```
data Term = Var String    -- variable
  | Lambda String Term    -- abstraction
  | Application Term Term -- application
  deriving (Show, Eq)

-- [x -> s]y
subst :: Term -> Term -> Term -> Term

-- variable case
subst (Var x) s (Var y) = if x == y
  then s
  else (Var y)

-- application case
subst x s (Application t1 t2)  = Application (subst x s t1) (subst x s t2)

-- abstraction case
subst a@(Var x) b c@(Lambda y t) = if x == y
  then c
  else Lambda y (subst a b t)

isValue :: Term -> Bool
isValue (Lambda _ _) = True
isValue _ = False


eval1 :: Term -> Maybe Term
-- E_APPABS: (Lx.t)v -> [x->v]t
eval1 (Application (Lambda x t) v2) = if isValue v2
  then Just (subst (Var x) v2 t)
  else Nothing

eval1 (Application t1 t2) = if isValue t1
  then case (eval1 t2) of   -- E_APP2
       Just t -> Just (Application t1 t)
       otherwise -> Nothing
  else case (eval1 t1) of   -- E_APP1
       Just t -> Just (Application t t2)
       otherwise -> Nothing

eval1 _ = Nothing


eval :: Term -> Term
eval t = case (eval1 t) of
  Just t' -> eval t'
  Nothing -> t
```