Lambda Calculus

Isaac Elliott

April 30, 2019

What is computation?

What is computation?

State-transition systems

- Finite state machines

What is computation?

State-transition systems

- Finite state machines
- Cellular automata

What is computation?

State-transition systems

- Finite state machines
- Cellular automata
- Turing machines

What is computation?

State-transition systems

Computation = state changing (according to some rules) over time?

What is computation?

State-transition systems

$$f(x) = x^2 + 2x + 1$$

$$f(1)$$

$$4$$

λ-calculus

λ-calculus

Definition

Three expression forms:

λ-calculus

Definition

Three expression forms:

1. $\lambda x.\ e$ (abstraction)

λ-calculus

Definition

Three expression forms:

1. $\lambda x.\ e$ (abstraction)
2. $a\ b$ (application - nests to the left - $a\ b\ c$ is $(a\ b)\ c$)

λ-calculus

Definition

Three expression forms:

1. $\lambda x.\ e$ (abstraction)
2. $a\ b$ (application - nests to the left - $a\ b\ c$ is $(a\ b)\ c$)
3. x (variable)

λ-calculus

Definition

A syntactic operation (substitution)

λ-calculus
Definition

A syntactic operation (substitution)

- $x[x \mapsto a] \rightsquigarrow a$

λ-calculus

Definition

A syntactic operation (substitution)

- $x[x \mapsto a] \rightsquigarrow a$
- $y[x \mapsto a] \rightsquigarrow y \quad$ (given $y \neq x$)

λ-calculus
Definition

A syntactic operation (substitution)

- $x[x \mapsto a] \rightsquigarrow a$
- $y[x \mapsto a] \rightsquigarrow y \quad$ (given $y \neq x$)
- $(m\ n)[x \mapsto a] \rightsquigarrow m[x \mapsto a]\ n[x \mapsto a]$

λ-calculus

Definition

A syntactic operation (substitution)

- $x[x \mapsto a] \rightsquigarrow a$
- $y[x \mapsto a] \rightsquigarrow y \quad \text{(given } y \neq x\text{)}$
- $(m\ n)[x \mapsto a] \rightsquigarrow m[x \mapsto a]\ n[x \mapsto a]$
- $(\lambda x.\ e)[x \mapsto a] \rightsquigarrow \lambda x.\ e$

λ-calculus

Definition

A syntactic operation (substitution)

- $x[x \mapsto a] \leadsto a$
- $y[x \mapsto a] \leadsto y \quad (\text{given } y \neq x)$
- $(m\ n)[x \mapsto a] \leadsto m[x \mapsto a]\ n[x \mapsto a]$
- $(\lambda x.\ e)[x \mapsto a] \leadsto \lambda x.\ e$
- $(\lambda y.\ e)[x \mapsto a] \leadsto \lambda y.\ e[x \mapsto a] \quad (\text{given } y \neq x)$

A reduction rule (beta reduction)

λ-calculus

Definition

A reduction rule (beta reduction)

- $(\lambda x.e)\ a \rightsquigarrow e[x \mapsto a]$

λ-calculus

Terminology

λ-calculus

Terminology

Bound variable - a variable that has been abstracted over by a lambda
Free variable - a variable that is not bound

λ-calculus

Terminology

Bound variable - a variable that has been abstracted over by a lambda
Free variable - a variable that is not bound

- $\lambda x.\ x$

λ-calculus

Terminology

Bound variable - a variable that has been abstracted over by a lambda
Free variable - a variable that is not bound

- $\lambda x.\ x$
- $\lambda a.\ \lambda b.\ a\ b$

λ-calculus

Terminology

Bound variable - a variable that has been abstracted over by a lambda
Free variable - a variable that is not bound

- $\lambda x.\ x$
- $\lambda a.\ \lambda b.\ a\ b$
- $\lambda m.\ \lambda n.\ m\ o$

λ-calculus

Terminology

Bound variable - a variable that has been abstracted over by a lambda
Free variable - a variable that is not bound

- $\lambda x.\ x$
- $\lambda a.\ \lambda b.\ a\ b$
- $\lambda m.\ \lambda n.\ m\ o$

Substitution can only affect free variables

λ-calculus

Terminology

Bound variable - a variable that has been abstracted over by a lambda
Free variable - a variable that is not bound

- $\lambda x.\ x$
- $\lambda a.\ \lambda b.\ a\ b$
- $\lambda m.\ \lambda n.\ m\ o$

