Programming Languages and Compilers (CS 421)

Elsa L Gunter

2112 SC, UIUC

http://courses.engr.illinois.edu/cs421

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

Warm-up Scoping Question

Consider this code:

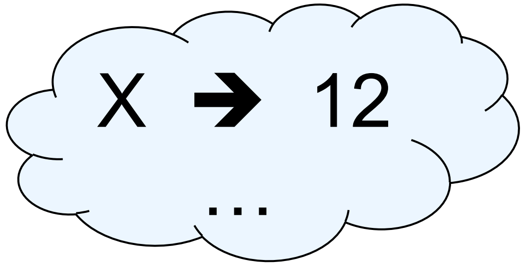
```
let x = 27;;
let f x =
      let x = 5 in
            (fun x -> print_int x) 10;;
f 12;;
```

What value is printed?

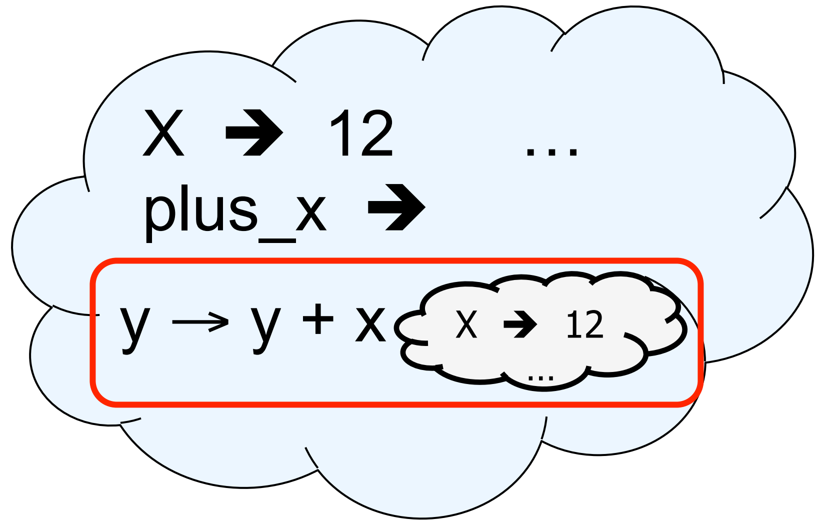
5
10
12
27

Recall: let plus_x = fun x => y + x

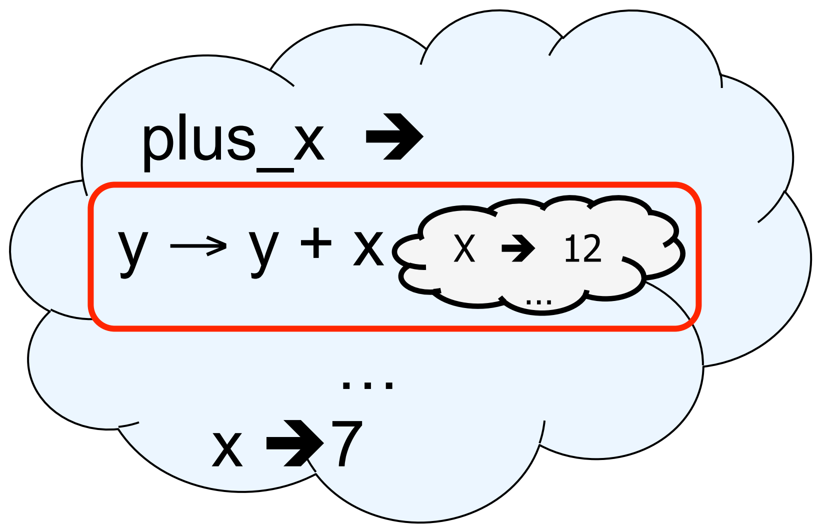
let x = 12



let plus_x = fun y => y + x



let x = 7



Closure for plus_x

- When plus_x was defined, had environment:

$$\rho_{plus\_x} = \{\ldots, x \rightarrow 12, \ldots\}$$

- Recall: let plus_x y = y + x

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

- Closure for fun y -> y + x:

$$<y \rightarrow y + x, \rho_{plus\_x}>$$

- Environment just after plus_x defined:

$$\{plus\_x \rightarrow <y \rightarrow y + x, \rho_{plus\_x}>\} + \rho_{plus\_x}$$

