

# CS 476 – Programming Language Design

William Mansky

## Questions

Nobody has responded yet.

Hang tight! Responses are coming in.

# From Typed Lambda Calculus to OCaml

- User-friendly syntax
  - Basic types, tuples, records
  - Inductive datatypes and pattern-matching
- Local declarations
- References
  - Type inference
  - Generics/polymorphism

# OCaml: Local Definitions

- In lambda calculus, every variable is either a function parameter or an undefined “free variable”
- OCaml has two kinds of local definitions:

```
let t1 = type_of e1 in subtype t1 t2
```

```
let x = 3;;
```

```
let y = x + 1;;
```

# OCaml: Local Declarations

- Can we treat this as syntactic sugar?

```
let x = 3;;
```

```
let y = x + 1;;
```

=>

```
let x = 3 in let y = x + 1;;
```

→

```
let y = 3 + 1;;
```

- Yes, but the intermediate states won't be very helpful!

# OCaml: Local Declarations

- Can we treat this as syntactic sugar?

```
let f x y = x + y;; (* let f = fun x y -> x + y *)
```

```
let g z = f z z;;
```

```
f (g 4) 3;;
```

```
=>
```

```
(fun x y -> x + y)
```

```
  (fun z -> (fun x y -> x + y) z z) 3;;
```

- Yes, but the intermediate states won't be very helpful!

# OCaml: Local Declarations

```
let x = 3;;
let y = x + 1;;
→
let y = x + 1;; (* {x = 3} *)
```

# OCaml: Local Declarations

$L ::= \dots$

$TD ::= \dots$

$T ::= \dots$

$C ::= \text{let } <\!\!\text{ident}\!\!> = L \mid C ; ; C \mid \text{skip}$

$P ::= TD ; ; \dots TD ; ; C ; ;$

# Local Declarations: Semantics

- As before, our configurations are now pairs of expression/command and environment

$$\frac{\rho(x) = v}{(x, \rho) \Downarrow v}$$

$$\frac{(l, \rho) \Downarrow v}{(\text{let } x = l, \rho) \rightarrow (\text{skip}, \rho[x \mapsto v])}$$

$$\frac{(c_1, \rho) \rightarrow (c'_1, \rho')}{(c_1 ; ; c_2, \rho) \rightarrow (c'_1 ; ; c_2, \rho')}$$

$$\frac{}{(\text{skip} ; ; c_2, \rho) \rightarrow (c_2, \rho)}$$

## Questions

Nobody has responded yet.

Hang tight! Responses are coming in.

# OCaml: Declarations and Scope

- At the top level, this gives us mutable variables!

```
let x = 3;;
```

```
let x = 4;;
```

→

```
let x = 4;; (* {x = 3} *)
```

→

```
(* skip *) (* {x = 4} *)
```

# OCaml: Declarations and Scope

- At the top level, this gives us mutable variables!

```
let x = 3;;
let f = fun y -> x + y;;
(* {x = 3; f = ?} *)
```

# OCaml: Declarations and Scope

- At the top level, this gives us mutable variables!

```
let x = 3;;
let f = fun y -> x + y;;
(* {x = 3; f = fun y -> x + y} *)
let x = 4;;
let z = f 2;;
```

- Exercise: What should the value of z be?

# OCaml: Declarations and Scope

- At the top level, this gives us mutable variables!

```
let x = 3;;
let f = fun y -> x + y;;           x is a free variable in f
(* {x = 3; f = fun y -> x + y} *)
let x = 4;;
let z = f 2;;    (* z = x + 2 = 6 *) (* !! *)
```

The code illustrates the scope of variables in OCaml. The first declaration `let x = 3;;` creates a mutable variable `x` at the top level. In the second declaration `let f = fun y -> x + y;;`, the variable `x` is a free variable, as indicated by a red wavy underline under `x` and a blue arrow pointing to the text "x is a free variable in f". This means that the value of `x` is not included in the closure of the function `f`. When `f` is applied to `2` in the fourth line, the result is `6` because the value of `x` has not been updated from its initial value of `3`.

