

## 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

8/23/16

1

## Contact Information - Elsa L Gunter

- Office: 2112 SC
- Office hours:
  - Tuesday 12:00pm – 1:30pm
  - Thursday 3:30pm - 4:20pm
  - Also by appointment
- Email: [egunter@illinois.edu](mailto:egunter@illinois.edu)

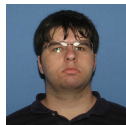
8/23/16

2

## Course TAs



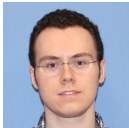
John Lee



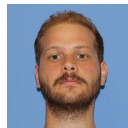
Kevin Banker



Tom Bogue



Eric Huber



Deniz Arsan

8/23/16

3

## Contact Information - TAs

- Teaching Assistants Office: 0207 SC
- John Lee
  - Email: [lee170@illinois.edu](mailto:lee170@illinois.edu)
  - Hours: Mon 10:00am – 10:50am  
Fri 10:00am – 10:50am
- Tom Bogue
  - Email: [tbogue2@illinois.edu](mailto:tbogue2@illinois.edu)
  - Hours: Wed 10:00am – 10:50am  
Fri 12:30pm – 1:20pm

8/23/16

4

## Contact Information – TAs cont

- Kevin Banker
  - Email: [banker2@illinois.edu](mailto:banker2@illinois.edu)
  - Hours: Tues 10:00am – 10:50am  
Thurs 12:30pm – 1:20pm
- Eric Huber
  - Email: [echuber2@illinois.edu](mailto:echuber2@illinois.edu)
  - Hours: Mon 3:30pm – 4:20pm  
Wed 1:00pm – 1:50pm

8/23/16

5

## Contact Information – TAs cont

- Deniz Arsan
  - Email: [darsan2@illinois.edu](mailto:darsan2@illinois.edu)
  - Hours: Mon 2:00pm – 2:50pm  
Wed 3:30pm – 4:20pm

8/23/16

6

## Course Website

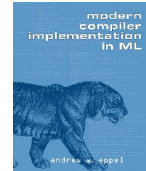
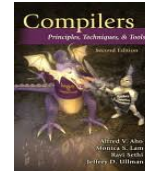
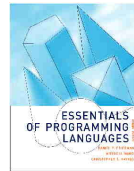
- <http://courses.engr.illinois.edu/cs421>
- Main page - summary of news items
- Policy - rules governing course
- Lectures - syllabus and slides
- MPs - information about assignments
- Exams
- Unit Projects - for 4 credit students
- Resources - tools and helpful info
- FAQ

8/23/16

7

## Some Course References

- No required textbook
- Some suggested references



8/23/16

8

## Some Course References

- No required textbook.
- Pictures of the books on previous slide
- Essentials of Programming Languages (2nd Edition) by Daniel P. Friedman, Mitchell Wand and Christopher T. Haynes, MIT Press 2001.
- Compilers: Principles, Techniques, and Tools, (also known as "The Dragon Book"); by Aho, Sethi, and Ullman. Published by Addison-Wesley. ISBN: 0-201-10088-6.
- Modern Compiler Implementation in ML by Andrew W. Appel, Cambridge University Press 1998
- Additional ones for Ocaml given separately

8/23/16

9

## Course Grading

- Assignments 20%
  - About 12 Web-based Assignments (WA) (~7%)
    - May be converted to Hand Written (HW) if our web development is inadequate
  - About 8 MPs (in Ocaml) (~7%)
  - About 4 Labs (~6%)
  - All MPs Submitted by **svn**
    - MPs – plain text code that compiles; HWs – pdf
  - Late submission penalty for MLs and WA: 20% of assignments total value
- Two Midterms (CBTF) %20 each
- Final (CBTF) %40

8/23/16

10

## Computer-Based Testing Facility

- Labs and exams in Computer-Based Testing Center (Basement of Granger)
  - Self-scheduled
  - Over a four day period
  - No extensions beyond the four day period
  - Fall back:
    - Labs become MPs
    - Exams are given in class on the date marked for the exam

