Exercise 04

Task 1: Deferred substitutions and closures

Consider the interpreters for FLAE.

1. In the interpreter with environments we introduced closures. What is a closure? What do we need them for ?
2. Why did we not need closures in the interpreter with substitution ?
3. Why did we not need closures in the *F1LAE* interpreter with environments ?

Task 2: Lambda Calculus

In the lecture, we introduced first-class functions. To have a minimal *Turing Complete* language, we only need 'Id', 'Fun' and 'App'. This language is called as LAMBDA CALCULUS. Refer to the interpreter *FEInterp.scala*.

The grammar of Lambda Calculus is given by

$$e ::= x \mid \lambda x.e \mid e\ e$$

1. Convert following expressions from our language into lambda expressions
   1. `App(Fun('x, 'x), 'z)`
   2. `App(Fun('x, App(Fun('y, App('x, 'y)), 'x)), Fun('z, 'p))`
   3. `App(App(Fun('x, App(Fun('y, App(Fun('x, Fun('y, 'x)), 'z)), 'z)), Fun('p, 'p)), 'q)`
2. What are *values* ? and how do we represent them in the Lambda Calculus ?
3. What are the interesting steps of the interpreter that evaluates the expressions of the lambda calculus ?

Beta reduction is used to further reduce and evaluate the lambda expressions. Rule for the beta reduction is given by

$$(\lambda x.e_1)\ e_2 \rightarrow e_1[x/e_2]$$

1. Use beta reduction to reduce the following expressions by hand.
   1. $\lambda x.(\lambda y.y\ x)\ z$
   2. $(\lambda f.\lambda g.((f\ g)(g\ f)))\ (\lambda h.(h\ h))\ (\lambda x.x)$
   3. $((((\lambda x.(\lambda y.(\lambda f.((x\ f)(y\ f))))))\ (\lambda g.(\lambda h.(h\ g))))(\lambda h.z))\ p)$

2. Use scala interpreter to reduce the following expressions.
    1. $(\lambda x.x)\ (\lambda x.x)$
    2. $(\lambda x.x)\ (\lambda x.x)\ y$
    3. $(\lambda x.\lambda y.x)\ (\lambda x.x)\ (\lambda f.\lambda g.\lambda h.(f\ g\ h))\ (\lambda x.\lambda y.y)$

Task 3: Data representation in Lambda Calculus

How can different data be represented in Lambda Calculus ? E.g. numbers, booleans and different other data structures. Data needs to be represented in the form of functions.

Represent following data / data structures using lambda expressions and using our FEInterp language.

1. Booleans
2. Natural Numbers
3. If - then - else condition