CSC410 Project 1 Report

**Approach**

1. Written Set Variables (check\_in1.py)

To collect the written set of each loop, WriteSetVisitor examines each assignment statement in the loop and collects the variable in the left side of the statement in a set. The variables were stored in a dictionary (self.collection) with the for loop node as the key and a set of the written variables as the value.

2. Reaching Definitions (check\_in2.py)

ReachingDefsVisitor first collects each assignment statement as potential reaching definitions before the loop. Inside a for loop, the for loop node is used as key in dictionaries that store potential reaching definitions and written variables of the loop. It collects the written variables. It also collects all assignment statements since they could be possible reaching definitions for subsequent for loops. For nested loops, the reaching definitions of the inner loop are the ones of the outer loop and its index update, if the index is used in the inner loop. Once the visitor is done visiting, only the dictionary entries corresponding to the loop’s written variables are kept. The rest are deleted.\

3a. Loop Analysis

LoopAnalyser.py is a python file that imports the code from each check-in. The analyser compiles all of the information collected within the loops and loop nests of each test input file and outputs this information to stdout. The LoopInfoCollection class initializes a FuncWriteSetVisitor, FuncReachingDefsVisitor, FuncReadSetVisitor, FuncIndexSetVisitor (which contains information about loop index sets, loop index update statements, and loop index vectors), and FuncLiveVariableVisitor for the minic ast associated with the c input file. After each visitor visits the minic ast, a \_\_str\_\_ method is used to return a string representing all of the information collected by each loop. all\_inputs\_tester.py is a tool which runs LoopAnalyser.py on the final 8 p1\_input[i].c files located in the project1inputs folder.

3b. Indexes and Updates to Indexes(check\_in3b4a.py)

To collect the index variables of each loop, IndexSetVisitor examines each loop nest for assignment statements. For each assignment statement, any assignment that has an Array Reference node on the right-hand side is examined and any Array Reference node that contains a variable subscript is added to the set of index variables corresponding to the outermost loop. The variables were stored in a set (self.indexset) with the outermost for loop node of the loop nest as the key and the index variables set as the value.

Once the set of index variables are collected for each loop nest, all statements that updated the index, including statements in the for() statement, sans the initialization, are stored. This is done by assessing each assignment statements in the loop and nested loops. If the variable on the left-hand side of the statement is an index variable in the same loop nest, the right hand side of the statement is stored as an update to the index. For each loop nest, the corresponding loop index update statements are stored in a map from variable names to a list of corresponding update statements. Each loop nest’s index update statements are then stored in a map (self.updatestatements) with the outermost for loop node of the nested loop as the key.

4. Index Vectors (chec\_in3b4a.py)

To collect the index vectors of the nested loops, the list of the indexes for each loop are collected from the outermost loop to the innermost loop. Identifying index vectors is done by identifying the type of the loop initialization, where each node can either be an Assignment or a Declaration List, and adding the indexes of the loop nest to the index vector list for the loop nest. This data is stored in a map (self.indexvectors) with the outermost for loop node of the nested loop as the key.

5/6. Test files and Output

During these two check-ins, we discovered issues in our existing code with the addition of newer and more complex test files. Corrections were made to fix our code in previous check-ins, and changes were made to improve the readability and organization of our analysis output.

**Difficulties**

Initially, we had difficulties understanding how to start completing the tasks specified in the weekly check ins and project specification. Later, we began grasping the logic to solve the tasks by examining each test file and tracing it to get the expected result of each task. This helped with understanding what needed to be done in the code to complete the tasks. Additionally, there were various tasks with greater complexity that required troubleshooting and patience. In particular, issues with the order of node visitation, branching, and nesting of nodes were difficult to address.

Issues with time management and coping with our individual school course loads also prevented us from being able to complete many of the project requirements. Internal group conflicts and the removal and addition of group members to our group particularly complicated matters. In particular, due to errors in our check in outputs which were discovered with the addition of newer and more complex test files, a large amount of time was spent debugging and correcting our previous check-ins rather than completing the final check-ins. Due to these issues, we were unable to complete the second component of check-in 4, as well as check-ins 5 and some components of check-in 6. Thus, we were not able to successfully implement code that would output the set of dependence vectors for loop nests, account for flow dependences(anti and output), or test dependence vectors for lexographic positivity.

However, our group has learned a lot from this experience. We are now aware of the importance of shared group responsibilities, and are cognizant of the importance of addressing issues with work distribution and internal conflict early on. Additionally, we are more aware of the importance of time management and testing our code on inputs with greater complexity in order to prevent situations like ours.