8/23/16

11

## Course Assignments – WA & MP

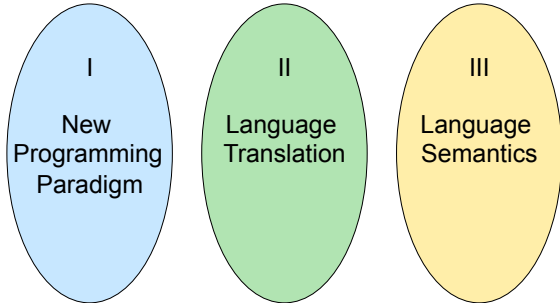
- You may discuss assignments and their solutions with others
- You may work in groups, but you must **list members with whom you worked** if you share solutions or solution outlines
- **Each student must write up and turn in their own solution separately**
- You may look at examples from class and other similar examples from any source – cite appropriately
  - Note: University policy on plagiarism still holds - cite your sources if you are not the sole author of your solution

8/23/16

12

# Programming Languages & Compilers

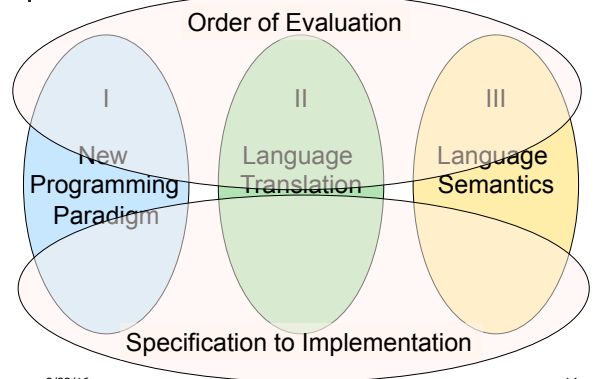
## Three Main Topics of the Course



8/23/16

13

# Programming Languages & Compilers

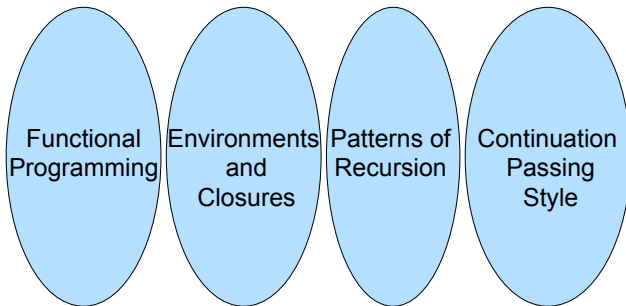


8/23/16

14

# Programming Languages & Compilers

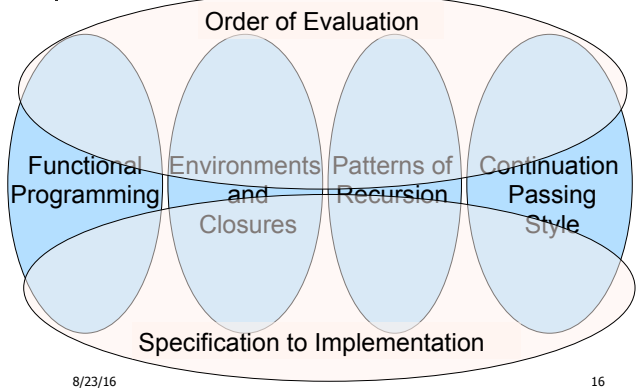
## I : New Programming Paradigm



8/23/16

15

# Programming Languages & Compilers

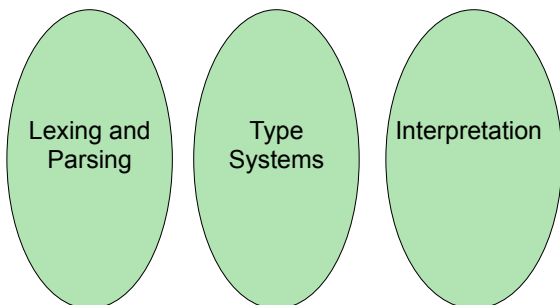


8/23/16

16

# Programming Languages & Compilers

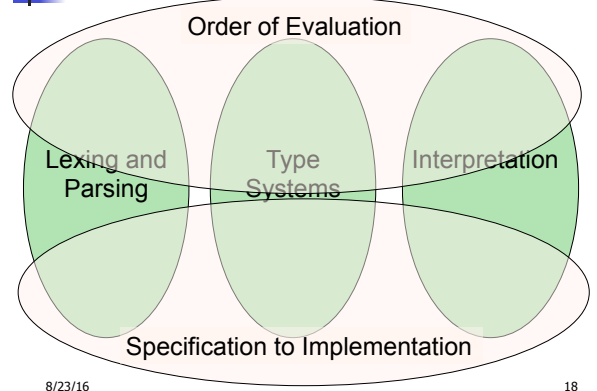
## II : Language Translation



8/23/16

17

# Programming Languages & Compilers

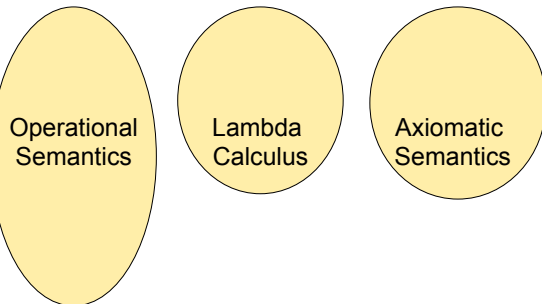


8/23/16

18

## Programming Languages & Compilers

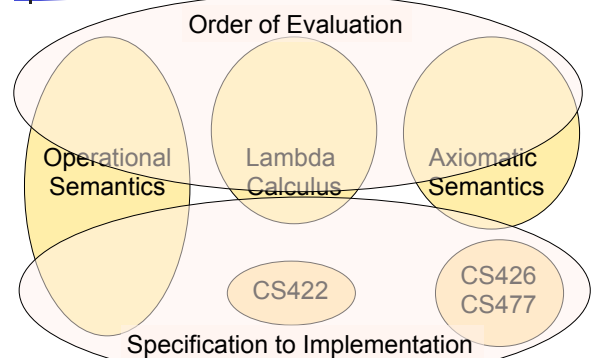
### III : Language Semantics



8/23/16

19

## Programming Languages & Compilers



8/23/16

20

## Course Objectives

- New programming paradigm
  - Functional programming
  - Environments and Closures
  - Patterns of Recursion
  - Continuation Passing Style
- Phases of an interpreter / compiler
  - Lexing and parsing
  - Type systems
  - Interpretation
- Programming Language Semantics
