ソフトウェアサイエンス実験 S8 課題 3-2

200911434 青木大祐

平成 24 年 11 月 8 日

eval3 のソースコードを以下に示す。また、作成したソースコード全体は末尾に添付する。

```
let rec eval3 ?(mode=0) e env =           (* env を引数に追加 *)
  if mode != 0 then print_env env;        (* デバッグモード *)

  let binop f e1 e2 env =       (* binop の中でも eval3 を呼ぶので env を追加 *)
    match (eval3 e1 env ~mode, eval3 e2 env ~mode) with
    | (IntVal(n1),IntVal(n2)) -> IntVal(f n1 n2)
    | _ -> failwith "integer value expected"
  in
  match e with
  | Var(x)       -> lookup x env
  | IntLit(n)    -> IntVal(n)
  | BoolLit(b) -> BoolVal(b)
  | Plus(e1,e2)  -> binop (+) e1 e2 env     (* env を追加 *)
  | Times(e1,e2) -> binop ( * ) e1 e2 env   (* env を追加 *)
  | Eq(e1,e2) ->
      begin
  match (eval3 e1 env ~mode, eval3 e2 env ~mode) with
    | (IntVal(n1),IntVal(n2)) -> BoolVal(n1=n2)
    | (BoolVal(b1),BoolVal(b2)) -> BoolVal(b1=b2)
    | _ -> failwith "wrong value"
      end
  | If(e1,e2,e3) ->
      begin
        match (eval3 e1 env ~mode) with          (* env を追加 *)
          | BoolVal(true)  -> eval3 e2 env   (* env を追加 *)
          | BoolVal(false) -> eval3 e3 env   (* env を追加 *)
          | _ -> failwith "wrong value"
      end
  | Let(x,e1,e2) ->
      let env1 = ext env x (eval3 e1 env ~mode)
      in eval3 e2 env1 ~mode
  | _ -> failwith "unknown expression";;
```

3.1.1 動作の確認

次のような例を与えて動作を確認した。

```
eval3 (Let ("x", IntLit 1, (Plus (IntLit 2, Var "x")))) [];;
eval3 (Let ("x", IntLit 1, Let("y", IntLit 3, (Plus (Var "y", Var "x"))))) [];;
eval3 (Let ("x", BoolLit true, If(Eq(Var "x", BoolLit true), IntLit 1, IntLit 2))) [];;
```

実行結果は以下の通り。

```
# - : value = IntVal 3
# - : value = IntVal 4
# - : value = IntVal 1
```

正しく実行できていることが分かる。

3.1.2 let のネスト

次のような式で、let 文をネストした際の動作を確認した。

```
eval3 (Let ("x", IntLit 1, (Let ("x", IntLit 2, Var "x")))) [];;
```

実行結果は以下のとおり。

```
#   - : value = IntVal 2
```

また、Ocaml で実行した結果は以下のとおり。同じように計算できていることが分かる。

```
# let x = 1 in let x = 2 in x;;
Warning 26: unused variable x.
- : int = 2
```

3.1.3 let のスコープ

次のような例を与えて動作を確認した。

```
eval3 ~mode:1  (Let ("x", IntLit 1, (Let ("y", Plus(Var "x", IntLit 1), Plus(Var "x", Var "y"))))) [];;
```

実行結果は以下のとおり。

```
[][][x = 1;][x = 1;][x = 1;][x = 1;][y = 2;x = 1;][y = 2;x = 1;][y = 2;x = 1;]- : value = IntVal 3
```

eval の中で次の eval が呼ばれる時に現在の env を渡しているので、内側の式でも外側で定義した変数が参照できていることが分かる。

3.1.4 束縛の解除

Let 文では、受け取った環境について新たな束縛を追加した環境を新しく作り、内側の式には新しい環境を適用している。そのため、Let 文の外側の環境には影響を及ぼさない仕組みになっている。

