A Very Simple Big-Step λ-Interpreter

Hayo Thielecke

February 17th, 2012

Code - Type Definitions

```
type debruijn =
  Lambda of debruijn |
  Index of int |
  App of debruijn * debruijn |
  Const of int;;

type value =
  Closure of debruijn * value list |
  Con of int;;
```

Code - The Lookup Function

```
let rec index n xs =
  match xs with
    x::ys ->
      if n = 1 then x
      else index (n - 1) ys;;
```

Code - The Eval Function (Interpreter)

```
let rec eval exp env =
match exp with
  Const n ->
    Con n |
  Index n ->
    index n env |
  Lambda body ->
    Closure (body, env) |
  App (funct, argument) ->
    match eval funct env with
      Closure (body, env2) ->
        let argval = eval argument env in
          eval body (env2 @ [argval]);;
```

Code - Tests

```
let t1 = Lambda (Lambda (Index 2));;
let t2 = App (t1, t1);;
let t3 = App(App(t1, Const 666), Const 777);;
let t4 = App(t2, t3);;
eval t4 [];;

let t5 = Lambda (Lambda (App (Index 1, Index 2)));;
let t6 = Lambda (Index 1);;
let t7 = App (t5, t6);;
let t8 = App (t7, Const 5);;
eval t8 [];;

let o1 = Lambda (App(Index 1, Index 1));;
let omega = App(o1, o1);;
eval omega [];;
```