  - Lambda Calculus
  - Operational Semantics
  - Axiomatic Semantics

8/23/16

21

## OCAML

- Locally:
  - Compiler is on the EWS-linux systems at `/usr/local/bin/ocaml`
- Globally:
  - Main CAML home: <http://ocaml.org>
  - To install OCAML on your computer see: <http://ocaml.org/docs/install.html>
  - To run ocaml in a browser: [http://ocsigen.org/js\\_of\\_ocaml/2.8.1/files/toplevel/index.html](http://ocsigen.org/js_of_ocaml/2.8.1/files/toplevel/index.html)

8/23/16

22

## References for OCaml

- Supplemental texts (not required):
  - The Objective Caml system release 4.03, by Xavier Leroy, online manual
  - Introduction to the Objective Caml Programming Language, by Jason Hickey
  - Developing Applications With Objective Caml, by Emmanuel Chailoux, Pascal Manoury, and Bruno Pagano, on O'Reilly
    - Available online from course resources

8/23/16

23

## OCAML Background

- CAML is European descendant of original ML
  - American/British version is SML
  - O is for object-oriented extension
- ML stands for Meta-Language
- ML family designed for implementing theorem provers
  - It was the meta-language for programming the "object" language of the theorem prover
  - Despite obscure original application area, OCAML is a full general-purpose programming language

8/23/16

24

## 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
  
- It's fast - winners of the 1999 and 2000 ICFP Programming Contests used OCAML

8/23/16

25

## 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 (eg parsing, compilers, user interfaces)
- Industrially Relevant: Jane Street trades billions of dollars per day using OCaml programs
- Similar languages: Microsoft F#, SML, Haskell, Scala

