Laboratorul 6

SIMPLE Type Checker

Pornind de la limbajul SIMPLE prezentat in curs, să se implementeze un verificator de tipuri pentru acesta.

Sintaxa abstractă

```
type Name = String

data BinAop = Add | Mul | Sub | Div | Mod

data BinCop = Lt | Lte | Gt | Gte

data BinEop = Eq | Neq

data BinLop = And | Or

data Exp
    = Id Name
    | I Integer
    | B Bool
    | UMin Exp
    | BinA BinAop Exp Exp
    | BinC BinCop Exp Exp
    | BinE BinEop Exp Exp
    | BinL BinLop Exp Exp
    | Not Exp

data Stmt
    = Asgn Name Exp
    | If Exp Stmt Stmt
    | Read String Name
    | Print String Exp
    | While Exp Stmt
    | Block [Stmt]
    | Decl Name Exp
  deriving (Show)
```

SIMPLE type Checker

Vom folosi o "stare" in care fiecare variabila are ascociat un tip; "starile" sunt definite folosind Data.Map

```
import Data.Map.Strict (Map)
import qualified Data.Map.Strict as Map

type CheckerState = Map Name Type
```

```haskell
emptyCheckerState :: CheckerState
emptyCheckerState = Map.empty
```

Functia de verificare va asocia unei constructii sintactice o valoare M Type, unde M este o monada iar Type este un tip:

```haskell
data Type = TInt | TBool
  deriving (Eq)
```

Tipul "unit" () va fi tipul instructiunilor.

Monada M este o combinatie între monada Reader si monada Either

```haskell
newtype EReader a =
    EReader { runEReader :: CheckerState ->  (Either String a) }

instance Monad EReader where

    return a = EReader (\env ->  Right a)
    act >>= k = EReader  f
                where
                 f env  = case (runEReader act env) of
                           Left s -> Left s
                           Right va -> runEReader (k va) env

type M = EReader
```

Exercitiu

Implementati urmatoarele functii pentru a defini un verificator de tipuri pentru limbajul de mai sus folosind monada EReader.

```haskell
checkExp :: Exp -> M Type
checkStmt :: Stmt -> M ()
checkBlock :: [ Stmt ] -> M ()
checkPgm :: [ Stmt ] -> Bool
```