Leo Chau
Jachen Liu

CS591 Final Write-up

Our approach was to use methods used that we learned in class. We used Python's Abstract Syntax Tree (AST), and then we parsed through a user's code to look for function definitions that use tail-end recursion. After parsing through their code and finding the function definitions, we essentially made a copy of the user's code, edited the code via instrumentation, and then outputted the newly written executable code into a separate, new Python file which the user will be able to find and run. The code will be edited such that any functions with tail-end recursion will be changed.

Another feature we could add to this in the future is making it so that the user can change their code such that there are comments or maybe a decorator that will be used to identify functions that should be instrumented (instead of instrumenting through ALL tail-end recursive functions). This selective process would give the user more control over what they would like to change.

The program was implemented in Python. Originally, we had all the functionality (looking for tail-recursion, implementation, etc.) in one file, but then for readability reasons, we divided up the code into separate files.

Currently, the code for reading a user's code, the call to the tail recursion detection, the call to instrument the code, and the call to write the new code to a python file happens in the main function of main.py. There are then the writeFile.py, instrumentation.py and checkTailRecursion.py, which all contain functions which do as their names suggest. We made use of ast and astor libraries in order to parse through a user's code's AST and then turn it back into code.

The biggest challenge for us was figuring out the best and most concise way to transform a recursive into an iterative loop. The solution we ended up coming to was thinking about the body of the recursive function as one iteration of the loop and then the base case for the recursion to be the condition to break out of the loop. Then we changed the return statement that included the tail recursion to be that we update the variables from the parameters. Thus we ended up wrapping the body of the recursive function in a while loop and then changed the return to an assign (as ast nodes) and put it back into the body of the code. This solution works on simple code and we tested it on a tail recursive reversing an array function as was suggested during our presentation. Our code also works on that function.

A second challenge that we came across was actually instrumenting the code. There were many times where we accidentally tried to put something like a variable into the body of a function, but didn't make it into an ast.Name. Errors arose when we used the astor library. The error message from the astor library was: 'No defined handler for node of type %s'. After testing things for a bit, we figured out that the issue was indeed with how we were trying to put it back into the body of the code. Other than that, we did not come across much other problems. We were able to apply the methods that we learned from class very well and we learned a lot throughout this process and course.

The instructions on how to run our code is included in the README in the main repository under the compiling and running instructions. Please read it!