8/23/16

26

## OCaml Intro Code

- A (possibly better, non-PowerPoint) text version of this lecture can be found at <http://course.engr.illinois.edu/class/cs421/lectures/ocaml-intro-shell.txt>
- For the OCAML code for today's lecture see <http://course.engr.illinois.edu/class/cs421/lectures/ocaml-intro.ml>

8/23/16

27

## Session in OCAML

```
% ocaml
Objective Caml version 4.01
# (* Read-eval-print loop; expressions and
  declarations *)
  2 + 3;; (* Expression *)
- : int = 5
# 3 < 2;;
- : bool = false
```

8/23/16

28

## No Overloading for Basic Arithmetic Operations

```
# 15 * 2;;
- : int = 30
# 1.35 + 0.23;; (* Wrong type of addition *)
Characters 0-4:
  1.35 + 0.23;; (* Wrong type of addition *)
  ^^^^
Error: This expression has type float but an
expression was expected of type
  int
# 1.35 +. 0.23;;
- : float = 1.58
```

8/23/16

29

## No Implicit Coercion

```
# 1.0 * 2;; (* No Implicit Coercion *)
Characters 0-3:
  1.0 * 2;; (* No Implicit Coercion *)
  ^^^
Error: This expression has type float but an
expression was expected of type
  int
```

8/23/16

30

## Sequencing Expressions

```
# "Hi there";; (* has type string *)
- : string = "Hi there"
# print_string "Hello world\n";; (* has type unit *)
Hello world
- : unit = ()
# (print_string "Bye\n"; 25);; (* Sequence of exp *)
Bye
- : int = 25
```

8/23/16

31

## 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
```

8/23/16

32

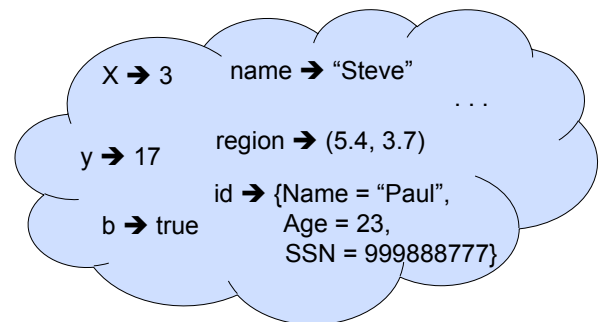
## Environments

- *Environments* record what value is associated with a given identifier
- Central to the semantics and implementation of a language
- Notation
  - $\rho = \{name_1 \rightarrow value_1, name_2 \rightarrow value_2, \dots\}$
  - Using set notation, but describes a partial function
- Often stored as list, or stack
  - To find value start from left and take first match

8/23/16

33

## Environments



8/23/16

34

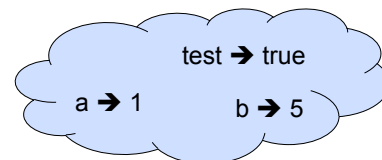
## Global Variable Creation

```
# 2 + 3;; (* Expression *)
// doesn't affect the environment
# let test = 3 < 2;; (* Declaration *)
val test : bool = false
//  $\rho_1 = \{test \rightarrow false\}$ 
# let a = 1 let b = a + 4;; (* Seq of dec *)
//  $\rho_2 = \{b \rightarrow 5, a \rightarrow 1, test \rightarrow false\}$ 
```

8/23/16

35

## Environments



8/23/16

36

## New Bindings Hide Old

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

- What is the environment after this declaration?

8/23/16

37

## New Bindings Hide Old

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

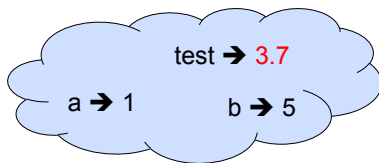
- What is the environment after this declaration?

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

8/23/16

38

## Environments



8/23/16

39

Now it's your turn

You should be able to do WA1  
Problem 1, parts (\* 1 \*) and (\* 2 \*)

8/23/16

40

## Local Variable Creation

