Lambda calculus for dummies

David Luposchainsky

2019-10-19

Lambdas today

Everyone has them

```
// Javascript
addOne = function(x) { return x+1; } // Old style
addOne = x => x+1;                   // ES6 style
```

Everyone has them

```
// Javascript
addOne = function(x) { return x+1; } // Old style
addOne = x => x+1;                   // ES6 style

// Java
boilerplate addOne = x -> return x+1;
```

Everyone has them

```
// Javascript
addOne = function(x) { return x+1; } // Old style
addOne = x => x+1;                   // ES6 style

// Java
boilerplate addOne = x -> return x+1;

// C++
auto addOne = [](int x) { return x+1; }
```

Everyone has them

```
// Javascript
addOne = function(x) { return x+1; } // Old style
addOne = x => x+1;                   // ES6 style

// Java
boilerplate addOne = x -> return x+1;

// C++
auto addOne = [](int x) { return x+1; }

-- Haskell
addOne = \x -> x+1
```

But why?

- Functions as first class values
- Long forgotten/ignored, then rediscovered

But why?

- Functions as first class values
- Long forgotten/ignored, then rediscovered

```
map (\x -> x+1) [1,2,3] -- [2,3,4]
```

But why?

- Functions as first class values
- Long forgotten/ignored, then rediscovered

```
map (\x -> x+1) [1,2,3] -- [2,3,4]
```

```
http.createServer((request, response) => {
    doStuff(request);
    response.writeHead(200, {'Content-Type': 'text/plain'});
    response.end('Hello World\n');
}).listen(1337, '127.0.0.1');
```

Back a century

Same thing, different views

Researching foundations of mathematics and computation

- Turing: imperative machine emulates mathematician
- Church: mathematical construct emulates mathematics

Turing machines

- Technical and somewhat complicated
- Powerful
- Theoretical use: enormous
- Practical use: saying wrong or useless things on the internet

Definition of a Turing machine

A Turing machine consists of

$$
\begin{aligned}
Q &\ \text{finite set of states}\\
\Gamma &\ \text{finite set of tape symbols}\\
b \in \Gamma &\ \text{blank symbol}\\
\Sigma \subseteq \Gamma \setminus \{b\} &\ \text{set of input symbols}\\
q_0 \in Q &\ \text{initial state}\\
F \subseteq Q &\ \text{set of final states}\\
\delta : (Q \setminus F) \times \Gamma \to Q \times \Gamma \times \{L, R\} &\ \text{transition function}
\end{aligned}
$$

Lambda calculus

- Extremely simple
- Just as powerful
- Theoretical use: enormous
- Practical use: incrementing lists of numbers

Lambda calculus

Definition of Lambda Calculus

Terms
- x
- $\lambda x.T$
- $T_1 \ T_2$

Definition of Lambda Calculus

Terms
- x
- $\lambda x.T$
- $T_1 \ T_2$

Example terms
- $\lambda x.\ x$
- $\lambda x.\ (\lambda f.\ f\,(f\,x))$

Definition of Lambda Calculus

Terms
- x
- $\lambda x. T$
- $T_1 \;\; T_2$

Evaluation
- $\lambda x. T_x \equiv \lambda y. T_y$
- $(\lambda x. T) \; y \rightsquigarrow T_{x \rightarrow y}$
- $\lambda x. (f \, x) \equiv f$