Substitution can only affect free variables

- $x[x \mapsto y] \rightsquigarrow y$

λ-calculus

Terminology

Bound variable - a variable that has been abstracted over by a lambda
Free variable - a variable that is not bound

- $\lambda x.\ x$
- $\lambda a.\ \lambda b.\ a\ b$
- $\lambda m.\ \lambda n.\ m\ o$

Substitution can only affect free variables

- $x[x \mapsto y] \rightsquigarrow y$
- $(\lambda x.\ x)[x \mapsto y] \rightsquigarrow \lambda x.\ x$

λ-calculus

Terminology

Beta redex - beta red(ucible) ex(pression)

λ-calculus

Terminology

Beta redex - beta red(ucible) ex(pression)

- $(\lambda a.\ a)\ (\lambda b.\ b)$

λ-calculus

Terminology

Beta redex - beta red(ucible) ex(pression)

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

λ-calculus

Terminology

Beta redex - beta red(ucible) ex(pression)

- $(\lambda a.\ a)\ (\lambda b.\ b)$
- $(\lambda x.\ \lambda y.\ x)\ a$
- $(\lambda m.\ x)\ b$

λ-calculus

Terminology

Value - anything that's not a beta redex

Value - anything that's not a beta redex

- a

λ-calculus

Terminology

Value - anything that's not a beta redex

- a
- $\lambda x.\ x$

λ-calculus
Terminology

Value - anything that's not a beta redex

- a
- $\lambda x.\ x$
- $\lambda a.\ \lambda b.\ (\lambda x.\ x)\ a$

λ-calculus

Terminology

Evaluate - to reduce to a value

λ-calculus

Examples

$$(\lambda x.\ x)\ a$$

λ-calculus

Examples

$$(\lambda x.\ x)\ a$$

$$x[x \mapsto a]$$

λ-calculus

Examples

$$(\lambda x.\ x)\ a$$

$$x[x \mapsto a]$$

$$a$$

λ-calculus

Examples

$$(\lambda f.\ f\ x)\ (\lambda b.\ b)$$

λ-calculus

Examples

$$(\lambda f.\ f\ x)\ (\lambda b.\ b)$$

$$(f\ x)[f \mapsto \lambda b.\ b]$$

λ-calculus

Examples

$$(\lambda f.\ f\ x)\ (\lambda b.\ b)$$

$$(f\ x)[f \mapsto \lambda b.\ b]$$

$$f[f \mapsto \lambda b.\ b]\ x[f \mapsto \lambda b.\ b]$$

λ-calculus

Examples

$$(\lambda f.\ f\ x)\ (\lambda b.\ b)$$
$$(f\ x)[f \mapsto \lambda b.\ b]$$
$$f[f \mapsto \lambda b.\ b]\ x[f \mapsto \lambda b.\ b]$$
$$(\lambda b.\ b)\ x$$

λ-calculus

Examples

$$(\lambda f.\ f\ x)\ (\lambda b.\ b)$$
$$(f\ x)[f \mapsto \lambda b.\ b]$$
$$f[f \mapsto \lambda b.\ b]\ x[f \mapsto \lambda b.\ b]$$
$$(\lambda b.\ b)\ x$$
$$b[b \mapsto x]$$

λ-calculus

Examples

$$(\lambda f.\ f\ x)\ (\lambda b.\ b)$$
$$(f\ x)[f \mapsto \lambda b.\ b]$$
$$f[f \mapsto \lambda b.\ b]\ x[f \mapsto \lambda b.\ b]$$
$$(\lambda b.\ b)\ x$$
$$b[b \mapsto x]$$
$$x$$

λ-calculus

Currying

Maths	λ-calculus

λ-calculus

Currying

Maths	λ-calculus
$f(x) = \ldots$	$f = \lambda x.\ \ldots$

λ-calculus

Currying

Maths	λ-calculus
$f(x) = \ldots$	$f = \lambda x.\ \ldots$
$g(x, y) = \ldots$	$g = \lambda x.\ \lambda y.\ \ldots$

λ-calculus

Currying

Maths	λ-calculus
$f(x) = \ldots$	$f = \lambda x.\ \ldots$
$g(x, y) = \ldots$	$g = \lambda x.\ \lambda y.\ \ldots$
$g(a, b)$	$g\ a\ b$

λ-calculus

How does λ-calculus compare to Turing machines in terms of computability?

λ-calculus

How does λ-calculus compare to Turing machines in terms of computability?

