301398647 CMPT 383: Programming Assignment 3 Report Evan Mangat

Step 1:

Not much to say here, a very simple step. Half of it is given in the pdf and the other half is given roughly on Slide 19 of Lecture 18. I merely followed the logic laid out there saying "a typing environment is a map that maps identifiers to their types" and turned that to code.

Step 2:

This function takes 2 Types and either outputs Just Tint if they are both Tint's or nothing otherwise. Another relatively simple implementation, its essentially just a variation of the basic maybe function we've used many times. Very little to worry about since we are given the cases in the pdf.

Step 3:

Identical to step 2 except with an additional case that leads to the TBool rather than TInt. Also 2 Tints given to this function will return TBool instead. Almost nothing to say here.

Step 4:

The first and by far the most complex function in this assignment. Solving this required quite a bit of preparation and understanding of the FUN Language. I heavily utilized and borrowed from the logic shown on page 1 of the pdf as well as slides 18-25 of lecture 18 for the later types (Abs, App, LetIn). I decided to model the code design after a similar function we have done in that past, the eval function from P1. I found they are similar in terms of breaking down a larger expression on a case-by-case basis. I essentially just did my best to translate the typing rules into code. Most of the difficulty came from the later types, as types like Plus, Minus, Equal, etc were all relatively simple outputs.

Step 5:

As soon as I read this function's description in the pdf I knew it is nearly identical to the readFormula function in P1 except instead of type Prop it converts to type Expr. Thus the solution is essentially readFormula but swapped Prop for Expr. I also knew this shouldn't be more complex given it's a simple function. This function takes a String, uses the read function/module from deriving to assign it the Expr type.

Step 6:

Like in step 5, I drew inspiration from a similar function in P1. However unlike this functions equivalent checkFormulas, this function is of type Expr -> String. Thus I decided to directly use Step 4 and follow a case pattern using value v as described in P3.pdf. By using Step 4 to

determine whether the expression is well-typed or not, we can decide between the 2 cases and return them.

Main:

Similar setup to P1 and P2 with an added twist. In P1 and P2 you merely have to use your last step function since they were both String -> String. However in P3, the last function is Expr -> String and step 5 goes String -> Expr. Thus I used both in conjunction to effectively achieve String -> String. Other than this its relatively the same as P1/P2. We get arguments, start reading the file with the first line being the head arg and save it in input. Input is broken up by line with lines and saved into the expressions variable. Each line in expressions is read with readExpr turning it into type Expression, finally we then typeCheck each of those to get our desired result that we print.

Sample run of the program. exprs.txt contains the following test case from P3.pdf:

```
Var "x1"
Plus (CInt 1) (CInt 2)
ITE (CBool True) (CInt 1) (CInt 2)
Abs "x" TInt (Var "x")
App (Abs "x" TInt (Var "x")) (CInt 1)
LetIn "x" TInt (CInt 1) (Var "x")
```

The result:

```
PS C:\Users\evanm> ghc ./P3_Evan_Mangat.hs
[1 of 1] Compiling Main             ( P3_Evan_Mangat.hs, P3_Evan_Mangat.o )
Linking P3_Evan_Mangat.exe ...
PS C:\Users\evanm> ./P3_Evan_Mangat exprs.txt
Type Error
TInt
TInt
TArr TInt TInt
TInt
TInt
```