Assignment 3 – Kaivan Wadia

This assignment was a great learning experience. I decided to use a composition architecture for creating the framework. This was a decision which cost me quite a lot at first especially since I was not very strong with template types and inheritance of template types in C++. Although once I got the architecture setup and compiling I found writing the actual pass very simple in comparison.

I first attempted Liveness which also was probably a bad choice considering Reaching definition might be an easier pass. The transfer function for liveness was pretty straightforward except I had to write various helper functions since I used vectors. In retrospect I think I should have just stuck with unordered_set and provided my own hash function. The meet function for liveness had to consider the case of phi functions. My first idea was to have a map for the phi information but then I switched it to just an iteration over the phi nodes while doing the meet. My reasoning for this was that in a basic block the number of phi functions would be significantly less than the number of total instructions therefore the overhead added would not be very significant.

Reaching definitions was relatively very simple as the meet was a straightforward union and the transfer was trivial too. As pointed out in the Piazza post there is confusion regarding the concept of reaching definition and so I wrote a solution that is probably overly conservative and propagates a reaching definition even beyond its final use. I had an idea to use def-use chains for finding if a use was the last use but was not confident that it would work and whether using def-use chains was allowed in the first place so I removed it from my solution.

My worklist is basically dependent on the direction of flow of the pass. In a backward analysis such as liveness I use the post-order to determine which basic block to analyze next. For forward analysis I use the reverse post order to determine the same.

During the assignment I discussed some cases and ideas with Mark, Steve and Alyssa. It was helpful to get different viewpoints on certain issues and also with difficult errors I was facing in C++ templates.

Overall I felt this assignment was a good exercise in not only learning how a pass works but more importantly how to make an abstract framework for all passes with maximum code re-use. I felt composition resulted in probably a more fragmented approach at first but greatly improved re-use of code and the fragmentation made a lot of sense at the end.

I should also mention that a –mergereturn and pass should be run on the test files before running any of my passes.