

Programming Languages and Compilers (CS 421)

Elsa L Gunter
2112 SC, UIUC



<https://courses.engr.illinois.edu/cs421/fa2017/CS421D>

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

9/12/17

1

Contact Information - Elsa L Gunter

- Office: 2112 SC
- Office hours:
 - Thursday 9:00am – 9:50pm
 - Friday 1:00pm – 1:50pm
 - Also by appointment
- Email: egunter@illinois.edu

9/12/17

2

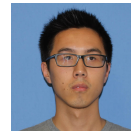
Overflow Section

- If you are not registered and want to be, fill out the form at
- <http://go.cs.illinois.edu/CS421Overflow>

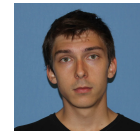
9/12/17

3

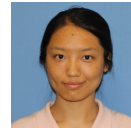
Course TAs



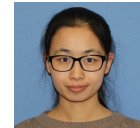
Taiyu Dong



Dan Plyukhin



Jing Huang



Fan Yang

9/12/17

4

Contact Information - TAs

- Teaching Assistants Office: 0207 SC
- Taiyu Dong
 - Email: tdong7@illinois.edu
 - Hours: Mon 9:00am – 9:50am
Thu 2:00pm – 2:50pm
- Jing Huang
 - Email: jhuang81@illinois.edu
 - Hours: Tue 6:00pm – 7:40pm

9/12/17

5

Contact Information – TAs cont

- Dan Plyukhin
 - Email: daniilp2@illinois.edu
 - Hours: Mon, Wed 12:30pm – 1:20pm
- Fan Yan
 - Email: fanyang6@illinois.edu
 - Hours: Wed, Fri 9:00am – 9:50am

9/12/17

6

Course Website

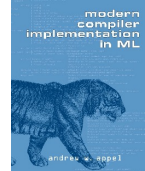
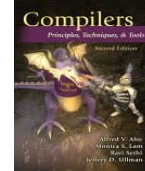
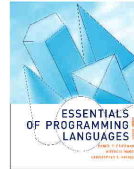
- <https://courses.engr.illinois.edu/cs421/fa2017/CS421D>
- Main page - summary of news items
- Policy - rules governing course
- Lectures - syllabus and slides
- MPs - information about assignments
- Exams
- Unit Projects - for 4 credit students
- Resources - tools and helpful info
- FAQ

9/12/17

7

Some Course References

- No required textbook
- Some suggested references



9/12/17

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

9/12/17

9

Course Grading

- Assignments 20%
 - About 12 Web Assignments (WA) (~7%)
 - About 6 MPs (in Ocaml) (~7%)
 - About 5 Labs (~6%)
 - All WAs and MPs Submitted by **PrairieLearn**
 - Late submission penalty: 20%
 - Labs in Computer-Based Testing Center (Grainger)
 - Self-scheduled over a three day period
 - No extensions beyond the three day period
 - Fall back: Labs become MPs

9/12/17

10

Course Grading

- 2 Midterms - 20% each
 - Labs in Computer-Based Testing Center (Grainger)
 - Self-scheduled over a three day period
 - No extensions beyond the three day period
 - Fall back: In class backup dates – **Oct 12, Nov 16**
 - **DO NOT MISS EXAM DATES!**
- Final 40% - Dec 19, 8:00am – 11:00am
- Mayuse of CBTC for Final
- Percentages are approximate

9/12/17

11

Course Assingments – WA & MP

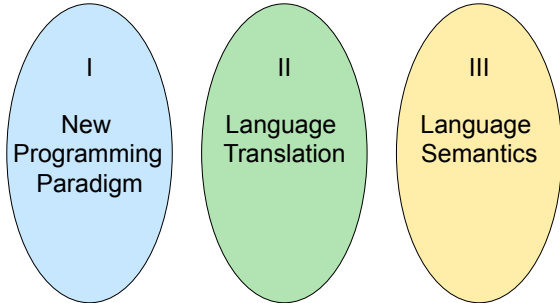
- You may discuss assignments and their solutions with others
- You may work in groups, but you must **list members with whom you worked** if you share solutions or solution outlines
- **Each student must write up and turn in their own solution separately**
- You may look at examples from class and other 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