```
// 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
```

8/23/16

41

## Local let binding

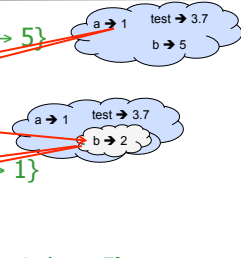
```
// ρ5 = {test → 3.7, a → 1, b → 5}  
# let c =  
  let b = a + a  
  // ρ6 = {b → 2} + ρ5  
  //   = {b → 2, test → 3.7, a → 1}  
  in b * b;;  
val c : int = 4  
// ρ7 = {c → 4, test → 3.7, a → 1, b → 5}  
# b;;  
- : int = 5
```

8/23/16

42

## Local let binding

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

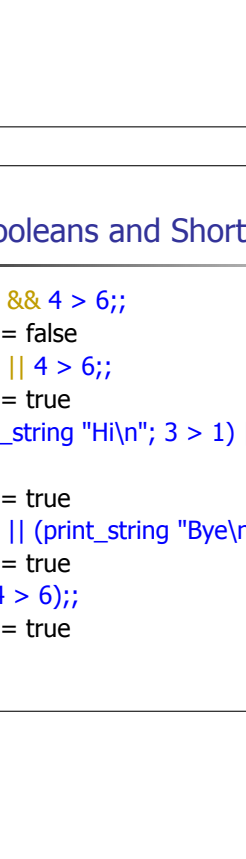


8/23/16

43

## Local let binding

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



8/23/16

44

Now it's your turn

You should be able to do WA1  
Problem 1 , parts (\* 3 \*) and (\* 4 \*)

8/23/16

45

## Booleans (aka Truth Values)

```
# true;;
- : bool = true
# false;;
- : bool = false
// ρ7 = {c → 4, test → 3.7, a → 1, b → 5}
# if b > a then 25 else 0;;
- : int = 25
```

8/23/16

46

## Booleans and Short-Circuit Evaluation

```
# 3 > 1 && 4 > 6;;
- : bool = false
# 3 > 1 || 4 > 6;;
- : bool = true
# (print_string "Hi\n"; 3 > 1) || 4 > 6;;
Hi
- : bool = true
# 3 > 1 || (print_string "Bye\n"; 4 > 6);;
- : bool = true
# not (4 > 6);;
- : bool = true
```

8/23/16

47

Now it's your turn

You should be able to do WA1  
Problem 1 , part (\* 5 \*)

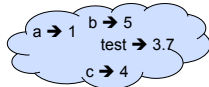
8/23/16

48



## Tuples as Values

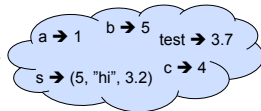
```
// ρ7 = {c → 4, test → 3.7,  
          a → 1, b → 5}
```



```
# let s = (5, "hi", 3.2);;
```

```
val s : int * string * float = (5, "hi", 3.2)
```

```
// ρ8 = {s → (5, "hi", 3.2),  
          c → 4, test → 3.7,  
          a → 1, b → 5}
```

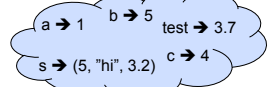


8/23/16

49

## Pattern Matching with Tuples

```
/ ρ8 = {s → (5, "hi", 3.2),  
        c → 4, test → 3.7,  
        a → 1, b → 5}
```

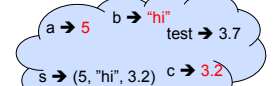


```
# let (a,b,c) = s;; (* (a,b,c) is a pattern *)
```

```
val a : int = 5
```

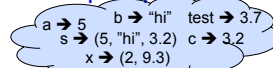
```
val b : string = "hi"
```

```
val c : float = 3.2
```



```
# let x = 2, 9.3;; (* tuples don't require parens in  
                  Ocaml *)
```

```
val x : int * float = (2, 9.3)
```



8/23/16

50

## Nested Tuples

```
# (*Tuples can be nested *)
```

```
let d = ((1,4,62),("bye",15),73.95);;
```

```
val d : (int * int * int) * (string * int) * float =  
        ((1, 4, 62), ("bye", 15), 73.95)
```

```
# (*Patterns can be nested *)
```

```
let (p,(st,_,_) = d;; (* _ matches all, binds nothing  
                      *)
```

```
val p : int * int * int = (1, 4, 62)
```

```
val st : string = "bye"
```

8/23/16

51

Now it's your turn

You should be able to do WA1  
through step

8/23/16

52

## Functions

