CS131: Programming Languages

Lun Liu

05.05.17

HW3: An Interpreter for Mini-OCaml (MOCaml)

- Due: Monday, May 8, 2017, 11:30 PM
- Today
  - HW3 hints

MOCaml: AST

- Abstract Syntax Tree
  - Produced by parser
  - Represent the program
  - Will be given to (`evalDecl: modecl -> moenv -> moresult)`
- See `ast.ml`
- Play with `(toAST : string -> modecl)` in Ocaml toplevel

```
# toAST "match x with 0 -> 1 | a -> a";;
- : modecl =
Expr (Match (Var "x",
             [(IntPat 0, IntConst 1);
              (VarPat "a", Var "a")]))
```

MOCaml: Important Data Types

- mopat: patterns
- moexpr: expressions
- modecl: declarations
- movalue: values
- moenv: environment
- moresult: evaluation result

MOCaml: mopat

- match x with **p1** -> e1 | **p2** -> e2
- function **p** -> e

mopat	example OCaml code	example mopat instance
`IntPat of int`	`3`	`IntPat(3)`
`BoolPat of bool`	`true`	`BoolPat(true)`
`WildcardPat`	`_`	`WildcardPat`
`VarPat of string`	`myvar`	`VarPat("myvar")`
`TuplePat of mopat list`	`(a, b)`	`TuplePat ([VarPat("a"); VarPat("b")])`
`DataPat of string * mopat option`	`Cons(x, y)`	`DataPat ("Cons", Some (TuplePat [VarPat "x"; VarPat "y"]))`

MOCaml: moexpr

- moexpr: expressions
- moop = Plus | Minus | Times | Eq | Gt (* + | - | * | = | >*)

moexpr	example OCaml code	example moexpr instance
`IntConst of int`	`3`	`IntConst (3)`
`BoolConst of bool`	`true`	`BoolConst (true)`
`Var of string`	`myvar`	`Var ("myvar")`
`BinOp of moexpr * moop * moexpr`	`1 + 2`	`BinOp (IntConst(1), Plus, IntConst(2))`
`Negate of moexpr`	`-3`	`Negate(IntConst(3))`
`If of moexpr * moexpr * moexpr`	`if true then 1 else 2`	`If (BoolConst(true), IntConst(1), IntConst(2))`

MOCaml: moexpr

- moexpr: expressions
- moop = Plus | Minus | Times | Eq | Gt (* + | - | * | = | >*)

moexpr	example OCaml code	example moexpr instance
`Function of mopat * moexpr`	`function x -> -x`	`Function(VarPat("x"),Negate(Var("x")) )`
`FunctionCall of moexpr * moexpr`	`times2 x`	`FunctionCall (Var "times2", Var "x")`
`Match of moexpr * (mopat * moexpr) list`	`match x with 1 -> 2`	`Match (Var "x", [(IntPat 1, IntConst 2)])`
`Tuple of moexpr list`	`(1,a)`	`Tuple [IntConst 1; Var "a"]`
`Data of string * moexpr option`	`Cons(1, Nil)`	`Data ("Cons", Some (Tuple [IntConst 1; Data ("Nil", None)]))`

MOCaml: modecl

- modecl: declarations

modecl	example OCaml code	example modecl instance
`Expr of moexpr`	`1 + 2`	`Expr (BinOp (IntConst 1, Plus, IntConst 2))`
`Let of string * moexpr`	`let x = 1 + 2`	`Let ("x", BinOp (IntConst 1, Plus, IntConst 2))`
`LetRec of string * moexpr`	`let rec myRecFun = function x -> myRecFun x`	`LetRec ("myRecFun", Function (VarPat "x", FunctionCall (Var "myRecFun", Var "x")))`

