CSE216
Programming Abstraction

Instructor: Zhoulai Fu

State University of New York, Korea

Today

- Review: Capture avoiding substitution in beta reduction
- Extending lambda calculus core
- A bit of OCaml

Review exercises alpha-renaming to avoid capture

Reduce each of the lambda terms below until it becomes a normal term

- (λ x .λ y. x y) y
- (λ x. λ y. λ f. f x y) (f x) (g y)

Solution

- (λ x .λ y. x y) y ->beta λ y. x y [x:=y] ->_alpha λ p. x p [x:=y] = λ p. y p

- (λ x. λ y. λ f. f x y) (f x) (g y) ->beta (λ y. λ f. f x y) (x := f x) (g y) ->alpha (λ y. λ p. p x y) [x := f x] (g y) = (λ y. λ p. p (f x) y) (g y) -> (λ p. p (f x) y) [y:= g y] = λ p. p (f x) (g y)

Extending lambda calculus core

“extending lambda calculus core”?

- The core has only three constructs
- It does not have numbers, conditions, logic, loops
- These things can all be encoded by the core
- We will write **||<languaage syntax>||=<lambda-term>** for the encoding
- "*The integers were created by God, everything else is the work of man”*

TRUE, FALSE, and IF

- IF c e1 e2 returns e1 if c is TRUE, or e2 if c is FALSE
- So we encode TRUE by λ x. λ y. x
- We encode FALSE by λ x. λ y. y
- We encode IF by λ c. λ x. λ y. c x y

- Examples
  - if true then b else c = (λx.λy.x) b c → (λy.b) c → b
  - if false then b else c = (λx.λy.y) b c → (λy.y) c → c

Logic AND

- ||AND x y|| = ||IF x y FALSE|| = x y FALSE = x y x
- Thus ||AND|| = λ x. λ y. x y x
- Exercise: ||OR||=?
- Exercise: ||NOT||=?

Logic OR

- Exercise: $||OR||=?$

Logic NOT

- Exercise: $\|NOT\|=?$

Numbers

- Any counting system that makes sense would work
- We want n f x = f(f(f(f(… f(x)))))
- Since 0 f x = x, we have $||0|| = \lambda f.\ \lambda x.\ x$
- Since 1 f x = f x, we have $||1|| = \lambda f.\ \lambda x.\ f\ x$
- … (called Church numerals)

Exercise

- Write lambda calculus to encode SUM
- Think what would be SUM m n

Exercise

- PROD

An important thing in lambda calculus which we will go over quickly: Recursion

Implementing recursion

To give us access to a function itself, we pass it in as another parameter.

$$FACT = (\lambda f.\lambda n.\text{if } (= n\ 0)\ 1\ (*\ n\ (f\ f\ (-\ n\ 1))))$$

(FACT is just shorthand for that string of characters.)

Now if we write

FACT FACT 5

This will work, because the β-reduction substitutes FACT for f, resulting in a function call FACT FACT 4. Etc.