```
# let plus_two n = n + 2;;
```

```
val plus_two : int -> int = <fun>
```

```
# plus_two 17;;
```

```
- : int = 19
```

8/23/16

53

## Functions

```
let plus_two n = n + 2;;
```

```
plus_two 17;;
```

```
- : int = 19
```

8/23/16

54

## Nameless Functions (aka Lambda Terms)

```
fun n -> n + 2;;  
(fun n -> n + 2) 17;;  
- : int = 19
```



8/23/16

55

## 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

8/23/16

56

## 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>)
```

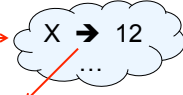
Note: in fun v -> exp(v), scope of variable is only the body exp(v)

8/23/16

57

## 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?

8/23/16

58

## 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
```

8/23/16

59

## 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?

8/23/16

60

## Values fixed at declaration time

```
# let x = 7;; (* New declaration, not an update *)  
val x : int = 7  
# plus_x 3;;
```

*(Note: In the original image, 'plus\_x 3;;' is circled in red, and a cloud contains 'x → 7'. A smaller cloud contains 'x → 12'.)*

What is the result this time?

8/23/16

61

## Values fixed at declaration time

```
# let x = 7;; (* New declaration, not an update *)  
val x : int = 7  
  
# plus_x 3;;  
- : int = 15
```

8/23/16

62

## Question

- Observation: Functions are first-class values in this language
- Question: What value does the environment record for a function variable?
- Answer: a closure

8/23/16

63

## 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:  
$$f \rightarrow \langle (v_1, \dots, v_n) \rightarrow \text{exp}, \rho_f \rangle$$
- Where  $\rho_f$  is the environment in effect when  $f$  is defined (if  $f$  is a simple function)

8/23/16

64

## Closure for plus\_x

- When plus\_x was defined, had environment:  
$$\rho_{\text{plus\_x}} = \{\dots, x \rightarrow 12, \dots\}$$
- Recall: `let plus_x y = y + x`  
is really `let plus_x = fun y -> y + x`
- Closure for `fun y -> y + x`:  
$$\langle y \rightarrow y + x, \rho_{\text{plus\_x}} \rangle$$
- Environment just after plus\_x defined:  
$$\{\text{plus\_x} \rightarrow \langle y \rightarrow y + x, \rho_{\text{plus\_x}} \rangle\} + \rho_{\text{plus\_x}}$$

8/23/16

65

Now it's your turn

You should be able to do HW1  
Problem 1 , parts (\* 7 \*) and (\* 8 \*)

8/23/16

66

## Evaluation of Application of plus\_x;;

- Have environment:  
 $\rho = \{\text{plus\_x} \rightarrow \langle y \rightarrow y + x, \rho_{\text{plus\_x}} \rangle, \dots, y \rightarrow 3, \dots\}$   
where  $\rho_{\text{plus\_x}} = \{x \rightarrow 12, \dots, y \rightarrow 24, \dots\}$
- Eval (plus\_x y,  $\rho$ ) rewrites to
- Eval (App  $\langle y \rightarrow y + x, \rho_{\text{plus\_x}} \rangle 3, \rho$ ) rewrites to
- Eval ( $y + x, \{y \rightarrow 3\} + \rho_{\text{plus\_x}}$ ) rewrites to
- Eval ( $3 + 12, \rho_{\text{plus\_x}}$ ) = 15

8/23/16

67

## 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

8/23/16

68

## 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
```

8/23/16

69

## 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!"
```

8/23/16

70

## 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")
```

8/23/16

71

## 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

8/23/16

72



## Closure for plus\_pair

---

- Assume  $\rho_{\text{plus\_pair}}$  was the environment just before `plus_pair` defined

- Closure for `plus_pair`:

$\langle (n,m) \rightarrow n + m, \rho_{\text{plus\_pair}} \rangle$

- Environment just after `plus_pair` defined:

$\{\text{plus\_pair} \rightarrow \langle (n,m) \rightarrow n + m, \rho_{\text{plus\_pair}} \rangle\}$   
+  $\rho_{\text{plus\_pair}}$