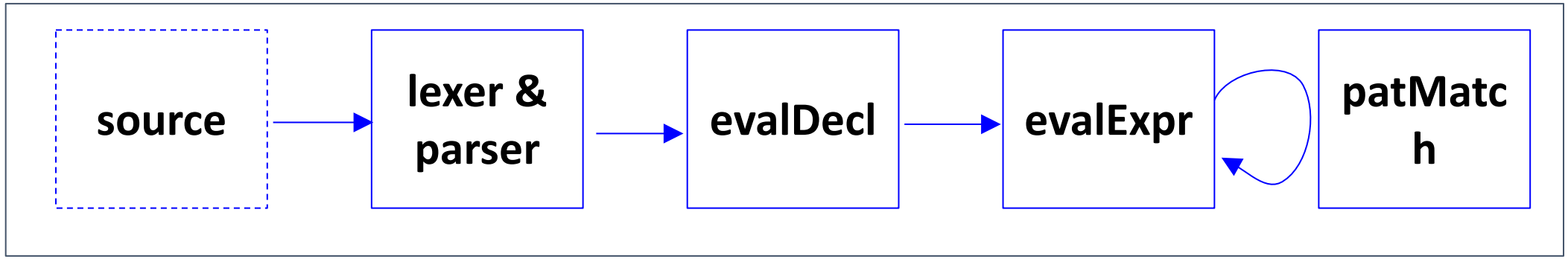
MOCaml: movalue

- movalue: values (after evaluation)

movalue	example OCaml code	example movalue instance
`IntVal of int`	`1`	`IntVal(1)`
`BoolVal of bool`	`true`	`BoolVal(true)`
`FunctionVal of string option * mopat * moexpr * moenv`	`function p -> e`	`FunctionVal(None, VarPat("p"), Var("e"), env)`
`TupleVal of movalue list`	`(2, 1+2)`	`TupleVal(IntVal(1), IntVal(3))`
`DataVal of string * movalue option`	`Nil`	`DataVal("Nil", None)`

MOCaml Main Entrance

```
let testOne test env =
      let decl = main token (Lexing.from_string (test^";;")) in
      let res = evalDecl decl env in
      let str = print_result res in
            match res with
            (None, v) -> (str, env)
            | (Some x,v) -> (str, Env.add_binding x v env)
```



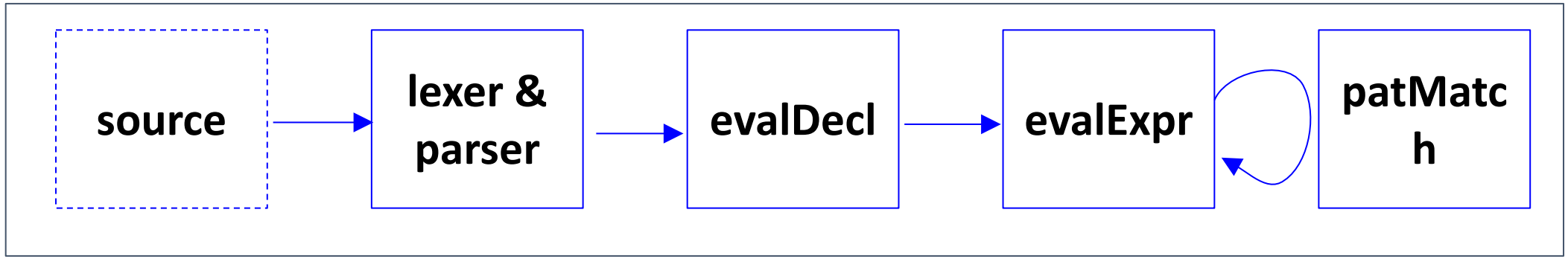
REPL (read-eval-print) loop

This slide is modified from https://docs.google.com/presentation/d/16JHZKEuEhF_Pzjr9N5LflyUXCofYl0OyLH_Mp6csa_g/edit#slide=id.g216ae3ddab_0_118

MOCaml Main Entrance

AST : modecl

```
let testOne test env =
      let decl = main token (Lexing.from_string (test^";;")) in
      let res = evalDecl decl env in
      let str = print_result res in
            match res with
            (None, v) -> (str, env)
            | (Some x,v) -> (str, Env.add_binding x v env)
```



