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 ::= …

TD ::= …

T ::= …

C ::= `let` <ident> **=** *L* | *C*`;;` *C* | `skip`

P ::= *TD*`;;` … *TD*`;;` *C*`;;`

Local Declarations: Semantics

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

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

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

$$\frac{(c_1, \rho) \to (c_1', \rho')}{(c_1 ;; c_2, \rho) \to (c_1' ;; c_2, \rho')}$$

$$\frac{}{(\texttt{skip};; c_2, \rho) \to (c_2, \rho)}$$

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 = 3; f = fun y -> x + y} *)
let x = 4;;
let z = f 2;;  (* z = x + 2 = 6 *) (* !! *)
```

`x` is a free variable in `f`

the value doesn't include the value of `x`

OCaml: Declarations and Scope

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

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

`x` is a free variable in `f`

the value doesn't include the value of `x`

OCaml: Declarations and Scope

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

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

"close" the function's free variables

"closure"

OCaml: Declarations and Scope

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

```
let x = 3;;
let f = fun y -> x + y;;
(* {x = 3; f = <fun y -> x + y; {x = 3}>} *)
let x = 4;;
let z = f 2;;  (* (x + 2 with {x = 3}) = 5 *)
```

“close” the function’s free variables

“closure”

Local Declarations: Semantics

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

$$\frac{}{(\texttt{fun}\ x\ \texttt{->}\ l, \rho) \Downarrow \texttt{fun}\ x\ \texttt{->}\ l}$$

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

Local Declarations: Semantics

- Functions are no longer values, *closures* are

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

$$\frac{}{(\texttt{fun}\ x\ \texttt{->}\ l, \rho) \Downarrow \langle \texttt{fun}\ x\ \texttt{->}\ l, \rho \rangle}$$

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

Local Declarations: Semantics

- Functions are no longer values, *closures* are

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

$$\frac{}{(\texttt{fun}\ x\ \texttt{->}\ l, \rho) \Downarrow \langle \texttt{fun}\ x\ \texttt{->}\ l, \rho \rangle}$$

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

Local Declarations: Semantics

- Functions are no longer values, *closures* are

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

$$\frac{}{(\texttt{fun}\ x\ \texttt{->}\ l, \rho) \Downarrow \langle \texttt{fun}\ x\ \texttt{->}\ l, \rho \rangle}$$

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

Local Declarations: Semantics

- Functions are no longer values, *closures* are
- Substitution semantics:
  `(fun x -> fun y -> x+y)` 2 ⇓ `fun y -> 2+y`
- Closure semantics:

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

Local Declarations: Semantics

- Functions are no longer values, *closures* are
- Substitution semantics:

  $$\texttt{(fun x -> fun y -> x+y)}\ 2 \Downarrow \texttt{fun y -> 2+y}$$

- Closure semantics:

$$\texttt{(fun x -> fun y -> x+y)}\ 2$$
$$\Downarrow \langle \texttt{fun y -> x+y}, \{x = 2\} \rangle$$

Questions

Nobody has responded yet.

Hang tight! Responses are coming in.