The Lambda Calculus

Ronghui Gu

Spring 2024

Columbia University

* Course website: https://verigu.github.io/4115Spring2024/

What is the lambda calculus?

The lambda calculus can be called the smallest universal programming language of the world (by Alonzo Church, 1930s).

What is the lambda calculus?

The lambda calculus can be called the smallest universal programming language of the world (by Alonzo Church, 1930s).

- syntax: a single function definition scheme

What is the lambda calculus?

The lambda calculus can be called the smallest universal programming language of the world (by Alonzo Church, 1930s).

- syntax: a single function definition scheme
- semantics: a single transformation rule (variable substitution)

What is the lambda calculus?

The lambda calculus can be called the smallest universal programming language of the world (by Alonzo Church, 1930s).

- syntax: a single function definition scheme
- semantics: a single transformation rule (variable substitution)
- universal: any computable function can be expressed and evaluated using this formalism.

Lambda Expressions

Function application written in prefix form. "Add x and five"

$$(+\ x\ 5)$$

Lambda Expressions

Function application written in prefix form. "Add x and five"

$$(+\ x\ 5)$$

Evaluation: select a redex and evaluate it:

$$\begin{aligned}(+\ (*\ 5\ 6)\ (*\ 8\ 3)) &\rightarrow (+\ 30\ (*\ 8\ 3)) \\ &\rightarrow (+\ 30\ 24) \\ &\rightarrow 54\end{aligned}$$

Lambda Expressions

Function application written in prefix form. “Add x and five”

$$(+\ x\ 5)$$

Evaluation: select a redex and evaluate it:

$$\begin{aligned}(+\ (*\ 5\ 6)\ (*\ 8\ 3)) &\rightarrow (+\ 30\ (*\ 8\ 3))\\ &\rightarrow (+\ 30\ 24)\\ &\rightarrow 54\end{aligned}$$

Often more than one way to proceed:

$$\begin{aligned}(+\ (*\ 5\ 6)\ (*\ 8\ 3)) &\rightarrow (+\ (*\ 5\ 6)\ 24)\\ &\rightarrow (+\ 30\ 24)\\ &\rightarrow 54\end{aligned}$$

Lambda Abstraction

The only other thing in the lambda calculus is lambda abstraction: a notation for defining unnamed functions.

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

$$\begin{array}{ccccccc} (& \lambda & x & . & + & x & 1 \ ) \\ & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow \end{array}$$

The function of x that adds x to 1

Lambda Abstraction

The only other thing in the lambda calculus is lambda abstraction: a notation for defining unnamed functions.

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

$$\begin{array}{cccccccc} (& \lambda & x & . & + & x & 1 &) \\ & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \end{array}$$

The function of x *that adds* x *to* 1

Replace the λ with **fun** and the dot with an arrow to get a lambda expression in Ocaml:

```
fun x -> (+) x 1
```

Evaluating Lambda Abstraction

Evaluation of a lambda abstraction—beta-reduction—is just substitution:

$$\begin{aligned}(\lambda x. + \; x \; 1) \; 4 &\to (+ \; 4 \; 1) \\ &\to 5\end{aligned}$$

Evaluating Lambda Abstraction

Evaluation of a lambda abstraction—beta-reduction—is just substitution:

$$\begin{aligned}(\lambda x.+\ x\ 1)\ 4 &\rightarrow (+\ 4\ 1)\\ &\rightarrow 5\end{aligned}$$

The argument may appear more than once

$$\begin{aligned}(\lambda x.+\ x\ x)\ 4 &\rightarrow (+\ 4\ 4)\\ &\rightarrow 8\end{aligned}$$

Evaluating Lambda Abstraction

Evaluation of a lambda abstraction—beta-reduction—is just substitution:

$$\begin{aligned}(\lambda x.+\ x\ 1)\ 4 &\rightarrow (+\ 4\ 1)\\ &\rightarrow 5\end{aligned}$$