# OCaml: Declarations and Scope

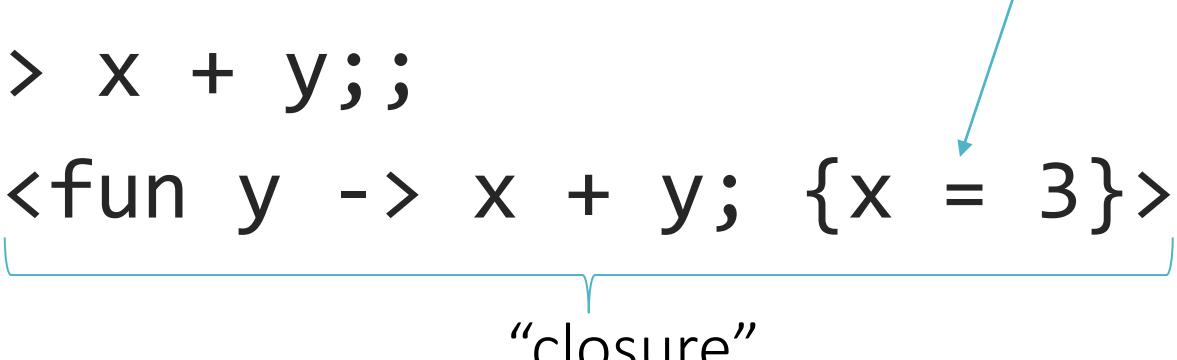
- We need function values to remember the values of variables!

```
let x = 3;;  
let f = fun y -> x + y;;      x is a free variable in f  
(* {x = 3; f = fun y -> x + y} *)    the value doesn't include the value of x  
let x = 4;;  
let z = f 2;;  (* z = x + 2 = 6 *) (* !! *)
```

# OCaml: Declarations and Scope

- We need function values to remember the values of variables!

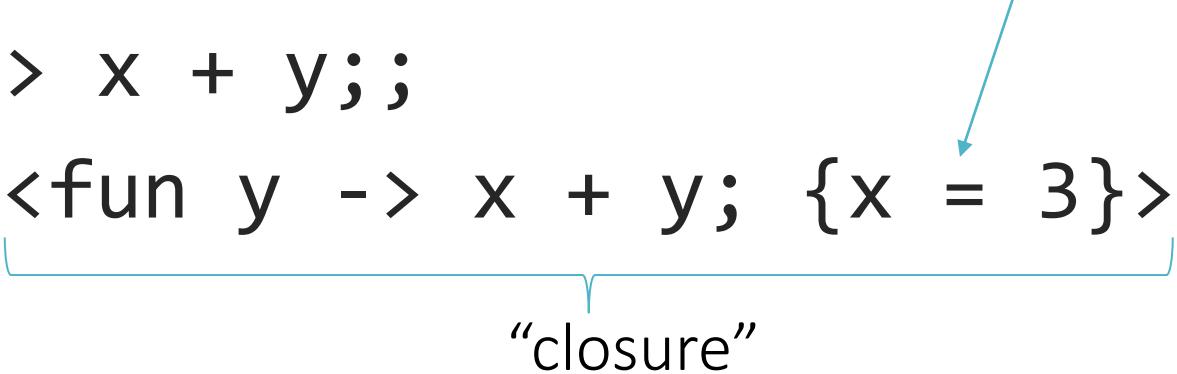
```
let x = 3;;                                "close" the function's free variables  
let f = fun y -> x + y;;  
(* {x = 3; f = <fun y -> x + y; {x = 3}>} *)  
let x = 4;;                                "closure"  
let z = f 2;;
```



# OCaml: Declarations and Scope

- We need function values to remember the values of variables!

```
let x = 3;;                                "close" the function's free variables  
let f = fun y -> x + y;;  
(* {x = 3; f = <fun y -> x + y; {x = 3}>} *)  
let x = 4;;                                "closure"  
let z = f 2;;    (* (x + 2 with {x = 3}) = 5 *)
```