Functions on tuples

```
# let plus_pair (n,m) = n + m;;
val plus_pair : int * int -> int = <fun>
# plus_pair (3,4);;
- : int = 7
# let double x = (x,x);;
val double : 'a -> 'a * 'a = <fun>
# double 3;;
- : int * int = (3, 3)
# double "hi";;
- : string * string = ("hi", "hi")
```

Your turn now

Try Problem 1 on MP2

Save the Environment!

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

$$< (v1,\ldots,vn) \rightarrow exp, \rho >$$

- Where ρ is the environment in effect when the function is defined (for a simple function)

Closure for plus_pair

- Assume $\rho_{\text{plus\_pair}}$ was the environment just before plus_pair defined
- Closure for fun (n,m) -> n + m:

$$<(n,m) \rightarrow n + m, \rho_{\text{plus\_pair}}>$$

- Environment just after plus_pair defined:

$$\{\text{plus\_pair} \rightarrow <(n,m) \rightarrow n + m, \rho_{\text{plus\_pair}} >\} + \rho_{\text{plus\_pair}}$$

Your turn now

Try (* 1 *) from HW2

Functions with more than one argument

```
# let add_three x y z = x + y + z;;
val add_three : int -> int -> int -> int = <fun>
# let t = add_three 6 3 2;;
val t : int = 11
# let add_three =
   fun x -> (fun y -> (fun z -> x + y + z));;
val add_three : int -> int -> int -> int = <fun>
```

Again, first syntactic sugar for second

Your turn now

Try Problem 2 on MP2

Curried vs Uncurried

- Recall

val add_three : int -> int -> int -> int = <fun>

- How does it differ from

let add_triple (u,v,w) = u + v + w;;

val add_triple : int * int * int -> int = <fun>

- add_three is *curried*;
- add_triple is *uncurried*

Curried vs Uncurried

```
# add_triple (6,3,2);;
- : int = 11
# add_triple 5 4;;
Characters 0-10:
  add_triple 5 4;;
  ^^^^^^^^^^
This function is applied to too many arguments,
maybe you forgot a `;'
# fun x -> add_triple (5,4,x);;
: int -> int = <fun>
```

Partial application of functions

```
let add_three x y z = x + y + z;;
```

```
# let h = add_three 5 4;;
val h : int -> int = <fun>
# h 3;;
- : int = 12
# h 7;;
- : int = 16
```

Your turn now

Try (* 2 *) from HW2

Caution!

Know what the argument is and what the body is

Functions as arguments

```
# let thrice f x = f (f (f x));;
val thrice : ('a -> 'a) -> 'a -> 'a = <fun>
# let g = thrice plus_two;;
val g : int -> int = <fun>
# g 4;;
- : int = 10
# thrice (fun s -> "Hi! " ^ s) "Good-bye!";;
- : string = "Hi! Hi! Hi! Good-bye!"
```

Your turn now

Try Problem 3 on MP2

Evaluating declarations

- Evaluation uses an environment ρ
- To evaluate a (simple) declaration let x = e
  - Evaluate expression e in ρ to value v
  - Update ρ with x v: $\{x \rightarrow v\} + \rho$
- Eval(let x = e, ρ) = $\{x \rightarrow \text{Eval}(e, \rho)\} + \rho$
- Update: $\rho_1 + \rho_2$ has all the bindings in ρ_1 and all those in ρ_2 that are not rebound in ρ_1

