to pass a parameter when insert a function call. It’s extremely hard to pass a string, so we pass the a int parameter to our “merge” function and in our “merge” function.