9/12/17

12

Programming Languages & Compilers

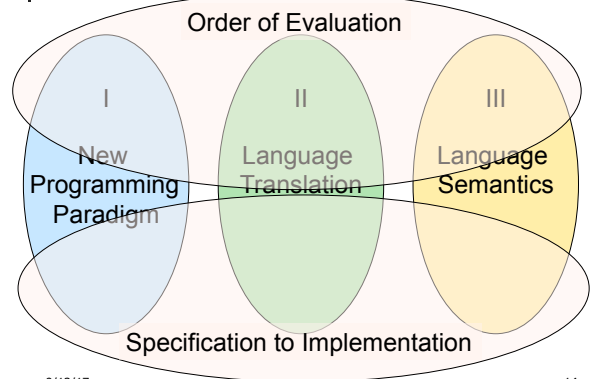
Three Main Topics of the Course



9/12/17

13

Programming Languages & Compilers

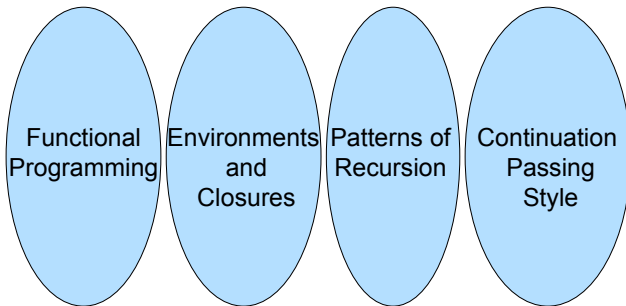


9/12/17

14

Programming Languages & Compilers

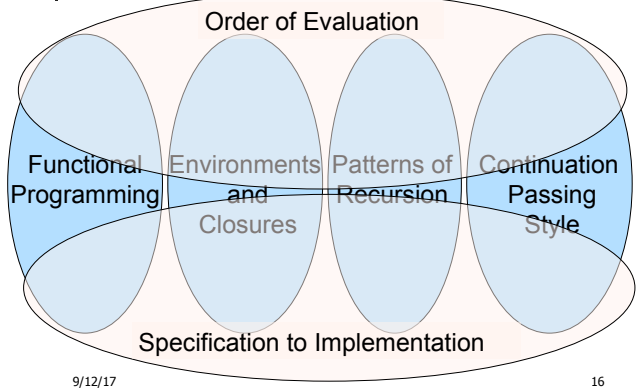
I : New Programming Paradigm



9/12/17

15

Programming Languages & Compilers

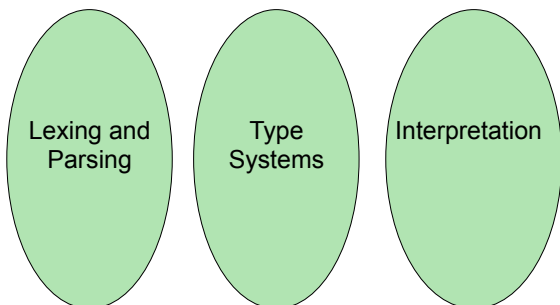


9/12/17

16

Programming Languages & Compilers

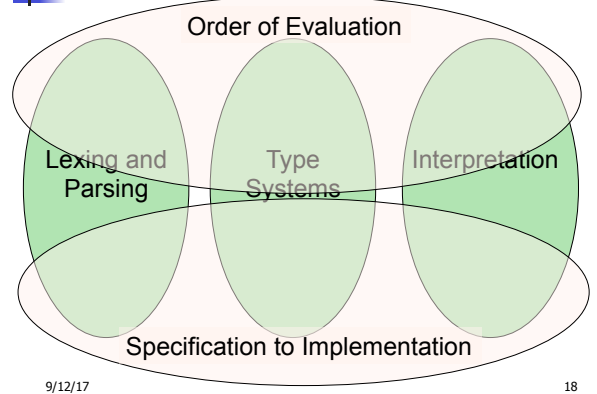
II : Language Translation



9/12/17

17

Programming Languages & Compilers

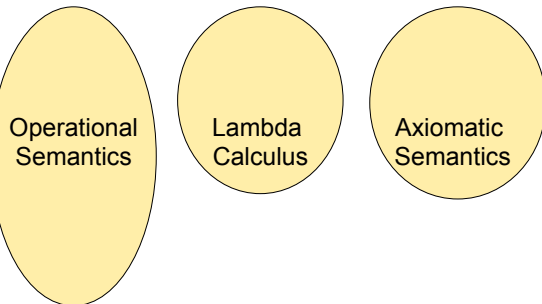


9/12/17

18

Programming Languages & Compilers

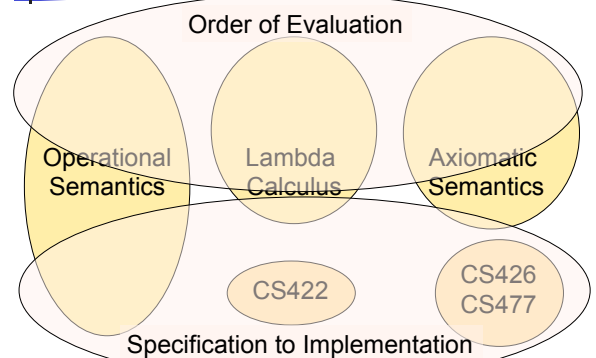
III : Language Semantics



9/12/17

19

Programming Languages & Compilers



9/12/17

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

9/12/17

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>

9/12/17

22

References for OCaml

- Supplemental texts (not required):
 - The Objective Caml system release 4.05, 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

9/12/17

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

9/12/17

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

9/12/17

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

9/12/17

26

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
```

9/12/17

27

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
```

9/12/17

28

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
```

9/12/17

29

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
```

9/12/17

30