$\{x \rightarrow 2, y \rightarrow 3, a \rightarrow \text{“hi”}\} + \{y \rightarrow 100, b \rightarrow 6\}$

$= \{x \rightarrow 2, y \rightarrow 3, a \rightarrow \text{“hi”}, b \rightarrow 6\}$

Evaluating expressions

- Evaluation uses an environment ρ
- A constant evaluates to itself
- To evaluate an variable, look it up in ρ ($\rho(v)$)
- To evaluate uses of +, -, *, etc, eval args, then do operation
- Function expression evaluates to its closure
- To evaluate a local dec: let x = e1 in e2
  - Eval e1 to v, eval e2 using $\{x \rightarrow v\} + \rho$

Eval of Expressions as Rewrites

- $\mathrm{Eval}(v, \rho) = v$ if v is a constant (a value)
- $\mathrm{Eval}(x, \rho) = \rho(x)$ if x is a variable
- $\mathrm{Eval}(e_1+e_2, \rho) = (\mathrm{Eval}(e_1, \rho))+(\mathrm{Eval}(e_2, \rho))$
- Eval(fun $(x_1,\ldots,x_n)$ -> body, ρ) = $<(x_1,\ldots,x_n) \to \text{body}, \rho>$
- Eval(let $x = e_1$ in e_2, ρ) = $\mathrm{Eval}(e_2, \{x \to \mathrm{Eval}(e_1, \rho)\} + \rho)$
- $\mathrm{Eval}(f\ e, \rho) =$ $\mathrm{Eval}(\mathrm{app}(\mathrm{Eval}(f, \rho),\mathrm{Eval}(e, \rho)) , \rho)$

Evaluation of Application with Closures

- In environment ρ, evaluate left term to closure, $<(x_1,\ldots,x_n) \rightarrow b, \rho>$
- $(x_1,\ldots,x_n)$ variables in (first) argument
- Evaluate the right term to values, $(v_1,\ldots,v_n)$
- Update the environment ρ to

  $\rho' = \{x_1 \rightarrow v_1,\ldots, x_n \rightarrow v_n\} + \rho$
- Evaluate body b in environment ρ'

Eval Application of Closure

$$\text{Eval}(\text{app}(<(x_1,\ldots,x_n) \rightarrow b,\ \rho>,\ (v_1,\ldots,v_n)),\ \rho') =$$

$$\text{Eval}(b,\ \{x_1 \rightarrow v_1,\ldots,\ x_n \rightarrow v_n\} + \rho)$$

Evaluation of Application of plus_x;;

- Have environment:

  $\rho = \{plus\_x \rightarrow <y \rightarrow y + x, \rho_{plus\_x} >, \ldots ,$
  $y \rightarrow 3, \ldots\}$

  where $\rho_{plus\_x} = \{x \rightarrow 12, \ldots\}$
- Eval (plus_x y, ρ) rewrites to
- Eval(app (<y → y + x, ρ_{plus_x} >, 3) ρ) rewrites to
- Eval (y + x, {y → 3} +ρ_{plus_x}) rewrites to
- Eval (3 + 12 , ρ_{plus_x}) = 15

Evaluation of Application of plus_pair

- Assume environment

$\rho = \{x \rightarrow 3\ldots,$

$\quad plus\_pair \rightarrow <(n,m) \rightarrow n + m, \rho_{plus\_pair}>\} +$

$\quad \rho_{plus\_pair}$

- Eval (plus_pair (4,x), ρ)=
- App (<(n,m) $\rightarrow$ n + m, ρ_{plus_pair}>, (4,3)) =
- Eval (n + m, {n -> 4, m -> 3} + ρ_{plus_pair}) =
- Eval (4 + 3, {n -> 4, m -> 3} + ρ_{plus_pair}) = 7

Your turn now

Try (* 3 *) from HW2

Closure question

- If we start in an empty environment, and we execute:

```
let f = fun n -> n + 5;;
(* 0 *)
let pair_map g (n,m) = (g n, g m);;
let f = pair_map f;;
let a = f (4,6);;
```

What is the environment at (* 0 *)?