The argument may appear more than once

$$\begin{aligned}(\lambda x.+\ x\ x)\ 4 &\rightarrow (+\ 4\ 4)\\ &\rightarrow 8\end{aligned}$$

or not at all

$$(\lambda x.\ 3)\ 5 \rightarrow 3$$

The Syntax of the Lambda Calculus

$$
\begin{array}{lcl}
expr & ::= & expr\ expr \\
 & | & \lambda\ variable\ .\ expr \\
 & | & variable \\
 & | & (expr)
\end{array}
$$

The Syntax of the Lambda Calculus

$$
\begin{array}{lcl}
\textit{expr} & ::= & \textit{expr expr} \\
 & | & \lambda\ \textit{variable}\ .\ \textit{expr} \\
 & | & \textit{variable} \\
 & | & (\textit{expr})
\end{array}
$$

Function application binds more tightly than λ:

$$\lambda x.f\ g\ x = \lambda x.\big((f\ g)\ x\big)$$

First-Class Functions

Functions may be arguments (first-class functions)

$$\begin{aligned}(\lambda f.\ f\ 3)(\lambda x.+\ x\ 1) &\to (\lambda x.+\ x\ 1)\ 3\\ &\to (+\ 3\ 1)\\ &\to 4\end{aligned}$$

Free and Bound Variables

$$(\lambda x.+\ x\ y)\ 4$$

Here, x is like a function argument but y is like a global variable.

Free and Bound Variables

$$(\lambda x. + \; x \; y) \; 4$$

Here, x is like a function argument but y is like a global variable.

Technically, x occurs bound and y occurs free in

$$(\lambda x. + \; x \; y)$$

However, both x and y occur free in

$$(+ \; x \; y)$$

Beta-Reduction More Formally

$$(\lambda x.E)\ F \rightarrow_\beta E'$$

where E' is obtained from E by replacing every instance of x that appears free in E with F.

Beta-Reduction More Formally

$$(\lambda x.E)\ F \rightarrow_\beta E'$$

where E' is obtained from E by replacing every instance of x that appears free in E with F.

The definition of free and bound mean variables have scopes. Only the rightmost x appears free in

$$(\lambda x. +\ (-\ x\ 1))\ x\ 3$$

so

$$\begin{aligned}(\lambda x.(\lambda x. +\ (-\ x\ 1))\ x\ 3)\ 9 &\rightarrow (\lambda x. +\ (-\ x\ 1))\ 9\ 3\\ &\rightarrow +\ (-\ 9\ 1)\ 3\\ &\rightarrow +\ 8\ 3\\ &\rightarrow 11\end{aligned}$$

Another Example

$$\Big(\lambda x.\lambda y. +\ x\ \big((\lambda x. -\ x\ 3)\ y\big)\Big)\ 5\ 6$$

Another Example

$$\Big(\lambda x.\lambda y.+\ x\ ((\lambda x.-\ x\ 3)\ y)\Big)\ 5\ 6$$
$$\rightarrow\Big(\lambda y.+\ 5((\lambda x.-\ x\ 3)\ y)\Big)\ 6$$

Another Example

$$
\begin{aligned}
&\Big(\lambda x.\lambda y.+\ x\ \big((\lambda x.-\ x\ 3)\ y\big)\Big)\ 5\ 6\\
\rightarrow&\Big(\lambda y.+\ 5\big((\lambda x.-\ x\ 3)\ y\big)\Big)\ 6\\
\rightarrow&+5\big((\lambda x.-\ x\ \ 3)\ 6\big)\\
\rightarrow&+5\ (-\ 6\ 3)\\
\rightarrow&+5\ 3\\
\rightarrow&\ 8
\end{aligned}
$$

Alpha-Conversion

One way to confuse yourself less is to do α-conversion: renaming a λ argument and its bound variables. Formal parameters are only names: they are correct if they are consistent.

