CSE216 Programming Abstractions

Tiny PL

YoungMin Kwon

Tiny PL

- expr type in Variants

```
(*TODO: write the expr type*)
type expr
    = NUM  of          (*integer*)
    | BOOL of          (*Boolean*)
    | VAR  of          (*variable name*)
    (*arithmetic exprs*)
    | ADD of           (*binary operator*)
    | SUB of
    (*comparators*)
    | EQ of
    | GE of
    (*logical exprs*)
    | AND of
    | OR  of
    | NOT of           (*unary operator*)
    (*conditional expr*)
    | IF  of           (*condition-expr, true-expr, false-expr*)
    (*function definition: parameter, body*)
    | FUN of           (*param name and body-expr*)
    (*closure: parameter, body, environment: list of (name, expr) tuples*)
    | CLO of           (*param name, body-expr, env*)
    (*function application: operator, operand*)
    | APP of           (*function and argument*)
```

Tiny PL

- expr type in Variants

```
type expr
    = NUM  of int    (*number*)
    | BOOL of bool   (*Boolean*)
    | VAR  of string (*variable*)
    (*arithmetic exprs*)
    | ADD of expr * expr | SUB of expr * expr
    (*comparators*)
    | EQ of  expr * expr | GE of  expr * expr
    (*logical exprs*)
    | AND of expr * expr | OR  of expr * expr | NOT of expr
    (*conditional expr*)
    | IF  of expr * expr * expr
    (*function definition: parameter, body*)
    | FUN of string * expr
    (*closure: parameter, body, environment*)
    | CLO of string * expr * (string * expr) list
    (*function application: operator, operand*)
    | APP of expr * expr
```

Tiny PL

- Literal expressions

```
match expr with
(*TODO: evaluate the literal expressions*)
| BOOL b ->
| NUM n  ->
```

Tiny PL

- Literal expressions

```
match expr with
(*TODO: evaluate the literal expressions*)
| BOOL b -> BOOL b
| NUM n  -> NUM n
```

Tiny PL

- Environment

```
(*evaluate expr in env*)
let rec eval expr env =
…
    (*look up the value of a variable from an environment*)
    (*TODO: implement lookup
        environment is a list of (name, expr) tuples
    *)
    let rec lookup name env =
```

Tiny PL

- Environment

```
(*evaluate expr in env*)
let rec eval expr env =
…
    (*look up the value of a variable from an environment*)
    (*TODO: implement lookup
        environment is a list of (name, expr) tuples
    *)
    let rec lookup name env =
        match env with
        | [] -> print (VAR name); assert false
        | (n, e)::rest -> if name = n
                          then e
                          else lookup name rest in
```

Tiny PL

- Variable expression

```
(*TODO: evaluate the variable expression*)
| VAR v  ->
```

Tiny PL

- Variable expression

```
(*TODO: evaluate the variable expression*)
| VAR v  -> lookup v env
```

Tiny PL

- Primitive operators

```
(*TODO: implement the primitive operators*)
| ADD (a, b)   ->
| SUB (a, b)   ->
| EQ  (a, b)   ->
| GE  (a, b)   ->
| AND (a, b)   ->
| OR  (a, b)   ->
| NOT a        ->
```

Tiny PL

- Primitive operators

```
(*TODO: implement the primitive operators*)
| ADD (a, b)    -> eval a env |> fun x ->
                   eval b env |> fun y ->
                   NUM (dropNUM x + dropNUM y)
| SUB (a, b)    -> eval a env |> fun x ->
                   eval b env |> fun y ->
                   NUM (dropNUM x - dropNUM y)
| EQ (a, b)     -> eval a env |> fun x ->
                   eval b env |> fun y ->
                   BOOL (dropNUM x = dropNUM y)
| GE (a, b)     -> eval a env |> fun x ->
                   eval b env |> fun y ->
                   BOOL (dropNUM x >= dropNUM y)
| AND (a, b)    -> eval a env |> fun x ->
                   eval b env |> fun y ->
                   BOOL (dropBOOL x && dropBOOL y)
| OR (a, b)     -> eval a env |> fun x ->
                   eval b env |> fun y ->
                   BOOL (dropBOOL x || dropBOOL y)
| NOT a         -> eval a env |> fun x ->
                   BOOL (not (dropBOOL x))
```

Tiny PL

- Conditional expression

```
(*TODO: implement the conditional expression*)
| IF (c, t, f) ->
```

Tiny PL

- Conditional expression

```
(*TODO: implement the conditional expression*)
| IF (c, t, f) -> eval c env |> fun x ->
                  if dropBOOL x then eval t env
                                else eval f env
```

Tiny PL

- Function definition

```
(*TODO: evaluate the function definition*)
| FUN (v, e)   ->
```

Tiny PL

- Function definition

```
(*TODO: evaluate the function definition*)
| FUN (v, e)    -> CLO (v, e, env)
```

Tiny PL

- Function application

```
(*TODO: implement the function application*)
| APP (f, a)   ->
```

Tiny PL

- Function application

```
(*TODO: implement the function application*)
| APP (f, a)   -> eval f env  |> fun clo ->
                  eval a env  |> fun x ->
                  dropCLO clo |> fun (v, e, ev) ->
                  eval e ((v, x)::ev)
```