Answer

let f = fun n -> n + 5;;

$\rho_0 = \{f \rightarrow <n \rightarrow n + 5, \{ \}>\}$

Closure question

- If we start in an empty environment, and we execute:

```
let f = fun => n + 5;;
let pair_map g (n,m) = (g n, g m);;
(* 1 *)
let f = pair_map f;;
let a = f (4,6);;
```

What is the environment at (* 1 *)?

Answer

$\rho_0 = \{f \to$ <$n \to n + 5, \{\ \}$>$\}$

let pair_map g (n,m) = (g n, g m);;

$\rho_1 = \{$pair_map $\to$

<g $\to$ fun (n,m) -> (g n, g m),

$\{f \to$ <$n \to n + 5, \{\ \}$>$\}$>,

$f \to$ <$n \to n + 5, \{\ \}$>$\}$

Closure question

- If we start in an empty environment, and we execute:

```
let f = fun => n + 5;;
let pair_map g (n,m) = (g n, g m);;
let f = pair_map f;;
(* 2 *)
let a = f (4,6);;
```

What is the environment at (* 2 *)?

Evaluate pair_map f

ρ_0 = {f → <n → n + 5, { }>}

ρ_1 = {pair_map →<g→fun (n,m) -> (g n, g m), ρ_0>,
f→<n → n + 5, { }>}

let f = pair_map f;;

Evaluate pair_map f

ρ_0 = {f $\rightarrow$ <n $\rightarrow$ n + 5, { }>}

ρ_1 = {pair_map $\rightarrow$<g$\rightarrow$fun (n,m) -> (g n, g m), ρ_0>,
f$\rightarrow$<n $\rightarrow$ n + 5, { }>}

Eval(pair_map f, ρ_1) =

Evaluate pair_map f

ρ_0 = {f → <n → n + 5, { }>}

ρ_1 = {pair_map →<g→fun (n,m) -> (g n, g m), ρ_0>,
f→<n → n + 5, { }>}

Eval(pair_map f, ρ_1) =

Eval(app (<g→fun (n,m) -> (g n, g m), ρ_0>,
<n → n + 5, { }>), ρ_1) =

Evaluate pair_map f

ρ_0 = {f → <n → n + 5, { }>}

ρ_1 = {pair_map → <g→fun (n,m) -> (g n, g m), ρ_0>,
f → <n → n + 5, { }>}

Eval(pair_map f, ρ_1) =

Eval(app (<g→fun (n,m) -> (g n, g m), ρ_0>,
<n → n + 5, { }>), ρ_1) =

Evaluate pair_map f

ρ_0 = {f $\to$ <n $\to$ n + 5, { }>}

ρ_1 = {pair_map $\to$<g$\to$fun (n,m) -> (g n, g m), ρ_0>,

f$\to$<n $\to$ n + 5, { }>}

Eval(pair_map f, ρ_1) =

Eval(app (<g$\to$fun (n,m) -> (g n, g m), ρ_0>,

<n $\to$ n + 5, { }>), ρ_1) =

Eval(fun (n,m)->(g n, g m), {g$\to$<n$\to$n + 5, { }>}+ρ_0)

=<(n,m) $\to$(g n, g m), {g$\to$<n$\to$n + 5, { }>}+ρ_0>

=<(n,m) $\to$(g n, g m), {g$\to$<n$\to$n + 5, { }>

f$\to$<n$\to$n + 5, { }>}

Evaluate pair_map f

ρ_0 = {f → <n → n + 5, { }>}

ρ_1 = {pair_map →<g→fun (n,m) -> (g n, g m), ρ_0>,
f→<n → n + 5, { }>}

Eval(pair_map f, ρ_1) =

Eval(app (<g→fun (n,m) -> (g n, g m), ρ_0>,
<n → n + 5, { }>), ρ_1) =

Eval(fun (n,m)->(g n, g m), {g→<n→n + 5, { }>}+ρ_0)