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
```

9/12/17

31

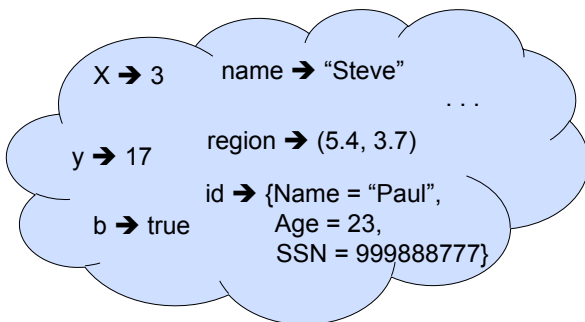
Environments

- *Environments* record what value is associated with a given identifier
- Central to the semantics and implementation of a language
- Notation
$$\rho = \{\text{name}_1 \rightarrow \text{value}_1, \text{name}_2 \rightarrow \text{value}_2, \dots\}$$
Using set notation, but describes a partial function
- Often stored as list, or stack
 - To find value start from left and take first match

9/12/17

32

Environments



9/12/17

33

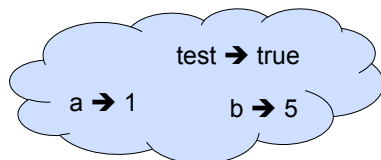
Global Variable Creation

```
# 2 + 3;; (* Expression *)
// doesn't affect the environment
# let test = 3 < 2;; (* Declaration *)
val test : bool = false
// ρ1 = {test → false}
# let a = 1 let b = a + 4;; (* Seq of dec *)
// ρ2 = {b → 5, a → 1, test → false}
```

9/12/17

34

Environments



9/12/17

35

New Bindings Hide Old

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

- What is the environment after this declaration?

9/12/17

36

New Bindings Hide Old

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

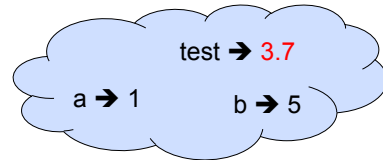
- What is the environment after this declaration?

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

9/12/17

37

Environments



9/12/17

38

Now it's your turn

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

9/12/17

39

Local Variable Creation

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

```
# let b = 5 * 4
```

```
// ρ4 = {b → 20, test → 3.7, a → 1}
```

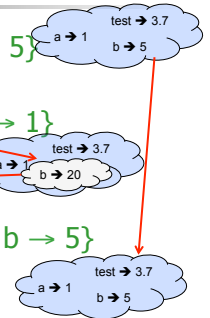
```
in 2 * b;;
```