Evaluation: example 1

$$((\lambda x.\ (\lambda y.\ x))\ 1)\ 2$$

Evaluation: example 1

$$((\lambda x.\ (\lambda y.\ x))\ 1)\ 2$$

$$= ((\lambda y.\ x)_{x \to 1})\ 2$$

Evaluation: example 1

$$((\lambda x.\ (\lambda y.\ x))\ 1)\ 2$$

$$= ((\lambda y.\ x)_{x \rightarrow 1})\ 2$$

$$= (\lambda y.\ 1) 2$$

Evaluation: example 1

$$((\lambda x.\ (\lambda y.\ x))\ 1)\ 2$$

$$= ((\lambda y.\ x)_{x \to 1})\ 2$$

$$= (\lambda y.\ 1) 2$$

$$= 1_{y \to 2}$$

Evaluation: example 1

$$((\lambda x.\ (\lambda y.\ x))\ 1)\ 2$$

$$= ((\lambda y.\ x)_{x \to 1})\ 2$$

$$= (\lambda y.\ 1)2$$

$$= 1_{y \to 2}$$

$$= 1$$

Evaluation: example 2

$$((\lambda f.\ (\lambda x.\ (f\ x)))\ \text{double})\ 4$$

Evaluation: example 2

$$((\lambda f.\ (\lambda x.\ (f\ x)))\ \text{double})\ 4$$

$$= ((\lambda x.\ (f\ x))_{f \to \text{double}})\ 4$$

Evaluation: example 2

$$((\lambda f.\ (\lambda x.\ (f\ x)))\ \text{double})\ 4$$

$$= ((\lambda x.\ (f\ x))_{f \to \text{double}})\ 4$$

$$= (\lambda x.\ (\text{double}\ x))\ 4$$

Evaluation: example 2

$$((\lambda f.\ (\lambda x.\ (f\ x)))\ \mathsf{double})\ 4$$

$$= ((\lambda x.\ (f\ x))_{f \to \mathsf{double}})\ 4$$

$$= (\lambda x.\ (\mathsf{double}\ x))\ 4$$

$$= (\mathsf{double}\ x)_{x \to 4}$$

Evaluation: example 2

$$((\lambda f.\ (\lambda x.\ (f\ x)))\ \text{double})\ 4$$

$$= ((\lambda x.\ (f\ x))_{f \to \text{double}})\ 4$$

$$= (\lambda x.\ (\text{double}\ x))\ 4$$

$$= (\text{double}\ x)_{x \to 4}$$

$$= \text{double}\ 4$$

What do we need to make this practical?

- numbers
- booleans
- tuples
- lists
- enums

Numbers

$=$ repeated function application

- $0 := \lambda f\, x.\ x$
- $1 := \lambda f\, x.\ f\, x$
- $2 := \lambda f\, x.\ f\, (f\, x)$

Numbers

$=$ repeated function application

- $0 := \lambda f\, x.\ x$
- $1 := \lambda f\, x.\ f\, x$
- $2 := \lambda f\, x.\ f\,(f\, x)$

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

Numbers

$=$ repeated function application

- $0 := \lambda f\, x.\ x$
- $1 := \lambda f\, x.\ f\, x$
- $2 := \lambda f\, x.\ f\, (f\, x)$

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

$$\text{add} := \lambda m\ n.\ (\lambda f\, x.\ m\ f\, (n\ f\, x))$$

Numbers

$=$ repeated function application

- $0 := \lambda f\, x.\ x$
- $1 := \lambda f\, x.\ f\, x$
- $2 := \lambda f\, x.\ f\,(f\, x)$

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

$$\text{add} := \lambda m\ n.\ (\lambda f\, x.\ m\ f\,(n\ f\ x))$$

$$\text{mul} := \lambda m\ n.\ (\lambda f\, x.\ m\ (n\ f)\ x)$$

Booleans

$=$ ignore one branch

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

Booleans

$=$ ignore one branch

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

$$\begin{aligned}
\text{ifThenElse} &:= \lambda p\ t\ f.\ (p\ t)\ f \\
&\equiv \lambda p\ t.\ p\ t \\
&\equiv \lambda p.\ p \\
&\equiv \text{id}
\end{aligned}$$

Booleans

$=$ ignore one branch

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

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

$$\begin{aligned}\text{ifThenElse} :=& \lambda p\ t\ f.\ (p\ t)\ f\\ \equiv& \lambda p\ t.\ p\ t\\ \equiv& \lambda p.\ p\\ \equiv& \text{id}\end{aligned}$$

$$\begin{aligned}\text{not} &:= \lambda p.\ p\ \text{false}\ \text{true}\\ \text{and} &:= \lambda p\ q.\ p\ q\ p\\ \text{or} &:= \lambda p\ q.\ p\ p\ q\end{aligned}$$

Pairs/lists

$=$ nil/cons like in Lisp

$$
\begin{aligned}
\text{pair} &:= \lambda x\ y\ f.\ f\ x\ y \\
\text{first} &:= \lambda p.\ p\ \text{true} \\
\text{second} &:= \lambda p.\ p\ \text{false} \\
\text{nil} &:= \lambda x.\ \text{true} \\
\text{null} &:= \lambda p.\ p\ (\lambda x\ y.\ \text{false})
\end{aligned}
$$

Includes, modules

$$(\lambda \mathit{helper}.\langle\text{program}\rangle)$$
$$(\langle\text{value for helper}\rangle)$$

Includes, modules

$$(\lambda \mathit{helper}.\langle\text{program}\rangle)$$
$$(\langle\text{value for helper}\rangle)$$

$$(\lambda \mathit{add}.\ \mathit{add}\ 3\ 4)$$
$$(\lambda m\ n.\ (\lambda f\ x.\ m\ f\ (n\ f\ x)))$$

Prettier: let

$$
\begin{array}{l}
(\lambda let. \\
\quad (let\ (\lambda m\ n.\ (\lambda f\ x.\ m\ f\ (n\ f\ x)))\ (\lambda\ add. \\
\qquad add\ 3\ 4)) \\
)\ (\lambda value\ body.\ body\ value)
\end{array}
$$

Recursion

...but how?

$$
\begin{aligned}
\text{factorial} := \lambda n.\ \text{ifThenElse}\ & (n > 0) \\
& (n * \mathit{recurse}\ (n - 1)) \\
& 1
\end{aligned}
$$

Recursion: fixed point combinator

$$\begin{aligned} \mathsf{Y}\ f &= f\,(\mathsf{Y}\ f) \\ &= f\,(f\,(\mathsf{Y}\ f)) \\ &= f\,(f\,(f\,(\ldots))) \end{aligned}$$

