unicaml

Normal order evaluation of λ-terms

Syntax of λ-terms

```
# type term = Var of string
            | Abs of (string * term)
            | App of term * term;;
```

Examples

```
# let id = Abs("x",Var "x");;
# let k = Abs("x",Abs("y",Var "x"));;
# let omega = App(Abs("x",App(Var "x",Var "x")),Abs("x",App(Var "x",Var "x")));;
# let const z = Abs("x",Var z);;
```

Pretty-printing functions

```
# let rec string_of_term t = match t with
    Var x -> x
  | Abs (x,t') -> "\\" ^ x ^ "." ^ string_of_term t'
  | App (t0,t1) -> "(" ^ (string_of_term t0) ^ " " ^ (string_of_term t1) ^ ")";;

# print_string (string_of_term id);;
# print_string (string_of_term k);;
# print_string (string_of_term omega);;
```

Free variables and substitutions

Sets

```
# type 'a set = Set of 'a list;;

# let emptyset = Set [];;

# let rec member x s = match s with
    Set [] -> false
  | Set (y::s') -> x=y or (member x (Set s'));;

# let rec union s t = match s with
    Set [] -> t
  | Set (x::s') -> (match union (Set s') t with Set t' ->
      if member x t then Set t' else Set (x::t'));;

# let rec diff s x = match s with
    Set [] -> s
  | Set (y::s') -> (match diff (Set s') x with Set t' ->
      if x=y then Set s' else Set (y::t'));;
```

Free variables

```
# let rec fv t = match t with
    Var x -> Set [x]
```

```
  | App(t0,t1) -> union (fv t0) (fv t1)
  | Abs(x,t0) -> diff (fv t0) x;;
```

```
# fv omega;;
# fv (const "z");;
# fv (Abs ("x", App(Var "x",Var "y")));;
```

Substitutions

```
# let count = ref(-1);;
# let gensym = fun () -> count := !count +1; "x" ^ string_of_int (!count);;
```

```
# let rec subst x t' t = match t with
    Var y -> if x=y then t' else Var y
  | App(t0,t1) -> App(subst x t' t0, subst x t' t1)
  | Abs(y,t0) when y=x -> Abs(x,t0)
  | Abs(y,t0) when y!=x && not (member y (fv t')) -> Abs(y, subst x t' t0)
  | Abs(y,t0) when y!=x && member y (fv t') ->
      let z = gensym() in Abs(z,subst x t' (subst z (Var y) t0));;
```

Leftmost-outermost reduction

```
# let isredex t = match t with
    App(Abs(x,t0),t1) -> true
  | _ -> false;;
```

```
# let rec hasredex t = match t with
    Var x -> false
  | Abs(x,t') -> hasredex t'
  | App(t0,t1) -> isredex t or hasredex t0 or hasredex t1;;
```

```
# exception Error;;
```

```
# let rec reduce1 t = if not (hasredex t) then t else match t with
    Abs(x,t') -> Abs(x,reduce1 t')
  | App(Abs(x,t0),t1) -> subst x t1 t0
  | App(t0,t1) -> if hasredex t0 then App(reduce1 t0,t1) else App(t0,reduce1 t1);;
```

```
# let rec reduce t k = if k=0 then t else let t' = reduce1 t in reduce t' (k-1);;
```

```
# let rec reducefix t = let t' = reduce1 t in if t'=t then t' else reducefix t';;
```

```
# reduce (App((const "z"),omega)) 1;;
```

untypedlambda.txt · Last modified: 2015/10/08 15:20 (external edit)