```
- : int = 40
```

```
// ρ5 = ρ3 = {test → 3.7, a → 1, b → 5}
```

```
# b;;
```

```
- : int = 5
```



9/12/17

40

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}
```

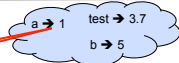
```
in b * b;;
```

```
val c : int = 4
```

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

```
# b;;
```

```
- : int = 5
```



9/12/17

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
```



9/12/17

42

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
```

9/12/17

43

Now it's your turn

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

9/12/17

44

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
```

9/12/17

45

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
```

9/12/17

46

Now it's your turn

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

9/12/17

47

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)

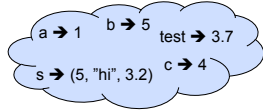
// ρ8 = {s → (5, "hi", 3.2),
          c → 4, test → 3.7,
          a → 1, b → 5}
```

9/12/17

48

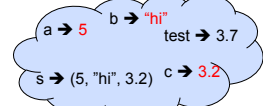
Pattern Matching with Tuples

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



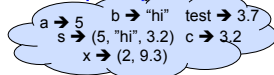
```
# let (a,b,c) = s;; (* (a,b,c) is a pattern *)
```

```
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)
```



9/12/17

49

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"
```

9/12/17

50

Now it's your turn

You should be able to do WA1
Problem 1 , part (* 6 *)

9/12/17

51

Functions

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

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

```
# plus_two 17;;
```

```
- : int = 19
```

9/12/17

52

Functions

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

```
plus_two 17;;
```

```
- : int = 19
```

9/12/17

53

Nameless Functions (aka Lambda Terms)

```
fun n -> n + 2;;
```

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

```
- : int = 19
```



9/12/17

54

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

9/12/17

55

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>)
```

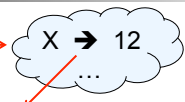
Note: in fun v -> exp(v), scope of variable is only the body exp(v)

9/12/17

56

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?

9/12/17

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;;  
- : int = 15
```

9/12/17

58

Values fixed at declaration time

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

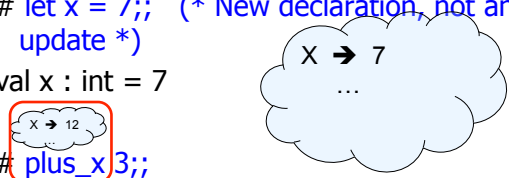
What is the result this time?

9/12/17

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?

9/12/17

60

Values fixed at declaration time

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

# plus_x 3;;
- : int = 15
```

9/12/17

61

Question

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

9/12/17

62

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 ρ_f is the environment in effect when f is defined (if f is a simple function)

9/12/17

63

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:
 $\{\text{plus_x} \rightarrow \langle y \rightarrow y + x, \rho_{\text{plus_x}} \rangle\} + \rho_{\text{plus_x}}$

9/12/17

64

Now it's your turn

You should be able to do WA1
Problem 1 , parts (* 7 *) and (* 8 *)

9/12/17

65

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, ρ)` rewrites to
- `App (Eval(plus_x, ρ), Eval(y, ρ))` rewrites to
- `App ($\langle y \rightarrow y + x, \rho_{\text{plus_x}} \rangle$, 3)` rewrites to
- `Eval (y + x, $\{y \rightarrow 3\} + \rho_{\text{plus_x}}$)` rewrites to
- `Eval (3 + 12, $\rho_{\text{plus_x}}$) = 15`

9/12/17

66

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

9/12/17

67

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
```

9/12/17

68

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!"
```

9/12/17

69

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")
```

9/12/17

70

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

9/12/17

71

Closure for plus_pair

- Assume ρ_{plus_pair} was the environment just before `plus_pair` defined
- Closure for `plus_pair`:
 $\langle (n,m) \rightarrow n + m, \rho_{plus_pair} \rangle$
- Environment just after `plus_pair` defined:
 $\{ plus_pair \rightarrow \langle (n,m) \rightarrow n + m, \rho_{plus_pair} \rangle \}$
+ ρ_{plus_pair}

9/12/17

72