CSE216
Programming Abstraction

Instructor: Zhoulai Fu

State University of New York, Korea

Many slides adapted from CMU. Thanks!
https://www.cs.cmu.edu/~venkatg/teaching/15252-sp20/notes/lambda-calculus-slides.pdf

Today

- Lambda calculus
- Its syntax and semantics overview
- A video summary

Alonzo Church's model of computing, **Lambda Calculus**.

Alonzo Church (1903–1995)

History

What is a computation/ algorithm?

Hilbert's 10th problem (1900):

Given a multivariate polynomial w/ integer coeffs,
e.g. $4x^2y^3 - 2x^4z^5 + x^8$,
"*devise a process according to which it can be determined in a finite number of operations*"
whether it has an integer root.

Gödel (1934):

Discusses some ideas for definitions of what functions/languages are "computable" but isn't confident what's a good definition.

Church (1936):

Invents lambda calculus, claims it should be the definition.

Meanwhile... a certain British grad student in Princeton, unaware of all these debates...

Turing

Syntax and semantics of lambda calculus

Lambda calculus Core

- TERM::= Var //Variables

  | lambda Var. TERM // Definition/Abstraction

  | TERM TERM // Application

  Var ::= x | y | z …

- λ x. λ y. xy

Temparily Introducing functions and constants

- We will start out with a version of the lambda calculus that has constants like 0, 1, 2… and functions such as + * =.
- We also include constants TRUE and FALSE, and logical functions AND, OR, NOT.

Syntax summary

<exp> ::= <constant>	Built-in constants & functions
| <variable>	Variable names, e.g. x, y, z…
| <exp> <exp>	Application
| λ <variable> . <exp>	Lambda Abstractions
| (<exp>)	Parens

Example: λf.λn. IF (= n 0) 1 (* n (f (- n 1)))

Example

A way to write expressions that denote functions.

Example: $(\lambda x \,.\, + \; x \; 1)$

Example

A way to write expressions that denote functions.

Example: $(\lambda x\ .\ + x\ 1)$

λ means here comes a function

Then comes the parameter x

Then comes the .

Then comes the body + x 1

The function is ended by the) (or end of string)

Example

A way to write expressions that denote functions.

Example: $(\lambda x \,.\, + \; x \; 1)$

λ means here comes a function

Then comes the parameter x

Then comes the .

Then comes the body + x 1

The function is ended by the) (or end of string)

$(\lambda x \,.\, + \; x \; 1)$ is the increment function.

$$(\lambda x \,.\, + \; x \; 1) \; 5 \rightarrow 6$$

(We'll explain this more thoroughly later.)

Lambda calculus Semantics

- **evaluating** the expression
  - Beta-reduction

“evaluating”

For example: (+ 4 5)

+ is a function. We write functions in prefix form.

Another example: (+ (* 5 6) (* 8 3))

“evaluating”?

For example: (+ 4 5)

+ is a function. We write functions in prefix form.

Another example: (+ (* 5 6) (* 8 3))

Evaluation proceeds by choosing a reducible expression and reducing it. (There may be more than one order.)

(+ (* 5 6) (* 8 3)) → (+ 30 (* 8 3)) → (+ 30 24) → 54

Function application is indicated by juxtaposition: f x

"The function f applied to the argument x"

What if we want a function of more than one argument? We could invent a notation f(x,y), but there's an alternative. To express the sum of 3 and 4 we write:

((+ 3) 4)

The expression (+ 3) denotes the function that adds 3 to its argument.

Function application is indicated by juxtaposition: $f\ x$

"The function f applied to the argument x"

Function application is indicated by juxtaposition: f x

"The function f applied to the argument x"

What if we want a function of more than one argument? We could invent a notation f(x,y), but there's an alternative. To express the sum of 3 and 4 we write:

((+ 3) 4)

Function application is indicated by juxtaposition: f x

"The function f applied to the argument x"

What if we want a function of more than one argument? We could invent a notation f(x,y), but there's an alternative. To express the sum of 3 and 4 we write:

((+ 3) 4)

The expression (+ 3) denotes the function that adds 3 to its argument.

So all functions take one argument. So when we wrote (+ 3 4) this is shorthand for ((+ 3) 4). (The arguments are associated to the left.)

So all functions take one argument. So when we wrote (+ 3 4) this is shorthand for ((+ 3) 4). (The arguments are associated to the left.)

This mechanism is known as currying (in honor of Haskell Curry).

So all functions take one argument. So when we wrote (+ 3 4) this is shorthand for ((+ 3) 4). (The arguments are associated to the left.)

This mechanism is known as currying (in honor of Haskell Curry).