- They are equivalent (Church-Turing thesis)

Church Encoding

Church Encoding

Booleans

Boolean values

Church Encoding

Booleans

Boolean values

- $\texttt{false} = \lambda x.\, \lambda y.\, x$

Church Encoding

Booleans

Boolean values

- $\texttt{false} = \lambda x.\, \lambda y.\, x$
- $\texttt{true} = \lambda x.\, \lambda y.\, y$

Church Encoding

Booleans

$$not = ?$$

Church Encoding

Booleans

$$\mathit{not}(x) = \begin{cases} \texttt{true} & \mapsto \texttt{false} \\ \texttt{false} & \mapsto \texttt{true} \end{cases}$$

Church Encoding

Booleans

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

$$\mathit{not} = \ ?$$

Church Encoding

Booleans

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

$$
\mathit{not} = \quad \lambda b.\, b\ \texttt{true}\ \texttt{false}
$$

Church Encoding

Booleans

$\textit{not}$ `true`

Church Encoding

Booleans

$\textit{not}\ \texttt{true}$

$$(\lambda b.\ b\ \texttt{true}\ \texttt{false})\ \texttt{true}$$

Church Encoding

Booleans

$$\textit{not}\ \texttt{true}$$

$$(\lambda b.\ b\ \texttt{true}\ \texttt{false})\ \texttt{true}$$

$$(b\ \texttt{true}\ \texttt{false})[b \mapsto \texttt{true}]$$

Church Encoding

Booleans

$$\textit{not}\ \texttt{true}$$

$$(\lambda b.\ b\ \texttt{true}\ \texttt{false})\ \texttt{true}$$

$$(b\ \texttt{true}\ \texttt{false})[b \mapsto \texttt{true}]$$

$$\texttt{true}\ \texttt{true}\ \texttt{false}$$

Church Encoding

Booleans

$$\textit{not}\ \texttt{true}$$
$$(\lambda b.\ b\ \texttt{true}\ \texttt{false})\ \texttt{true}$$
$$(b\ \texttt{true}\ \texttt{false})[b \mapsto \texttt{true}]$$
$$\texttt{true}\ \texttt{true}\ \texttt{false}$$
$$(\lambda x.\ \lambda y.\ y)\ \texttt{true}\ \texttt{false}$$

Church Encoding

Booleans

$$\mathit{not}\ \texttt{true}$$
$$(\lambda b.\ b\ \texttt{true}\ \texttt{false})\ \texttt{true}$$
$$(b\ \texttt{true}\ \texttt{false})[b \mapsto \texttt{true}]$$
$$\texttt{true}\ \texttt{true}\ \texttt{false}$$
$$(\lambda x.\ \lambda y.\ y)\ \texttt{true}\ \texttt{false}$$
$$(\lambda y.\ y)[x \mapsto \texttt{true}]\ \texttt{false}$$

Church Encoding

Booleans

$$\textit{not}\ \texttt{true}$$
$$(\lambda b.\ b\ \texttt{true}\ \texttt{false})\ \texttt{true}$$
$$(b\ \texttt{true}\ \texttt{false})[b \mapsto \texttt{true}]$$
$$\texttt{true}\ \texttt{true}\ \texttt{false}$$
$$(\lambda x.\ \lambda y.\ y)\ \texttt{true}\ \texttt{false}$$
$$(\lambda y.\ y)[x \mapsto \texttt{true}]\ \texttt{false}$$
$$(\lambda y.\ y)\ \texttt{false}$$

Church Encoding

Booleans

$$\mathit{not}\ \mathtt{true}$$
$$(\lambda b.\ b\ \mathtt{true}\ \mathtt{false})\ \mathtt{true}$$
$$(b\ \mathtt{true}\ \mathtt{false})[b \mapsto \mathtt{true}]$$
$$\mathtt{true}\ \mathtt{true}\ \mathtt{false}$$
$$(\lambda x.\ \lambda y.\ y)\ \mathtt{true}\ \mathtt{false}$$
$$(\lambda y.\ y)[x \mapsto \mathtt{true}]\ \mathtt{false}$$
$$(\lambda y.\ y)\ \mathtt{false}$$
$$y[y \mapsto \mathtt{false}]$$

