unicaml

Operational interpreter of a functional language

Syntax

```
type ide = Ide of string;;

type exp =
    N of int
  | Val of ide
  | Add of exp * exp
  | Sub of exp * exp
  | Mul of exp * exp
  | Div of exp * exp
  | True
  | False
  | Eq of exp * exp
  | Leq of exp * exp
  | Not of exp
  | And of exp * exp
  | Or of exp * exp
  | If of exp * exp * exp
  | Let of ide * exp * exp
  | Letrec of ide * exp * exp
  | Fun of ide * exp
  | Apply of exp * exp
;;
```

Semantic domains and environment

```
type eval =
    Bool of bool
  | Int of int
  | EFun of ide * exp * env
  | Unbound

and env = Env of (ide -> eval);;

exception UnboundIde of ide;;

let emptyenv = Env (fun x -> Unbound)
and bind (Env r) x d = Env (fun y -> if y=x then d else r y)
and applyenv (Env r) x = match r x with
  Unbound -> raise (UnboundIde x)
| _ as d -> d
;;
```

Operational semantics

```
let rec intval e r = match (sem e r) with
  Int n -> n
| _ -> raise TypeMismatch

and boolval e r = match sem e r with
  Bool b -> b
| _ -> raise TypeMismatch
```

```
and sem e r = match e with
  N n -> Int n
| Val x ->  applyenv r x
| Add (e1,e2) ->  Int (intval e1 r + intval e2 r)
| Sub (e1,e2) ->  Int (intval e1 r - intval e2 r)
| Mul (e1,e2) ->  Int (intval e1 r * intval e2 r)
| Div (e1,e2) ->  Int (intval e1 r / intval e2 r)
| True -> Bool true
| False -> Bool false
| Eq (e1,e2) -> Bool (intval e1 r = intval e2 r)
| Leq (e1,e2) -> Bool (intval e1 r <= intval e2 r)
| Not e' -> Bool (not (boolval e' r))
| And (e1,e2) -> Bool (boolval e1 r && boolval e2 r)
| Or (e1,e2) -> Bool (boolval e1 r || boolval e2 r)
| If(e0,e1,e2) -> if boolval e0 r then sem e1 r else sem e2 r
| Let (x,e1,e2) -> sem e2 (bind r x (sem e1 r))
| Letrec (x,e1,e2) -> let rec r' = Env(fun y -> applyenv (bind r' x (sem e1 r')) y) in sem e2 r'
| Fun (x,e') -> EFun (x,e',r)
| Apply (e1,e2) -> match sem e1 r with
    EFun (x,e',r') -> sem e' (bind r' x (sem e2 r))
  | _ -> raise TypeMismatch
;;
```

```
val sem : exp -> env -> eval = <fun>
```

Examples

```
let e0 = Let(Ide "succ",Fun(Ide "x", Add(Val(Ide "x"), N 1)),Apply(Val(Ide "succ"),N 8));;
sem e0 emptyenv;;

# - : eval = Int 9

let e1 = Letrec(Ide "fact",
                Fun(Ide "x",
                    If(Eq(Val(Ide "x"),N 0),
                       N 1,
                       Mul(Val(Ide "x"),
                           Apply(Val(Ide "fact"),Sub(Val(Ide "x"), N 1))))),
               Apply(Val(Ide "fact"),N 5));;

sem e1 emptyenv;;

# - : eval = Int 120

let bot = Letrec(Ide "f", Fun(Ide "x", Apply(Val(Ide "f"),Val(Ide "x"))),
                Apply(Val(Ide "f"), N 1));;

sem bot emptyenv;;

# (* non-terminating computation *)
```

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