Recursion: fixed point combinator

$$\begin{aligned} \mathsf{Y}\ f &= f\,(\mathsf{Y}\ f) \\ &= f\,(f\,(\mathsf{Y}\ f)) \\ &= f\,(f\,(f\,(\ldots))) \end{aligned}$$

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

Y combinator usage

$$\begin{aligned}\text{factorialStep} := \lambda \mathit{rec}\ n.\ \text{ifThenElse}\ & (n > 0) \\ & (n * \mathit{rec}\ (n - 1)) \\ & 1\end{aligned}$$

Y combinator usage

$$\begin{aligned}\text{factorialStep} := \lambda rec\ n.\ \text{ifThenElse}\ &(n > 0) \\ &(n * rec\ (n-1)) \\ &1\end{aligned}$$

$$\text{factorial} := \text{Y factorialStep}$$

Example

$$\begin{aligned}
\text{factorial } 3 &= (\text{Y factorialStep})\ 3 \\
&= \text{factorialStep (Y factorialStep) } 3 \\
&= (\lambda \mathit{rec}.\ \text{ifThenElse } (3 > 0)\ (3 \cdot \mathit{rec}\ (3 - 1))\ 1)\ (\text{Y factorialStep}) \\
&= (\lambda \mathit{rec}.\ (3 \cdot \mathit{rec}\ 2))\ (\text{Y factorialStep}) \\
&= 3 \cdot \text{factorial } 2 \\
&= 3 \cdot 2 \cdot 1 \cdot \text{factorial } 0 \\
&= 6 \cdot \text{Y factorialStep } 0 \\
&= 6 \cdot (\lambda \mathit{rec}.\ \text{ifThenElse } (0 > 0)\ (0 \cdot \mathit{rec}\ (0 - 1))\ 1)\ (\text{Y factorialStep}) \\
&= 6 \cdot (\lambda \mathit{rec}.\ 1)\ (\text{Y factorialStep}) \\
&= 6 \cdot 1 \\
&= 6
\end{aligned}$$

What now?

- Invent Lisp
- Add types (simply typed LC, Hindley/Milner)
- CPS transform your whole program to make maintenance a nightmare

Questions?

```
fibo :=
    (\let.
        let (\x _. x)                                       (\ true.
        let (\_ y. y)                                       (\ false.
        let (\x. x)                                         (\ ifThenElse.
        let (\f. (\x. f (x x)) (\x. f (x x)))               (\ Y.
        let (\f x. f x)                                     (\ 1.
        let (\f x. f (f x))                                 (\ 2.
        let (\n f x. n (\g h. h (g f)) (\_. x) (\u. u)) (\ pred.
        let (\n. n (\x. false) true)                        (\ =0.
        let (\m n f x. n f (m f x))                         (\ +.
        let (\m n. n pred m)                                (\ -.
        let (\m n. =0 (- m n))                              (\ <=.
            Y (\fib n. ifThenElse (<= n 2)
                           n
                           (+ (fib (- n 1))
                              (fib (- n 2)))))
        ))))))))))
    ) (\value body. body value)
```

Church-Rosser theorem, evaluation strategies

»Normal forms are unique«

Church-Rosser theorem, evaluation strategies

»Normal forms are unique«

Applicative order

- Reduce the rightmost-innermost redex first
- Used by eager languages (Lisps, C, Java, ...)
- Might not find normal form even though it exists:

$$(\lambda x\ y.\ x)\ 1\ \underbrace{((\lambda x.\ x\ x)\ (\lambda x.\ x\ x))}_{\Omega} \leadsto \perp$$

Church-Rosser theorem, evaluation strategies

»Normal forms are unique«

Applicative order

- Reduce the rightmost-innermost redex first
- Used by eager languages (Lisps, C, Java, …)
- Might not find normal form even though it exists:

$$(\lambda x\ y.\ x)\ 1\ \underbrace{((\lambda x.\ x\ x)\ (\lambda x.\ x\ x))}_{\Omega} \leadsto \bot$$

Normal order

- Reduce the leftmost-outermost redex first
- Used by lazy languages (Haskell, Miranda)
- If there is a normal form, it will be reached

$$(\lambda x\ y.\ x)\ 1\ \underbrace{((\lambda x.\ x\ x)\ (\lambda x.\ x\ x))}_{\Omega} \leadsto 1$$

SKI calculus

»Lambda calculus is too complicated!«

$$
\begin{aligned}
K\; x\; y &\longrightarrow x \\
S\; f\; g\; x &\longrightarrow f\; x\; (g\; x)
\end{aligned}
$$

Universal iota

»SK calculus is too complicated!«

$$\iota := \lambda f.\ f\ S\ K = S\ (S\ I\ (K\ S))\ (K\ K)$$
$$K = (\iota(\iota(\iota\iota)))$$
$$S = (\iota(\iota(\iota(\iota\iota))))$$