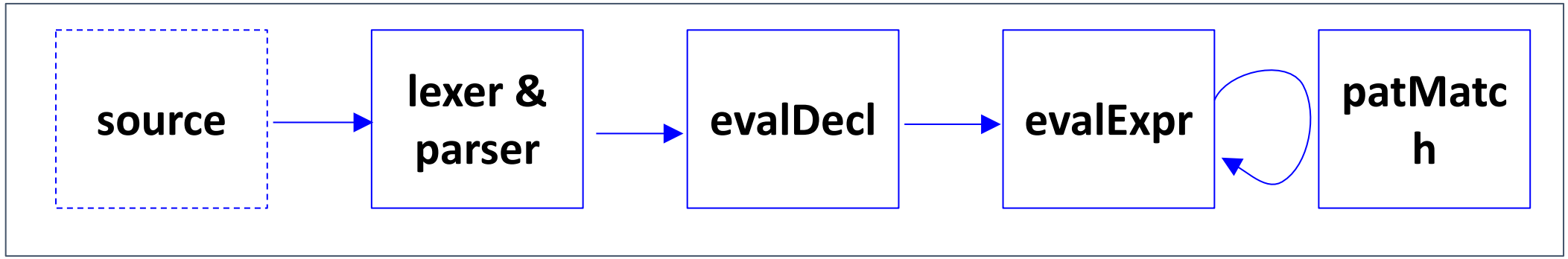
REPL (read-eval-print) loop

This slide is modified from https://docs.google.com/presentation/d/16JHZKEuEhF_Pzjr9N5LflyUXCofYl0OyLH_Mp6csa_g/edit#slide=id.g216ae3ddab_0_118

MOCaml Main Entrance

moresult (string option * movalue)

```
let testOne test env =
      let decl = main token (Lexing.from_string (test^";;")) in
      let res = evalDecl decl env in
      let str = print_result res in
            match res with
            (None, v) -> (str, env)
            | (Some x,v) -> (str, Env.add_binding x v env)
```



REPL (read-eval-print) loop

This slide is modified from https://docs.google.com/presentation/d/16JHZKEuEhF_Pzjr9N5LflyUXCofYl0OyLH_Mp6csa_g/edit#slide=id.g216ae3ddab_0_118

MOCaml: Environment Module

- `env.ml`
- Methods
  - `Env.empty_env: unit -> 'a env`
  - `Env.add_binding: string -> 'a -> 'a env -> 'a env`
  - `Env.combine_envs: 'a env -> 'a env -> 'a env`
  - `Env.lookup: string -> 'a env -> 'a`

- moenv = movalue Env.env (`'a` instantiated with `movalue`)

MOCaml: Pattern Matching

- Match of moexpr * (mopat * moexpr) list

```
match x with
      p1 -> e1
    | p2 -> e2
      …
    | pn -> en
```

MOCaml: Pattern Matching

- Match of moexpr * (mopat * moexpr) list

```
match x with
      p1 -> e1
    | p2 -> e2
      …
    | pn -> en
```

MOCaml: Pattern Matching

- Match of moexpr * (mopat * moexpr) list

```
match x with
      p1 -> e1    (p1,e1)
    | p2 -> e2
      …
    | pn -> en
```

MOCaml: Pattern Matching

- Match of moexpr * (mopat * moexpr) list

```
match x with
      p1 -> e1
    | p2 -> e2    (p2,e2)
      …
    | pn -> en
```

MOCaml: Pattern Matching

- Match of moexpr * (mopat * moexpr) list

```
match x with
      p1 -> e1
    | p2 -> e2
      …
    | pn -> en    (pn,en)
```

MOCaml: Pattern Matching

- Match of moexpr * (mopat * moexpr) list

```
match x with
      p1 -> e1
    | p2 -> e2
      …
    | pn -> en
```

```
[(p1, e1); (p2, e2); … (pn, en)]
```

MOCaml: Pattern Matching

- `Match of moexpr * (mopat * moexpr) list`
- Evaluate a `Match` Expr

```
let rec evalExpr (e:moexpr) (env:moenv) : movalue =  match e
with
   …
| Match (expr, patterns) -> ???
   …
```

- `patMatch expr` with every `mopat` in `patterns`
- `patMatch` should return a new `moenv` which has everything in the parent `moenv` and the new bindings from pattern mathching (shadowing??)
- Evaluate corresponding `moexpr`

MOCaml: Function & Function Call

- `Function of mopat * moexpr`
  - For `Function (p, e), p:` formal parameter; `e:` function body
- Evaluate a `Function`
  - Remember the `env`
- `FunctionCall of moexpr * moexpr`
  - The first `moexpr` should be evaluated to `a FunctionVal`
- Evaluate a `FunctionCall`
  - `patMatch` formal parameters and real arguments
  - Get new `env`
  - Evaluate function body under new `env`
- `let rec??`