Programming Languages and Compilers (CS 421)

Elsa L Gunter
2112 SC, UIUC
https://courses.engr.illinois.edu/cs421/sp2023

Based in part on slides by Mattox Beckman, as updated by Vikram Adve and Gul Agha
Features of OCAML

- Higher order applicative language
- Call-by-value parameter passing
- Modern syntax
- Parametric polymorphism
  - Aka structural polymorphism
- Automatic garbage collection
- User-defined algebraic data types
Why learn OCAML?

- Many features not clearly in languages you have already learned
- Assumed basis for much research in programming language research
- OCAML is particularly efficient for programming tasks involving languages (e.g., parsing, compilers, user interfaces)

Industrially Relevant:
- Jane Street trades billions of dollars per day using OCaml programs
- Major language supported at Bloomberg

Similar languages: Microsoft F#, SML, Haskell, Scala
% ocaml

Objective Caml version 4.07.1

# (* Read-eval-print loop; expressions and declarations *)

2 + 3;; (* Expression *)

- : int = 5

# 3 < 2;;

- : bool = false
Declarations; Sequencing of Declarations

# let x = 2 + 3;; (* declaration *)
val x : int = 5

# let test = 3 < 2;;
val test : bool = false

# let a = 1 let b = a + 4;; (* Sequence of dec *)
val a : int = 1
val b : int = 5
Functions

# let plus_two n = n + 2;;
val plus_two : int -> int = <fun>
# plus_two 17;;
- : int = 19
Functions

```ocaml
let plus_two n = n + 2;;
plus_two 17;;
- : int = 19
```
Environments

Environments record what value is associated with a given identifier.

Central to the semantics and implementation of a language.

Notation

\[ \rho = \{ \text{name}_1 \rightarrow \text{value}_1, \text{name}_2 \rightarrow \text{value}_2, \ldots \} \]

Using set notation, but describes a partial function.

Often stored as list, or stack.

- To find value start from left and take first match.
X $\Rightarrow$ 3

name $\Rightarrow$ “Steve”

y $\Rightarrow$ 17

region $\Rightarrow$ (5.4, 3.7)

b $\Rightarrow$ true

id $\Rightarrow$ \{Name = “Paul”, Age = 23, SSN = 999888777\}
Global Variable Creation

# 2 + 3;; (* Expression *)
// doesn’t affect the environment

# let test = 3 < 2;; (* Declaration *)
val test : bool = false
// ρ₁ = {test \rightarrow false}

# let a = 1 let b = a + 4;; (* Seq of dec *)
// ρ₂ = {b \rightarrow 5, a \rightarrow 1, test \rightarrow false}
Environments

- test ➔ true
- a ➔ 1
- b ➔ 5
// $\rho_2 = \{ b \rightarrow 5, a \rightarrow 1, \text{test} \rightarrow \text{false} \}$

let test = 3.7;;

- What is the environment after this declaration?
New Bindings Hide Old

```plaintext
// ρ₂ = {b → 5, a → 1, test → false}
let test = 3.7;;
```

What is the environment after this declaration?

```plaintext
// ρ₃ = {test → 3.7, a → 1, b → 5}
```
Environments

- test ➔ 3.7
- a ➔ 1
- b ➔ 5
Now it’s your turn

You should be able to start ACT1
Local Variable Creation

```ocaml
// ρ₃ = {test → 3.7, a → 1, b → 5}
# let b = 5 * 4
// ρ₄ = {b → 20, test → 3.7, a → 1}
   in 2 * b;;
- : int = 40

// ρ₅ = ρ₃ = {test → 3.7, a → 1, b → 5}
# b;;
- : int = 5
```
// \( \rho_5 = \rho_3 = \{\text{test} \rightarrow 3.7, a \rightarrow 1, b \rightarrow 5\} \)

