Simply Typed Lambda Calculus

From Untyped to Simply Typed Lambda Calculus

Sven Tennie

August 24, 2018

Dream IT
https://dreamit.de

Untyped Lambda Calculus

Naive Interpreter

```
module NaiveUntypedEval where
import qualified Data.Map.Strict as Map

type Name = String

data Term = Variable Name |
    Application Term Term |
    Abstraction Name Term
    deriving (Eq, Show)

eval :: Term -> Term
eval (Variable name) = Variable name
eval (Application term1 term2) = case eval term1 of
  (Abstraction name term1') -> eval $ substitute name term2
  term                      -> Application term term2
```

Interpreter with Environment

```
module UntypedEval where
import qualified Data.Map.Strict as Map

type Name = String
type Environment = Map.Map Name Term

data Term = Variable Name |
    Application Term Term |
    Abstraction Name Term
    deriving (Eq, Show)

eval :: Environment -> Term -> Maybe Term
eval env (Variable name) = find env name
eval env (Application term1 term2) = case eval env term1 of
  Just (Abstraction name term) -> eval (Map.insert name ter
```

Tests

Simply Typed Lambda Calculus

Interpreter

Type Checker

```
module TypedCheck where

import qualified Data.Map.Strict as Map
import Data.Either.Extra

type Name = String
type Environment = Map.Map Name Type

data Type  = TInt
   | TBool
   | TArr Type Type
   deriving (Eq, Show)

data Term = Variable Name |
      Application Term Term |
```

Tests