$$
\begin{aligned}
(\lambda x.(\lambda x.+\ (-\ x\ 1))\ x\ 3)\ 9 &\leftrightarrow (\lambda x.(\lambda y.+\ (-\ y\ 1))\ x\ 3)\ 9 \\
&\rightarrow ((\lambda y.+\ (-\ y\ 1))\ 9\ 3) \\
&\rightarrow (+\ (-\ 9\ 1)\ 3) \\
&\rightarrow (+\ 8\ 3) \rightarrow 11
\end{aligned}
$$

Reduction Order

The order in which you reduce things can matter.

$$(\lambda x.\lambda y.y)\ \big((\lambda z.z\ z)\ (\lambda z.z\ z)\big)$$

Two things can be reduced:

$$(\lambda z.z\ z)\ (\lambda z.z\ z)$$

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

Reduction Order

The order in which you reduce things can matter.

$$(\lambda x.\lambda y.y)\ \big((\lambda z.z\ z)\ (\lambda z.z\ z)\big)$$

Two things can be reduced:

$$(\lambda z.z\ z)\ (\lambda z.z\ z)$$

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

However,

$$(\lambda z.z\ z)\ (\lambda z.z\ z) \rightarrow (\lambda z.z\ z)\ (\lambda z.z\ z)$$

$$(\lambda x.\lambda y.y)(\cdots) \rightarrow (\lambda y.y)$$

Normal Form

A lambda expression that cannot be β-reduced is in normal form. Thus,

$$\lambda y.y$$

is the normal form of

$$(\lambda x.\lambda y.y)\ \big((\lambda z.z\ z)\ (\lambda z.z\ z)\big)$$

Normal Form

A lambda expression that cannot be β-reduced is in normal form. Thus,

$$\lambda y.y$$

is the normal form of

$$(\lambda x.\lambda y.y)\ \big((\lambda z.z\ z)\ (\lambda z.z\ z)\big)$$

Not everything has a normal form. E.g.,

$$(\lambda z.z\ z)\ (\lambda z.z\ z)$$

can only be reduced to itself, so it never produces an non-reducible expression.

Normal Form

Can a lambda expression have more than one normal form?

Normal Form

Can a lambda expression have more than one normal form?

Church-Rosser Theorem I Corollary: No expression may have two distinct normal forms.

Normal-Order Reduction

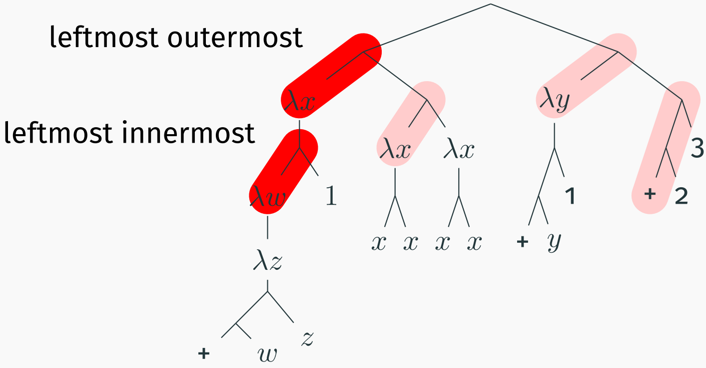
Not all expressions have normal forms, but is there a reliable way to find the normal form if it exists?

Church-Rosser Theorem II: If $E_1 \rightarrow E_2$ and E_2 is in normal form, then there exists a *normal order* reduction sequence from E_1 to E_2.

Normal order reduction: reduce the leftmost outermost redex.

