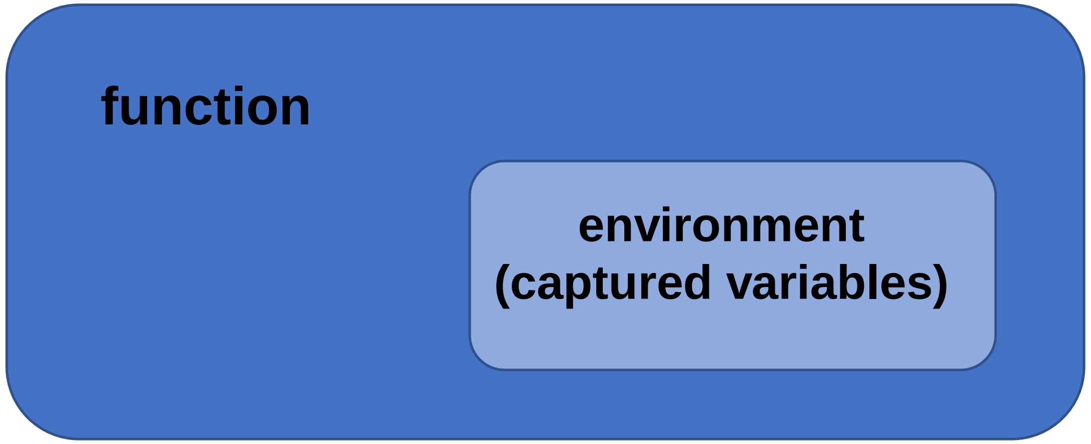
Week 5

Brendon Faleiro

Environments and Closures

- An environment is a mapping from variable names to values
  - Simply a table of (name, value) pairs

- A **closure** is a pair (f, e) consisting of function f and an environment e.
  - When the closure is invoked, f is evaluated using **e** to look up variable bindings



Currying and Dynamic Scoping

- let add = fun x y -> x + y
- let addTwo = add 2
  - returns a closure → (function y -> x + y, {x: 2})
- If OCaml used Dynamic scoping,
  - let x = 5 in addTwo 3
    - It is almost impossible to keep the previous context

Homework 3 overview

MOcaml patterns

- p ::= intconst | boolconst | _ | var | (p1, …, pn) | C | C p
  - intconst ::= integer constant
  - boolconst ::= true | false
  - var ::= variable -- an identifier whose first letter is lowercase
  - C ::= data constructor -- an identifier whose first letter is uppercase

intconst	IntPat	int
boolconst	BoolPat	bool
_	WildcardPat	
var	VarPat	string
(p1, …, pn)	TuplePat	mopat list
C | C p	DataPat	string * mopat option

MOcaml expressions

- op ::= + | - | * | = | >
- e ::= intconst | boolconst | **var** | e1 op e2 | -e | if e1 then e2 else e3
  | function p -> e | e1 e2 | match e with p1 -> e2 '|' … '|' pn -> en
  | (e1, …, en) | C | C e

e1 op **e2**	BinOp	moexpr * moop * moexpr
if **e1** then **e2** else **e3**	If	moexpr * moexpr * moexpr
function p -> e	Function	mopat * moexpr
e1 e2	FunctionCall	moexpr * moexpr
match e with p1 -> e1 ...	Match	moexpr * (mopat * moexpr) list
(e1, e2, ... , en)	Tuple	moexpr list
C | C e	Data	string * moexpr option

MOCaml Declaration (modecl)

- d ::= e | let x = e | let rec f p = e

e	Expr	moexpr
let **x** = **e**	Let	string * moexpr
let rec **f p** = **e**	LetRec	string * moexpr

MOCaml Value (movalue)

- v ::= intconst | boolconst | function p -> e | (v1, …, vn) | C | C v

intconst	IntVal	int
boolconst	BoolVal	bool
function p -> e	FunctionVal	string option * mopat * moexpr * moenv
(v1, … , vn)	TupleVal	movalue list
C | C v	DataVal	string * movalue option

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)
  ```

MOCaml Environment Module

Methods

- Env.empty_env
  - Returns an empty environment (empty list)

- Env.add_binding: string -> 'a -> 'a env -> 'a env
  - Takes a variable name and value and adds it to the environment

- Env.combine_envs: 'a env -> 'a env -> 'a env
  - Takes two environments and merges them. The second will shadow the first.

- Env.lookup: string -> 'a env -> 'a
  - Looks for a binding in the environment. Throws “NotBound” exception if not found.

Understanding the interpreter

- let rec patMatch (pat:mopat) (value:movalue) : moenv
  - Check if the pattern and value map and return an environment if any names have to be bound.
- let rec evalExpr (e:moexpr) (env:moenv) : movalue
  - Evaluate the expr till you get a value.
- let rec evalDecl (d:modecl) (env:moenv) : moresult