Eval2.hs

```
module Eval2 (eval) where

import AST

-- Estados
type State = [(Variable,Int)]

-- Estado nulo
initState :: State
initState = []

-- Busca el valor de una variabl en un estado, asumo que aparece exactamente una
vez en el estado
lookfor :: Variable -> State -> Int
lookfor x [(_,i)] = i
lookfor x ((y,i):xs) = if x==y then i else lookfor x xs

-- Cambia el valor de una variable en un estado
update :: Variable -> Int -> State -> State
update x i [] = [(x,i)]
update x i ((y,j):xs) = if x==y then ((x,i):xs) else (y,j):(update x i xs)

-- Evalua un programa en el estado nulo
eval :: Comm -> Maybe State
eval p = evalComm p (Just initState)

-- Evalua un comando en un estado dado
evalComm :: Comm -> Maybe State -> Maybe State
evalComm _ Nothing = Nothing
evalComm Skip s = s
evalComm (Let x ie) s1@(Just s) = case (evalIntExp ie s1) of
                        Nothing -> Nothing
                        Just i -> Just (update x i s)
evalComm (Seq c1 c2) s = evalComm c2 (evalComm c1 s)
evalComm (Cond be c1 c2) s = case (evalBoolExp be s) of
                                Just True -> evalComm c1 s
                                Just False -> evalComm c2 s
                                _ -> Nothing
evalComm w@(While be c) s = case (evalBoolExp be s) of
                                Just True -> evalComm (Seq c w) s
                                Just False -> s
                                _ -> Nothing


-- Evalua una expresion entera, sin efectos laterales
evalIntExp :: IntExp -> Maybe State -> Maybe Int
evalIntExp _ Nothing = Nothing
evalIntExp (Const i) s = Just i
evalIntExp (Var x) (Just s) = Just (lookfor x s)
evalIntExp (UMinus ie) s = aplicU (evalIntExp ie s) ((-1)*)
evalIntExp (Plus ie1 ie2) s = aplicBin (evalIntExp ie1 s) (evalIntExp ie2 s) (+)
evalIntExp (Minus ie1 ie2) s = aplicBin (evalIntExp ie1 s) (evalIntExp ie2 s) (-)
evalIntExp (Times ie1 ie2) s = aplicBin (evalIntExp ie1 s) (evalIntExp ie2 s) (*)
evalIntExp (Div ie1 ie2) s = aplicDiv (evalIntExp ie1 s) (evalIntExp ie2 s)
```

```haskell
-- Evalua una expresion booleana, sin efectos laterales
evalBoolExp :: BoolExp -> Maybe State -> Maybe Bool
evalBoolExp _ Nothing = Nothing
evalBoolExp BTrue s = Just True
evalBoolExp BFalse s = Just False
evalBoolExp (Eq ie1 ie2) s = aplicBin (evalIntExp ie1 s) (evalIntExp ie2 s) (==)
evalBoolExp (Lt ie1 ie2) s = aplicBin (evalIntExp ie1 s) (evalIntExp ie2 s) (<)
evalBoolExp (Gt ie1 ie2) s = aplicBin (evalIntExp ie1 s) (evalIntExp ie2 s) (>)
evalBoolExp (And be1 be2) s = aplicBin (evalBoolExp be1 s) (evalBoolExp be2 s) (&&)
evalBoolExp (Or be1 be2) s = aplicBin (evalBoolExp be1 s) (evalBoolExp be2 s) (||)
evalBoolExp (Not be) s = aplicU (evalBoolExp be s) (not)


aplicU :: Maybe a -> (a -> a) -> Maybe a
aplicU Nothing _ = Nothing
aplicU (Just a) f = Just (f a)

aplicBin :: Maybe a -> Maybe a -> (a -> a -> b) -> Maybe b
aplicBin Nothing _ _ = Nothing
aplicBin _ Nothing _ = Nothing
aplicBin (Just a) (Just b) f = Just (f a b)

aplicDiv :: Maybe Int -> Maybe Int -> Maybe Int
aplicDiv Nothing _  = Nothing
aplicDiv _ Nothing  = Nothing
aplicDiv (Just a) (Just 0)  = Nothing
aplicDiv (Just a) (Just b) = Just (div a b)
```