Normal-Order Reduction

$$\Big(\big(\lambda x.((\lambda w.\lambda z. + \ w\ z)\ 1)\big)\big((\lambda x.x\ x)(\lambda x.x\ x)\big)\Big)\big((\lambda y. + \ y\ 1)(+\ 2\ 3)\big)$$



Boolean Logic in the Lambda Calculus

"Church Booleans"

$$\begin{aligned} \text{true} &= \lambda x.\lambda y.x \\ \text{false} &= \lambda x.\lambda y.y \end{aligned}$$

Each is a function of two arguments: true is select first; false is select second.

Boolean Logic in the Lambda Calculus

"Church Booleans"

$$\text{true} = \lambda x.\lambda y.x$$
$$\text{false} = \lambda x.\lambda y.y$$

Each is a function of two arguments: true is select first; false is select second. If-then-else uses its predicate to select *then* or *else*:

$$\text{ifelse} = \lambda p.\lambda a.\lambda b.\ p\ a\ b$$

Boolean Logic in the Lambda Calculus

"Church Booleans"

$$\begin{aligned} \text{true} &= \lambda x.\lambda y.x \\ \text{false} &= \lambda x.\lambda y.y \end{aligned}$$

Each is a function of two arguments: true is select first; false is select second. If-then-else uses its predicate to select *then* or *else*:

$$\text{ifelse} = \lambda p.\lambda a.\lambda b.\ p\ a\ b$$

E.g.,

$$\begin{aligned} \text{ifelse true } 42\ 58 &= \text{true } 42\ 58 \\ &\to (\lambda x.\lambda y.\ x)\ 42\ 58 \\ &\to (\lambda y.42)\ 58 \to 42 \end{aligned}$$

Boolean Logic in the Lambda Calculus

Logic operators? and $p\ q$

Boolean Logic in the Lambda Calculus

Logic operators? and $p\ q$

and $p\ q = \ p\ q\ p$

Boolean Logic in the Lambda Calculus

Logic operators? and $p\ q$

$$\text{and}\ p\ q = \ p\ q\ p$$

$$\text{and} = \lambda p.\lambda q.\ p\ q\ p$$

Boolean Logic in the Lambda Calculus

Logic operators? and $p\ q$

$$\text{and}\ p\ q = \ p\ q\ p$$

$$\text{and} = \lambda p.\lambda q.\ p\ q\ p$$

$$\begin{aligned} \text{and true false} &= (\lambda p.\lambda q.\ p\ q\ p)\ \text{true false} \\ &\rightarrow \text{true false true} \\ &\rightarrow (\lambda x.\lambda y.\ x)\ \text{false true} \\ &\rightarrow \text{false} \end{aligned}$$

Boolean Logic in the Lambda Calculus

Logic operators? or $p\ q$

Boolean Logic in the Lambda Calculus

Logic operators? or $p\ q$

or $p\ q = \ p\ p\ q$

Boolean Logic in the Lambda Calculus

Logic operators? or $p\ q$

$$\text{or}\ p\ q = \ p\ p\ q$$

$$\text{or} = \lambda p.\lambda q.\ p\ p\ q$$

Boolean Logic in the Lambda Calculus

Logic operators? or $p\ q$

$$\text{or}\ p\ q = \ p\ p\ q$$

$$\text{or} = \lambda p.\lambda q.\ p\ p\ q$$

$$\begin{aligned}\text{or false true} &= (\lambda p.\lambda q.\ p\ p\ q)\ \text{false true}\\ &\rightarrow\ \text{false false true}\\ &\rightarrow (\lambda x.\lambda y.\ y)\ \text{false true}\\ &\rightarrow \text{true}\end{aligned}$$