Exercise

- Beta-reduce FACT FACT 3

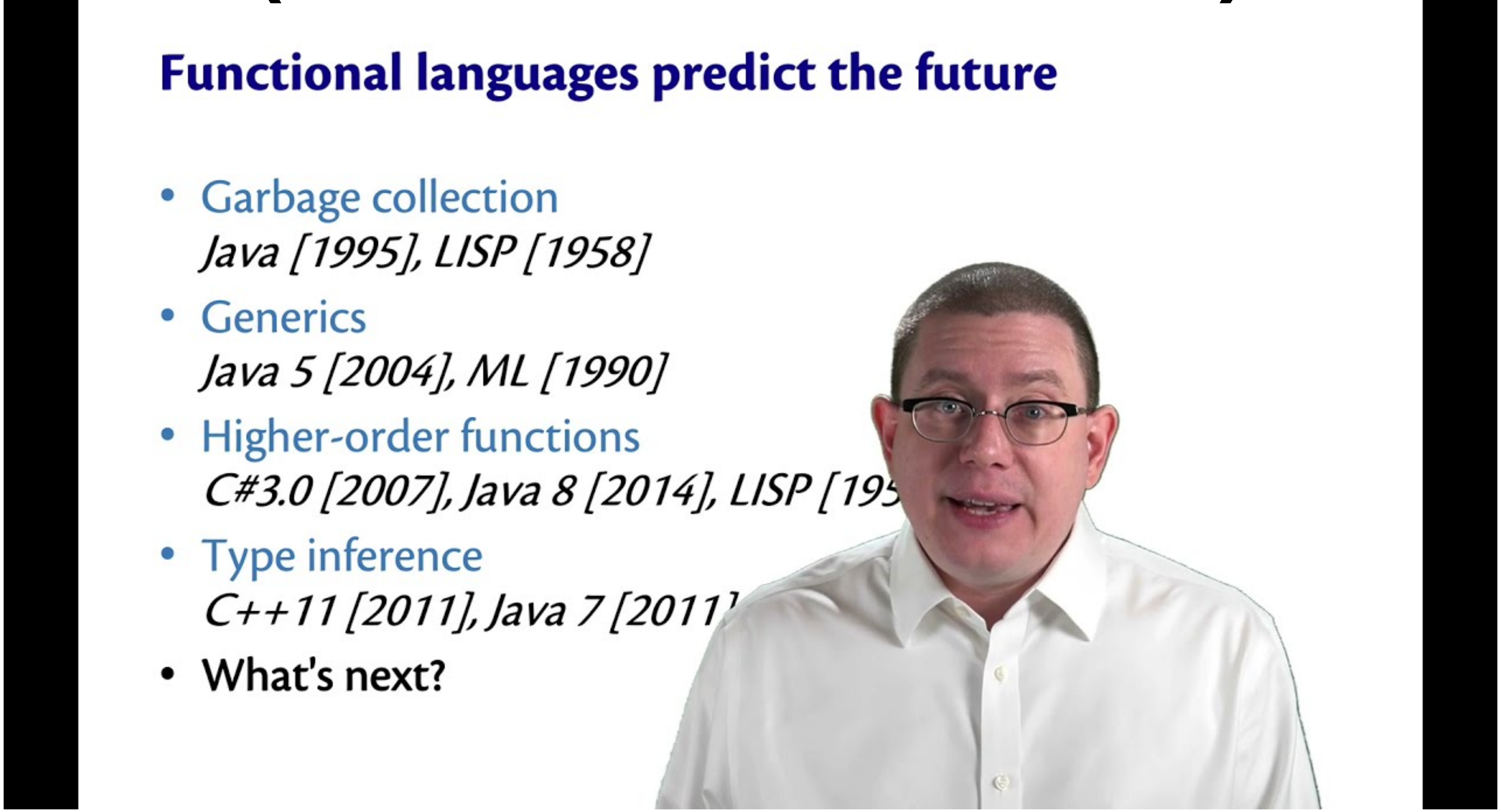
FACT = (λf.λn.if (= n 0) 1 (* n (f f (- n 1))))

Why Harvard, MIT, Stanford, Cambridge all teach functional programming

https://youtu.be/6APBx0WsgeQ

Things to do before next lecture

Another good explanation (to watch at home)



https://youtu.be/SKr3ItChPSI

Install Ocaml

- Official guide: https://ocaml.org/docs/up-and-running
- SBU Guide
  - https://sites.google.com/cs.stonybrook.edu/cse216/lectures?authuser=0 (may need SBU credentials)
- Once installed, get into the toplevel by running "ocaml". In the toplevel, run: *print_endline "hello"*
- If nothing works, use TryOcaml for now: https://try.ocamlpro.com/.

Toplevel Demo

```
# 42;;
- : int = 42
# let f x y = x + y;;
val f : int -> int -> int = <fun>
# f 3 ;;
- : int -> int = <fun>
# f 3 4 ;;
- : int = 7
# #use "hello.ml";;
hello world!
- : unit = ()
```

Thing to do summary

- Watch the video
- Install OCaml
- Run hello world code on Ocaml Toplevel, and play around