=<(n,m) →(g n, g m), {g→<n→n + 5, { }>}+ρ_0>

=<(n,m) →(g n, g m), {g→<n→n + 5, { }>
f→<n→n + 5, { }>}

Evaluate pair_map f

ρ_0 = {f → <n → n + 5, { }>}

ρ_1 = {pair_map →<g→fun (n,m) -> (g n, g m), ρ_0>,

f→<n → n + 5, { }>}

Eval(pair_map f, ρ_1) =

Eval(app (<g→fun (n,m) -> (g n, g m), ρ_0>,

<n → n + 5, { }>), ρ_1) =

Eval(fun (n,m)->(g n, g m), {g→<n→n + 5, { }>}+ρ_0)

=<(n,m) →(g n, g m), {g→<n→n + 5, { }>}+ρ_0>

=<(n,m) →(g n, g m), {g→<n→n + 5, { }>

f→<n→n + 5, { }>}

Answer

ρ_1 = {pair_map →
<g → fun (n,m) -> (g n, g m),{f → <n → n + 5, { }>}>,
f → <n → n + 5, { }>}

let f = pair_map f;;

ρ_2 = {f → <(n,m) →(g n, g m),
{g → <n → n + 5, { }>,
f → <n → n + 5, { }>}>,
pair_map → <g → fun (n,m) -> (g n, g m),
{f → <n → n + 5, { }>}>}

Closure question

- If we start in an empty environment, and we execute:

```
let f = fun => n + 5;;
let pair_map g (n,m) = (g n, g m);;
let f = pair_map f;;
let a = f (4,6);;
```

(* 3 *)

What is the environment at (* 3 *)?

Final Evalution?

ρ_2 = {f → <(n,m) →(g n, g m),
{g → <n → n + 5, { }>,
f → <n → n + 5, { }>}>,
pair_map → <g → fun (n,m) -> (g n, g m),
{f → <n → n + 5, { }>}>}

let a = f (4,6);;

Evaluate f (4,6);;

ρ_2 = {f → <(n,m) →(g n, g m),
{g → <n → n + 5, { }>,
f → <n → n + 5, { }>}>,
pair_map → <g → fun (n,m) -> (g n, g m),
{f → <n → n + 5, { }>}>}

Eval(f (4,6), ρ_2) =

Evaluate f (4,6);;

ρ_2 = {f → <(n,m) →(g n, g m),
{g → <n → n + 5, { }>,
f → <n → n + 5, { }>}>,
pair_map → <g → fun (n,m) -> (g n, g m),
{f → <n → n + 5, { }>}>}

Eval(f (4,6), ρ_2) =

Eval(app(<(n,m) →(g n, g m),
{g → <n → n + 5, { }>,
f → <n → n + 5, { }>}>,(4,6)), ρ_2) =

Evaluate f (4,6);;

ρ_2 = {$\boxed{\text{f}}$ → $\boxed{\text{<(n,m) →(g n, g m), \{g → <n → n + 5, \{ \}>, f → <n → n + 5, \{ \}>\}>}}$,

pair_map → <g → fun (n,m) -> (g n, g m),

{f → <n → n + 5, { }>}>}

Eval($\boxed{\text{f}}$ $\boxed{\text{(4,6)}}$, ρ_2) =

Eval(app($\boxed{\text{<(n,m) →(g n, g m), \{g → <n → n + 5, \{ \}>, f → <n → n + 5, \{ \}>\}>}}$,$\boxed{\text{(4,6)}}$), ρ_2) =

Evaluate f (4,6);;

Eval(app(<(n,m) →(g n, g m),
{g → <n → n + 5, { }>,
f → <n → n + 5, { }>}>,(4,6)), ρ_2) =