Boolean Logic in the Lambda Calculus

Logic operators? (not p) a b

Boolean Logic in the Lambda Calculus

Logic operators? (not p) a b

$$\text{not } p\ a\ b = \ p\ b\ a$$

Boolean Logic in the Lambda Calculus

Logic operators? (not p) a b

$$\text{not}\ p\ a\ b = \ p\ b\ a$$

$$\text{not} = \lambda p.\lambda a.\lambda b.\ p\ b\ a$$

Boolean Logic in the Lambda Calculus

Logic operators? (not p) a b

$$\text{not}\ p\ a\ b = \ p\ b\ a$$

$$\text{not} = \lambda p.\lambda a.\lambda b.\ p\ b\ a$$

$$\begin{aligned} \text{not true} \ &= \ (\lambda p.\lambda a.\lambda b.\ p\ b\ a)\ \text{true} \\ &\rightarrow_\beta \lambda a.\lambda b.\ \text{true}\ b\ a \\ &\rightarrow_\beta \lambda a.\lambda b.\ b \\ &\rightarrow_\alpha \lambda x.\lambda y.\ y = \text{false} \end{aligned}$$

Arithmetic: The Church Numerals

$$
\begin{aligned}
0 &= \lambda f.\lambda x.x \\
1 &= \lambda f.\lambda x.fx \\
2 &= \lambda f.\lambda x.f(fx) \\
3 &= \lambda f.\lambda x.f\big(f(fx)\big)
\end{aligned}
$$

Arithmetic: The Church Numerals

$$
\begin{aligned}
0 &= \lambda f.\lambda x.x \\
1 &= \lambda f.\lambda x.fx \\
2 &= \lambda f.\lambda x.f(fx) \\
3 &= \lambda f.\lambda x.f\big(f(fx)\big)
\end{aligned}
$$

I.e., for $n = 0, 1, 2, \ldots$, n, $fx = f^{(n)}(x)$.

Arithmetic: The Church Numerals

$$
\begin{aligned}
0 &= \lambda f.\lambda x.x \\
1 &= \lambda f.\lambda x.fx \\
2 &= \lambda f.\lambda x.f(fx) \\
3 &= \lambda f.\lambda x.f\big(f(fx)\big)
\end{aligned}
$$

I.e., for $n = 0, 1, 2, \ldots$, n, $fx = f^{(n)}(x)$.The successor function:

$$\text{succ} = \lambda n.\lambda f.\lambda x.\ f\ (n\ f\ x)$$

$$
\begin{aligned}
\text{succ}\ 2 &= \big(\lambda n.\lambda f.\lambda x.\ f\ (n\ f\ x)\big)\ 2 \\
&\rightarrow \lambda f.\lambda x.\ f\ (2\ f\ x) \\
&= \lambda f.\lambda x.\ f\Big(\big(\lambda f.\lambda x.\ f\ (f\ x)\big)\ f\ x\Big) \\
&\rightarrow \lambda f.\lambda x.\ f\big(f\ (f\ x)\big) = 3
\end{aligned}
$$

Adding Church Numerals

Finally, we can add:

Adding Church Numerals

Finally, we can add:

$$\text{plus} = \lambda m.\lambda n.\lambda f.\lambda x.\ m\ f\ (n\ f\ x)$$

Not surprising since $f^{(m)} \circ f^{(n)} = f^{(m+n)}$

Adding Church Numerals

Finally, we can add:

$$\text{plus} = \lambda m.\lambda n.\lambda f.\lambda x.\ m\ f\ (n\ f\ x)$$

Not surprising since $f^{(m)} \circ f^{(n)} = f^{(m+n)}$

$$\begin{aligned}\text{plus}\ 1\ 1 &= \big(\lambda m.\lambda n.\lambda f.\lambda x.\ m\ f\ (n\ f\ x)\big)\ 1\ 1 \\ &\to \lambda f.\lambda x.\ 1\ f\ (1\ f\ x) \\ &\to \lambda f.\lambda x.\ f\ (1\ f\ x) \\ &\to \lambda f.\lambda x.\ f\ (f\ x) \\ &= 2\end{aligned}$$

Multiplying Church Numerals

We can multiply:

