ソフトウェアサイエンス実験 S8 課題 4-1

200911434 青木大祐

平成 24 年 11 月 8 日

4.1.1 評価結果を返す関数

文字列として与えられた式を評価した結果を返す式として、以下の run 関数を作成した。

```
let run src =
  Eval.eval3 (Main.parse(src)) (Eval.emptyenv())
;;
```

例として ”1 + 2 ∗ 3” を与えた実行結果を以下に示す。

```
# val run : string -> Syntax.value = <fun>
- : Syntax.value = Syntax.IntVal 7
```

正しく計算できていることが分かる。

4.1.2 演算子の追加

parser.mly と *lexer.mll* に以下のような記述を追加した。

```
...

// 演算子
%token PLUS     // '+'
%token MINUS    // '-'
%token ASTERISK // '*'
%token SLASH    // '/'
%token EQUAL    // '='
%token LESS     // '<'
%token GREATER  // '>'
%token COLCOL   // "::"
%token NOTEQUAL // "<>"

...

%left EQUAL GREATER LESS NOTEQUAL

...

  // e1 <> e2
  | exp NOTEQUAL exp
    { Noteq ($1, $3) }

...
```

```
 ...

  (* 演算子 *)
  | '+'       { PLUS }
  | '-'       { MINUS }
  | '*'       { ASTERISK }
  | '/'       { SLASH }
  | '='       { EQUAL }
  | '<'       { LESS }
  | '>'       { GREATER }
  | "::"      { COLCOL }
  | "<>"      { NOTEQUAL }

...
```

また、*syntax.ml* にも新たに *Noteq* を追加した。

```
...

(* 式の型 *)
type exp =
  | Var of string         (* variable e.g. x *)
  | IntLit of int         (* integer literal e.g. 17 *)
  | BoolLit of bool       (* boolean literal e.g. true *)
  | If of exp * exp * exp (* if e then e else e *)
  | Let of string * exp * exp   (* let x=e in e *)
  | LetRec of string * string * exp * exp   (* letrec f x=e in e *)
  | Fun of string * exp   (* fun x -> e *)
  | App of exp * exp      (* function application i.e. e e *)
  | Eq of exp * exp       (* e = e *)
  | Noteq of exp * exp (* e <> e *)
  | Greater of exp * exp  (* e > e *)
  | Less of exp * exp     (* e < e *)
  | Plus of exp * exp     (* e + e *)
  | Minus of exp * exp    (* e - e *)
  | Times of exp * exp    (* e * e *)
```

```
  | Div of exp * exp      (* e / e *)
  | Empty               (* [ ] *)
  | Match of exp * ((exp * exp) list)    (* match e with e->e | ... *)
  | Cons of exp * exp     (* e :: e *)
  | Head of exp           (* List.hd e *)
  | Tail of exp           (* List.tl e *)

...
```

これらを踏まえて、eval3 にも Noteq に対する処理を追加した。

```
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
  | Noteq(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
  | _ -> failwith "unknown expression";;
```

以下に与えた例と実行結果を示す。

```
run "true <> false";;
run "(1+1) <> 2";;
run "let x = 1 in let y = 2 in x <> y";;

- : Syntax.value = Syntax.BoolVal true
- : Syntax.value = Syntax.BoolVal false
- : Syntax.value = Syntax.BoolVal true
```

正しく計算できていることが分かる。