Eval((g n, g m), {n → 4, m → 6} +
{g → <n → n + 5, { }>,
f → <n → n + 5, { }>}) =

Eval((app(<n → n + 5, { }>, 4),
app (<n → n + 5, { }>, 6)),
{n → 4, m → 6, g → <n → n + 5, { }>,
f → <n → n + 5, { }>}) =

Evaluate f (4,6);;

Eval(app(<(n,m) → (g n, g m),
{g → <n → n + 5, { }>,
f → <n → n + 5, { }>}>, (4,6)), ρ_2) =

Eval((g n, g m), {n → 4, m → 6} +
{g → <n → n + 5, { }>,
f → <n → n + 5, { }>}) =

Eval((app(<n → n + 5, { }>, 4),
app (<n → n + 5, { }>, 6)),
{n → 4, m → 6, g → <n → n + 5, { }>,
f → <n → n + 5, { }>}) =

Evaluate f (4,6);;

Eval(app(<(n,m) →(g n, g m),

{g → <n → n + 5, { }>,

f → <n → n + 5, { }>}>,(4,6)), ρ_2) =

Eval((g n, g m), {n → 4, m → 6} +

{g → <n → n + 5, { }>,

f → <n → n + 5, { }>}) =

Eval((app(<n → n + 5, { }>, 4),

app (<n → n + 5, { }>, 6)),

{n → 4, m → 6, g → <n → n + 5, { }>,

f → <n → n + 5, { }>}) =

Evaluate f (4,6);;

$\rho_3 = \{n \to 4, m \to 6, g \to <n \to n + 5, \{ \}>,$

$f \to <n \to n + 5, \{ \}>\})$

$\text{Eval}((\text{app}(<n \to n + 5, \{ \}>, 4),$

$\text{app}(<n \to n + 5, \{ \}>, 6)), \rho_3) =$

$\text{Eval}((\text{Eval}(n + 5, \{n \to 4\} + \{ \}),$

$(\text{Eval}(n + 5, \{n \to 6\} + \{ \})), \rho_3) =$

$\text{Eval}((\text{Eval}(4 + 5, \{n \to 4\} + \{ \}),$

$(\text{Eval}(6 + 5, \{n \to 6\} + \{ \})), \rho_3) =$

$\text{Eval}((9, 11), \rho_3) = (9. 11)$

Your turn now

Try (* 4 *) from HW2

Match Expressions

```
# let triple_to_pair triple =
  match triple
  with (0, x, y) -> (x, y)
  | (x, 0, y) -> (x, y)
  | (x, y, _) -> (x, y);;
val triple_to_pair : int * int * int -> int * int =
  <fun>
```

- Each clause: pattern on left, expression on right
- Each x, y has scope of only its clause
- Use first matching clause

Recursive Functions

```
# let rec factorial n =
    if n = 0 then 1 else n * factorial (n - 1);;
  val factorial : int -> int = <fun>
# factorial 5;;
- : int = 120
# (* rec  is needed for recursive function
   declarations *)
```

Your turn now

Try Problem 4 on MP2

Recursion Example

Compute n^2 recursively using:

$$n^2 = (2 * n - 1) + (n - 1)^2$$

```
# let rec nthsq n =          (* rec for recursion *)
   match n                   (* pattern matching for cases *)
   with 0 -> 0               (* base case *)
   | n -> (2 * n -1)         (* recursive case *)
         + nthsq (n -1);;    (* recursive call *)
val nthsq : int -> int = <fun>
# nthsq 3;;
- : int = 9
```

Structure of recursion similar to inductive proof

Recursion and Induction

```
# let rec nthsq n = match n with 0 -> 0
      | n -> (2 * n - 1) + nthsq (n - 1) ;;
```

- Base case is the last case; it stops the computation
- Recursive call must be to arguments that are somehow smaller - must progress to base case
- **if** or **match** must contain base case
- Failure of these may cause failure of termination