EECS 662

Programming Languages

Project 2 - Booleans, Identifiers, and Types

Mini Project 2 - Booleans, Identifiers, and Types EECS 662 - Programming Languages

The objective of this mini-project is to add Booleans and types to our language with `bind`. We will implement both an interpreter using substitution and an interpreter using an environment. Then we will write a type checker and integrate it with our interpreter.

To aid in your quest, the file p2.hs implements Haskell data types and function signatures needed for this project. I am not providing a parser for this project because parsers are evil.

Exercise 1 - Adding Booleans

Our first task is to implement `bind` as we've discussed in class. We will do an implementation that uses substitution and an implementation that uses an environment. Here is the new abstract syntax that includes `bind`:

```
data BBAE where
  Num :: Int -> BBAE
  Plus :: BBAE -> BBAE -> BBAE
  Minus :: BBAE -> BBAE -> BBAE
  Bind :: String -> BBAE -> BBAE -> BBAE
  Id :: String -> BBAE
  Boolean :: Bool -> BBAE
  And :: BBAE -> BBAE -> BBAE
  Leq :: BBAE -> BBAE -> BBAE
  IsZero :: BBAE -> BBAE
  If :: BBAE -> BBAE -> BBAE -> BBAE
  deriving (Show,Eq)
```

1. Define an evaluation function, `evalS :: BBAE -> (Maybe BBAE)` that uses `subst` to handle replacement of identifiers with their values.
2. Define an evaluation function, `evalM :: Env -> BBAE -> (Maybe BBAE)` that uses an environment to implement replacement of identifiers with their values.
3. Define a function, `testBBAE :: BBAE -> Bool` that takes a BBAE expression and tests our evaluators. `testBBAE` should take the expression and evaluate with both interpreters and compare their results.

Exercise 2 - Type Checking

Now let's add type checking. Here is a datatype representing number and Boolean types, respectively:

```
data TBBAE where
  TNum :: TBBAE
  TBool :: TBBAE
  deriving (Show,Eq)
```

Note that `TBBAE` is identical to our previous type definition with only a change in the datatype name. We could have used the original datatype without modification.

1. Define a type derivation function, `typeofM :: Cont -> BBAE -> (Maybe TBBAE)` that returns the type of an expression.
2. Define an evaluation function, `evalT :: BBAE -> (Maybe BBAE)` that finds the type of an expression and evaluates the expression if a type is found for it.

Note the signature of `evalT` does not accept an environment as input. You will need to provide an appropriate initial environment to `evalM` function. This is not hard!

Notes

We have talked through both the interpreters and the type inference function in class. What is required for this project is cleaning up code and fully implementing the interpreters. To give you an idea of the effort required for this mini-project, my code is about 200 lines long and took me roughly an