Parentheses can be added for clarity. E g:

((f ((+ 4) 3)) (g x))   is identical to f (+ 4 3) (g x)

Exercise 1

Lambda calculus is the base of any and all functional languages. A major aspect of any and all functional languages is the use of recursion. Some problems in computer science are primarily, or can only be thought of in recursive terms, such as traversing a tree. In other cases, you can take an iterative problem and turn it into a recursive problem. Think of the idea of multiplication between positive numbers as repeatedly adding a number to itself n times, e.g.

```
int mult (x, y) {
    int temp = x;
    for (int i = 1; i<y; i++) {
        temp+=x;
    }
    return temp;
}
```

Now try to do this recursively. Feel free to add as many helper functions as you want, but you must only use addition and subtraction to get your goal. No multiplication must be used, and remember: no need to worry about negative numbers.

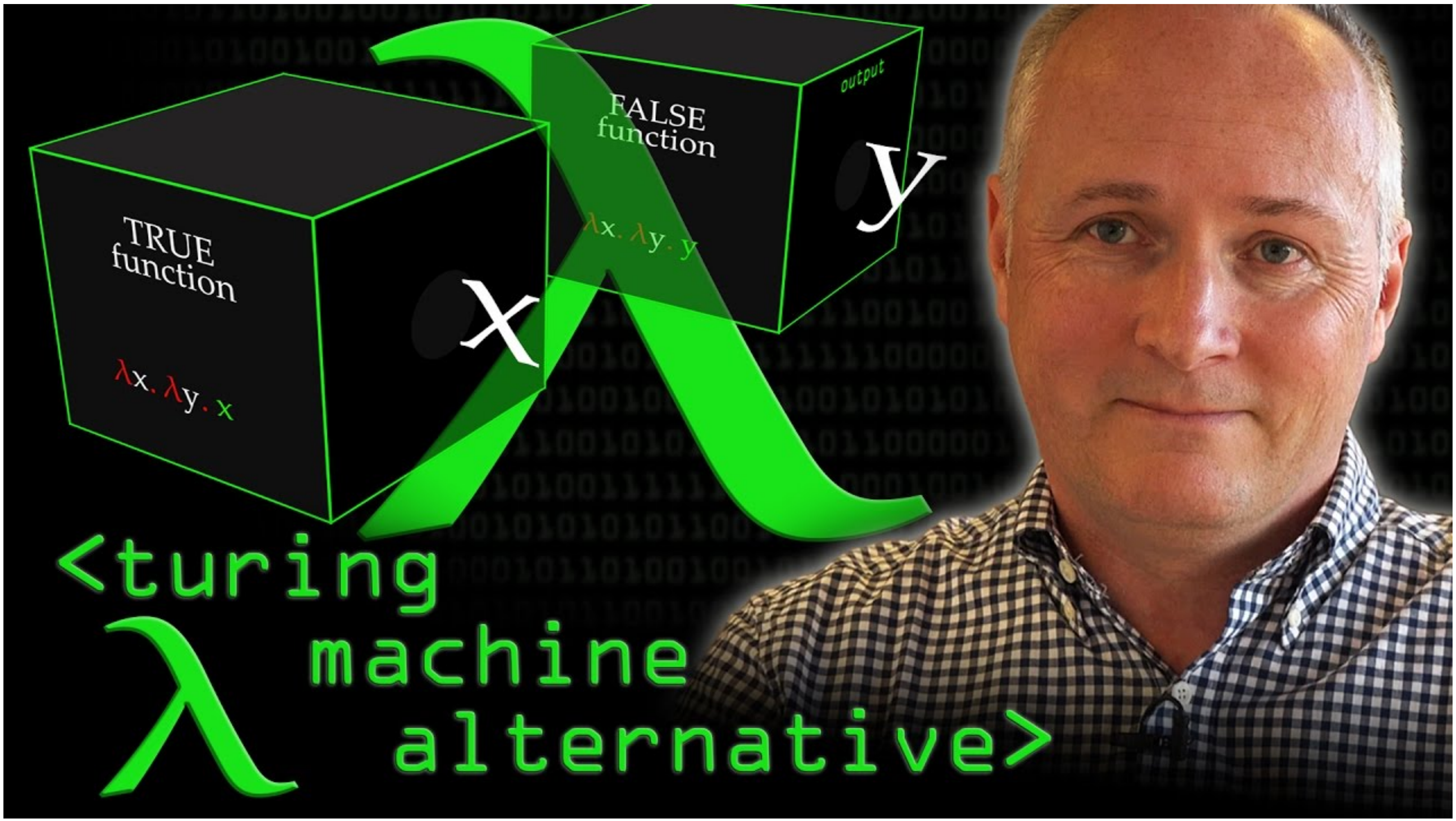
Exercise 2

- Evaluate the following lambda expressions until they are irreducible. (we have not learned this in detail)

- $(\lambda\ x.\lambda y.x)\ x\ y$

- $(\lambda x.xx)\ (\lambda x.xx)$

Summary (first 7 min)



https://youtu.be/eis11j_iGMs