Multiplying Church Numerals

We can multiply:

$$\text{mult} = \lambda m.\lambda n.\lambda f.\ m\ (n\ f)$$

Multiplying Church Numerals

We can multiply:

$$\text{mult} = \lambda m.\lambda n.\lambda f.\ m\ (n\ f)$$

$$\begin{aligned}
\text{mult}\ 2\ 3 &= \big(\lambda m.\lambda n.\lambda f.\ m\ (n\ f)\big)\ 2\ 3 \\
&\to \lambda f.\ 2\ (3\ f) \\
&\to \lambda f.\ 2\ (\lambda x.\ f(f(f\ x))) \\
&\leftrightarrow_\alpha \lambda f.\ 2\ (\lambda y.\ f(f(f\ y))) \\
&\to \lambda f.\lambda x.\ (\lambda y.\ f(f(f\ y)))\ ((\lambda y.\ f(f(f\ y)))\ x) \\
&\to \lambda f.\lambda x.\ (\lambda y.\ f(f(f\ y)))\ (f(f(f\ x))) \\
&\to \lambda f.\lambda x.\quad f(f(f(f(f(f\ x))))) \\
&= 6
\end{aligned}$$

Multiplying Church Numerals

We can multiply:

$$\text{mult} = \lambda m.\lambda n.\lambda f.\lambda x.\ m\ (n\ f)\ x$$

$$\begin{aligned}
\text{mult}\ 2\ 3 &= \big(\lambda m.\lambda n.\lambda f.\lambda x.\ m\ (n\ f)\ x\big)\ 2\ 3 \\
&\to \lambda f.\lambda x.\ 2\ (3\ f)\ x \\
&\to \lambda f.\lambda x.\ 2\ (\lambda x.\ f(f(f\ x)))\ x \\
&\leftrightarrow_\alpha \lambda f.\lambda x.\ 2\ (\lambda y.\ f(f(f\ y)))\ x \\
&\to \lambda f.\lambda x.\ (\lambda y.\ f(f(f\ y)))\ ((\lambda y.\ f(f(f\ y)))\ x) \\
&\to \lambda f.\lambda x.\ (\lambda y.\ f(f(f\ y)))\ (f(f(f\ x))) \\
&\to \lambda f.\lambda x.\qquad f(f(f(f(f(f\ x))))) \\
&= 6
\end{aligned}$$

The Y Combinator

Y Combinator: The function that takes a function f and returns $f(f(f(f(\cdots))))$, for recursion.

$$Y = \lambda f.\big(\lambda x.\ f\ (x\ x)\big)\big(\lambda x.\ f\ (x\ x)\big)$$

$$\begin{aligned} Y\ H &= \Big(\lambda f.\big(\lambda x.\ f\ (x\ x)\big)\big(\lambda x.\ f\ (x\ x)\big)\Big)\ H \\ &\to \big(\lambda x.\ H\ (x\ x)\big)\big(\lambda x.\ H\ (x\ x)\big) \\ &\to H\ \Big(\big(\lambda x.\ H\ (x\ x)\big)\big(\lambda x.\ H\ (x\ x)\big)\Big) \\ &\leftrightarrow H\ (Y\ H) \end{aligned}$$

Alonzo Church

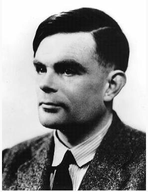
1903–1995
Professor at Princeton (1929–1967)
and UCLA (1967–1990)
Invented the Lambda Calculus

Had a few successful graduate students, including

- Stephen Kleene (Regular expressions)
- Michael O. Rabin† (Nondeterministic automata)
- Dana Scott† (Formal programming language semantics)
- Alan Turing (Turing machines)

† Turing award winners

Turing Machines vs. Lambda Calculus



In 1936,

- Alan Turing invented the Turing machine
- Alonzo Church invented the lambda calculus

In 1937, Turing proved that the two models were equivalent, i.e., that they define the same class of computable functions.

Modern processors are just overblown Turing machines.

Functional languages are just the lambda calculus with a more