# Local Declarations: Semantics

$$\frac{(l, \rho) \Downarrow v}{(\text{let } x = l, \rho) \rightarrow (\text{skip}, \rho[x \mapsto v])}$$

$$\overline{(\text{fun } x \rightarrow l, \rho) \Downarrow \text{fun } x \rightarrow l}$$

$$\frac{(l_1, \rho) \Downarrow \text{fun } x \rightarrow l \quad (l_2, \rho) \Downarrow v_2 \quad ([x \mapsto v_2]l, \rho) \Downarrow v}{(l_1 \ l_2, \rho) \Downarrow v}$$

# Local Declarations: Semantics

- Functions are no longer values, *closures* are

$$\frac{(l, \rho) \Downarrow v}{(\text{let } x = l, \rho) \rightarrow (\text{skip}, \rho[x \mapsto v])}$$

$$\frac{}{(\text{fun } x \rightarrow l, \rho) \Downarrow \langle \text{fun } x \rightarrow l, \rho \rangle}$$

$$\frac{(l_1, \rho) \Downarrow \text{fun } x \rightarrow l \quad (l_2, \rho) \Downarrow v_2 \quad ([x \mapsto v_2]l, \rho) \Downarrow v}{(l_1 \ l_2, \rho) \Downarrow v}$$

# Local Declarations: Semantics

- Functions are no longer values, *closures* are

$$\frac{(l, \rho) \Downarrow v}{(\text{let } x = l, \rho) \rightarrow (\text{skip}, \rho[x \mapsto v])}$$

$$\frac{}{(\text{fun } x \rightarrow l, \rho) \Downarrow \langle \text{fun } x \rightarrow l, \rho \rangle}$$

$$\frac{(l_1, \rho) \Downarrow \langle \text{fun } x \rightarrow l, \rho_1 \rangle \quad (l_2, \rho) \Downarrow v_2 \quad ([x \mapsto v_2]l, \rho_1) \Downarrow v}{(l_1 \ l_2, \rho) \Downarrow v}$$

# Local Declarations: Semantics

- Functions are no longer values, *closures* are

$$\frac{(l, \rho) \Downarrow v}{(\text{let } x = l, \rho) \rightarrow (\text{skip}, \rho[x \mapsto v])}$$

$$\frac{}{(\text{fun } x \rightarrow l, \rho) \Downarrow \langle \text{fun } x \rightarrow l, \rho \rangle}$$

$$\frac{(l_1, \rho) \Downarrow \langle \text{fun } x \rightarrow l, \rho_1 \rangle \quad (l_2, \rho) \Downarrow v_2 \quad (l, \rho_1[x \mapsto v_2]) \Downarrow v}{(l_1 \ l_2, \rho) \Downarrow v}$$

# Local Declarations: Semantics

- Functions are no longer values, *closures* are

- Substitution semantics:

$$(\text{fun } x \rightarrow \text{fun } y \rightarrow x+y) 2 \Downarrow \text{fun } y \rightarrow 2+y$$

- Closure semantics:

$$\frac{(l_1, \rho) \Downarrow \langle \text{fun } x \rightarrow l, \rho_1 \rangle \quad (l_2, \rho) \Downarrow v_2 \quad (l, \rho_1[x \mapsto v_2]) \Downarrow v}{(l_1 \ l_2, \rho) \Downarrow v}$$

# Local Declarations: Semantics

- Functions are no longer values, *closures* are

- Substitution semantics:

$$(\text{fun } x \rightarrow \text{fun } y \rightarrow x+y) 2 \Downarrow \text{fun } y \rightarrow 2+y$$

- Closure semantics:

$$\begin{aligned} & (\text{fun } x \rightarrow \text{fun } y \rightarrow x+y) 2 \\ & \Downarrow \langle \text{fun } y \rightarrow x+y, \{x = 2\} \rangle \end{aligned}$$

## Questions

Nobody has responded yet.

Hang tight! Responses are coming in.