# let c =
  let b = a + a

// \( \rho_6 = \{b \rightarrow 2\} + \rho_3 \)
// = \{b \rightarrow 2, \text{test} \rightarrow 3.7, a \rightarrow 1\}
  in b * b;;

val c : int = 4

// \( \rho_7 = \{c \rightarrow 4, \text{test} \rightarrow 3.7, a \rightarrow 1, b \rightarrow 5\} \)

# b;;

- : int = 5
let b = a + a
in b * b;;
val c : int = 4

let c =

val c : int = 4

// $\rho_5 =\rho_3 = \{test \rightarrow 3.7, a \rightarrow 1, b \rightarrow 5\}$

// $\rho_6 = \{b \rightarrow 2\} + \rho_3$

// $\rho_7 = \{c \rightarrow 4, test \rightarrow 3.7, a \rightarrow 1, b \rightarrow 5\}$
// ρ₅ = ρ₃ = {test → 3.7, a → 1, b → 5}
# let c =
  let b = a + a
  in b * b;;
val c : int = 4
// ρ₆ = {b → 2} + ρ₃
// = {b → 2, test → 3.7, a → 1}
// ρ₇ = {c → 4, test → 3.7, a → 1, b → 5}
# b;;
- : int = 5
Functions

# let plus_two n = n + 2;;
val plus_two : int -> int = <fun>
# plus_two 17;;
- : int = 19
let plus_two \( n \) = \( n + 2 \); 

plus_two 17;;

- : int = 19
fun n -> n + 2;;

(fun n -> n + 2) 17;;

- : int = 19
Functions

# let plus_two n = n + 2;;
val plus_two : int -> int = <fun>
# plus_two 17;;
- : int = 19

# let plus_two = fun n -> n + 2;;
val plus_two : int -> int = <fun>
# plus_two 14;;
- : int = 16

First definition syntactic sugar for second
Using a nameless function

# (fun x -> x * 3) 5;; (* An application *)
- : int = 15

# ((fun y -> y +. 2.0), (fun z -> z * 3));;
(* As data *)
- : (float -> float) * (int -> int) = (<fun>, <fun>)

Note: in fun v -> exp(v), scope of variable is only the body exp(v)
Values fixed at declaration time

let x = 12;;
val x : int = 12

let plus_x y = y + x;;
val plus_x : int -> int = <fun>

plus_x 3;;

What is the result?
Values fixed at declaration time

# let x = 12;;
val x : int = 12

# let plus_x y = y + x;;
val plus_x : int -> int = <fun>

# plus_x 3;;
- : int = 15
Values fixed at declaration time

# let x = 7;; (* New declaration, not an update *)

val x : int = 7

# plus_x 3;;

What is the result this time?
Values fixed at declaration time

```ocaml
# let x = 7;; (* New declaration, not an update *)
val x : int = 7
```

```ocaml
# plus_x 3;;
```

What is the result this time?
Values fixed at declaration time

# let x = 7;; (* New declaration, not an update *)
val x : int = 7

# plus_x 3;;
- : int = 15
Observation: Functions are first-class values in this language

Question: What value does the environment record for a function variable?

Answer: a closure
A *closure* is a pair of an environment and an association of a formal parameter (the input variables)* with an expression (the function body), written:

\[ f \rightarrow < (v_1, ..., v_n) \rightarrow \text{exp}, \rho_f > \]

Where \( \rho_f \) is the environment in effect when \( f \) is defined (if \( f \) is a simple function)

* Will come back to the “formal parameter”
Closure for plus_x

- When plus_x was defined, had environment:
  \[ \rho_{\text{plus}_x} = \{\ldots, x \rightarrow 12, \ldots\} \]

- Recall: let plus_x y = y + x
  is really let plus_x = fun y -> y + x

- Closure for fun y -> y + x:
  \[ <y \rightarrow y + x, \rho_{\text{plus}_x} > \]

- Environment just after plus_x defined:
  \[ \{\text{plus}_x \rightarrow <y \rightarrow y + x, \rho_{\text{plus}_x} >\} + \rho_{\text{plus}_x} \]
Now it’s your turn

You should be able to complete ACT1
125 minutes