COMP-301 Project 3 Report

Berke Can Rizai - 69282
Bumin Aybars İnci - 69032

All the parts are working properly, and we have run the tests provided. Outputs for the tests are exactly the same, we also added a few comments in the code to clarify things further.

1) Here, we first defined counter as a variable at top, and it's 0 initially.
In the call-exp, with begin, we execute the following lines.
We first set the counter as counter + 1. Then, we print the format such as "# of procedure calls " and print the counter then add a new line with \n.

2) We begin with the lang.scm, here, we defined the syntaxes of proc-nested, call-nested and letrec-nested according to grammar in PDF. Inside parentheses, we first have name then the "(" so that user will use them like abc(x … than the identifiers seperated by commas then close the parentheses and lastly the name of the expression.

In the environment, inside the extend-env case of apply-env if the search symbol is same as var we check the type of the value with cases, if it is proc-val and that proc-val happens to be nested-procedure we create a new nested-procedure with count and return it with proc-val. If it is not, we return the val with else. Else is for all other cases.

In the extend-env-rec-nested, this is the same as the extend-env-rec just with a small difference that is we return nested-procedure instead of regular procedure, and it also has count since nested-procedure has.

In the interp, for proc-nested-exp, we look up the count from the environment with apply-env then take that and turn it from expressed value to number with let expressions. We return the proc-val of the nested procedure with the number we have got.

In call-nested-exp, we take operator, operand and count as arguments. We define two temporary variables with let, these are proc which is the value of the operator turned

into proc from expressed value and the arg which is a pair of value of operand and numval of the count. We, then, take the value of the proc with respect to arg.

In letrec-nested-exp, we first check the value of b-count from the environment and then turn that expression into a number value named numval. We return the value of body with respect to extended environment with the arguments and numval.

In the nested-procedure at same file, we had previously consed up the operand and the count in call-nested-exp now, we separate them first. With begin, we call the recursive displayer that prints name and number. Return the value of body with respect to extended environment with count, number with the environment that was extended with operand.

In the translator.scm, in proc-exp, we return the proc-nested-exp with the var, ‘count, name anonym that is ‘anonym and translation of the body.

For call-exp, if the expression is a var-exp, we return a call-nested-exp that has translated operator, translated operand and incremented count. We increment count with two diff expressions. We do count - (- 1) that means count + 1 since we can use diff expression. If it is not var-exp, we return count=1 inside call-nested-exp.

In letrec-exp, we return the letrec-nested expression with ‘count and translated p-body and letrec-body.

Lastly, in the data-structures.scm, we defined the nested-procedure’s structure. It consists of symbol, exp, env, symbol and number. We check them with symbol? Etc. as before. Similarly, extend-env-rec-nested has id, bvar that are symbols, body that’s expression and env and the count.

3) Here, we first defined the apply-senv-number. In apply-senv-number, we found occurrences of variables in the environment we iterate the environment recursively and whenever we find occurrence of a wanted variable we increment the result by one. If the search variable is not present, we call the function with rest of the env and if we come to the end of the environment, we return 0. So that we return the result recursively. In the var-exp, proc-exp, call-exp methods in first parameter, we created a string in respect to

variables and their occurrences in environment. We checked how many times the same symbol was present in environment and added 1 to it and cast it to string to append it to and of the variable name that was also converted to string from symbol. Then we assigned the string to a variable. In other words, changed the variable's name. On the other parameters, we used translation-of recursively with respecto to saved environment.

Workload:

Part1: Berke Can Rizai

Part 2: Berke Can Rizai and Aybars İnci

Part3:Berke Can Rizai and Aybars İnci