3.1.5 ソースコード全体

```
(* eval2b : exp -> value *)
let debug = false;

type exp =
  | IntLit of int
  | Plus of exp * exp
  | Times of exp * exp
  | BoolLit of bool        (* 追加分; 真理値リテラル, つまり trueや false  *)
  | If of exp * exp * exp  (* 追加分; if-then-else式 *)
  | Eq of exp * exp        (* 追加分; e1 = e2 *)
  | Greater of exp * exp
  | Var of string
  | Let of string * exp * exp ;;

(* 値の型 *)
type value =
  | IntVal  of int          (* 整数の値 *)
  | BoolVal of bool         (* 真理値の値 *);;

let emptyenv () = [];;

let ext env x v = (x,v) :: env;;

let rec lookup x env =
   match env with
   | [] -> failwith ("unbound variable: " ^ x)
   | (y,v)::tl -> if x=y then v
                  else lookup x tl;;

let string_of_env env =
  let string_of_val (e:value) : string =
    match e with
      | IntVal n -> string_of_int n
      | BoolVal true ->  "true"
      | BoolVal false ->  "false"
  in
  let rec internal (env:(string * value) list) =
    match env with
      | [] -> ""
      | (name, v)::rest -> name ^ " = " ^ (string_of_val v) ^ ";" ^ (internal rest)
  in
  "[" ^ (internal env) ^ "]";;

(* eval3 : exp -> (string * value) list -> value *)
(* let と変数、環境の導入 *)
let print_env (env: (string * value) list) =
  print_string( string_of_env env);;

let rec eval3 ?(mode=0) e env =           (* env を引数に追加 *)
  if mode != 0 then print_env env;

  let binop f e1 e2 env =       (* binop の中でも eval3 を呼ぶので env を追加 *)
    match (eval3 e1 env ~mode, eval3 e2 env ~mode) with
    | (IntVal(n1),IntVal(n2)) -> IntVal(f n1 n2)
    | _ -> failwith "integer value expected"
  in
  match e with
  | Var(x)       -> lookup x env
  | IntLit(n)    -> IntVal(n)
  | BoolLit(b) -> BoolVal(b)
  | Plus(e1,e2)  -> binop (+) e1 e2 env     (* env を追加 *)
  | Times(e1,e2) -> binop ( * ) e1 e2 env   (* env を追加 *)
  | Eq(e1,e2) ->
      begin
  match (eval3 e1 env ~mode, eval3 e2 env ~mode) with
    | (IntVal(n1),IntVal(n2)) -> BoolVal(n1=n2)
    | (BoolVal(b1),BoolVal(b2)) -> BoolVal(b1=b2)
    | _ -> failwith "wrong value"
      end
  | If(e1,e2,e3) ->
      begin
        match (eval3 e1 env ~mode) with          (* env を追加 *)
          | BoolVal(true)  -> eval3 e2 env   (* env を追加 *)
          | BoolVal(false) -> eval3 e3 env   (* env を追加 *)
          | _ -> failwith "wrong value"
      end
  | Let(x,e1,e2) ->
      let env1 = ext env x (eval3 e1 env ~mode)
      in eval3 e2 env1 ~mode
  | _ -> failwith "unknown expression";;


(* 3.2.1 *)
eval3 (Let ("x", IntLit 1, (Plus (IntLit 2, Var "x")))) [];;
eval3 (Let ("x", IntLit 1, Let("y", IntLit 3, (Plus (Var "y", Var "x"))))) [];;
eval3 (Let ("x", BoolLit true, If(Eq(Var "x", BoolLit true), IntLit 1, IntLit 2))) [];;
(* 3.2.2 *)
```

```
eval3 (Let ("x", IntLit 1, (Let ("x", IntLit 2, Var "x")))) [];;
(* 3.2.3 *)
eval3 ~mode:1  (Let ("x", IntLit 1, (Let ("y", Plus(Var "x", IntLit 1), Plus(Var "x", Var "y"))))) [];;
(* 3.2.4 *)
eval3 (Let ("x", IntLit 1, Plus(Let("x", IntLit 2, Plus(Var "x", IntLit 1)), Times(Var "x", IntLit 2)))) [];;

print_env [("x", IntVal 1); ("y", IntVal 2); ("z", BoolVal true)]
```