Church Encoding

Booleans

$$\mathit{not}\ \texttt{true}$$
$$(\lambda b.\ b\ \texttt{true}\ \texttt{false})\ \texttt{true}$$
$$(b\ \texttt{true}\ \texttt{false})[b \mapsto \texttt{true}]$$
$$\texttt{true}\ \texttt{true}\ \texttt{false}$$
$$(\lambda x.\ \lambda y.\ y)\ \texttt{true}\ \texttt{false}$$
$$(\lambda y.\ y)[x \mapsto \texttt{true}]\ \texttt{false}$$
$$(\lambda y.\ y)\ \texttt{false}$$
$$y[y \mapsto \texttt{false}]$$
$$\texttt{false}$$

Church Encoding

Booleans

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

$$\mathit{and} = ?$$

Church Encoding

Booleans

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

$$\mathit{and} = \lambda a.\, \lambda b.\, a\ \texttt{false}\ b$$

Church Encoding

Booleans

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

$$\mathit{and} = \lambda a.\, \lambda b.\, a\ \texttt{false}\ b$$

Exercises:

Church Encoding

Booleans

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

$$\mathit{and} = \lambda a.\ \lambda b.\ a\ \texttt{false}\ b$$

Exercises:

- $\mathit{or} = ?$

Church Encoding

Booleans

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

$$\mathit{and} = \lambda a.\, \lambda b.\, a\ \texttt{false}\ b$$

Exercises:

- $\mathit{or} = ?$
- $\mathit{iff} = ?$

Church Encoding

Booleans

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

$$and = \lambda a.\, \lambda b.\, a\ \texttt{false}\ b$$

Exercises:

- $or = ?$
- $iff = ?$
- $xor = ?$

Church Encoding

Booleans

Booleans embody *choice*

Pairs

Church Encoding

Pairs

Pairs

- $\langle a, b \rangle = \lambda f.\ f\ a\ b$

Church Encoding

Pairs

$$\langle a, b \rangle = \lambda f.\ f\ a\ b$$

$$\texttt{fst} = ?$$

Church Encoding

Pairs

$$\langle a, b \rangle = \lambda f.\ f\ a\ b$$

$$\texttt{fst} = \ \lambda p.\ p\ (\lambda x.\ \lambda y.\ x)$$

Church Encoding

Pairs

$$\texttt{fst} = \lambda p.\ p\,(\lambda x.\ \lambda y.\ x)$$

$$\texttt{snd} = \lambda p.\ p\,(\lambda x.\ \lambda y.\ y)$$

Church Encoding

Pairs

$$\texttt{fst} = \lambda p.\ p\ (\lambda x.\ \lambda y.\ x)$$

$$\texttt{snd} = \lambda p.\ p\ (\lambda x.\ \lambda y.\ y)$$

Church Encoding

Pairs

$$\texttt{fst} = \lambda p.\ p\,(\lambda x.\ \lambda y.\ x)$$

$$\texttt{snd} = \lambda p.\ p\,(\lambda x.\ \lambda y.\ y)$$

Exercises:

Church Encoding

Pairs

$$\texttt{fst} = \lambda p.\ p\ (\lambda x.\ \lambda y.\ x)$$

$$\texttt{snd} = \lambda p.\ p\ (\lambda x.\ \lambda y.\ y)$$

Exercises:

- $swap = ?\qquad (\langle a, b\rangle \rightarrow \langle b, a\rangle)$

Church Encoding

Pairs

$$\texttt{fst} = \lambda p.\ p\ (\lambda x.\ \lambda y.\ x)$$

$$\texttt{snd} = \lambda p.\ p\ (\lambda x.\ \lambda y.\ y)$$

Exercises:

- $\mathit{swap} = ?$ $\quad (\langle a, b\rangle \rightarrow \langle b, a\rangle)$
- $\langle a, b, c\rangle = ?$

Church Encoding

Pairs

$$\texttt{fst} = \lambda p.\ p\,(\lambda x.\ \lambda y.\ x)$$

$$\texttt{snd} = \lambda p.\ p\,(\lambda x.\ \lambda y.\ y)$$

Exercises:

- $swap = ?\qquad (\langle a, b\rangle \to \langle b, a\rangle)$
- $\langle a, b, c\rangle = ?$
- $2to3 = ?\qquad (\langle a, \langle b, c\rangle\rangle \to \langle a, b, c\rangle)$

Church Encoding

Pairs

$$\texttt{false} = \lambda x.\ \lambda y.\ x$$

$$\texttt{true} = \lambda x.\ \lambda y.\ y$$

$$\texttt{fst} = \lambda p.\ p\ (\lambda x.\ \lambda y.\ x)$$

$$\texttt{snd} = \lambda p.\ p\ (\lambda x.\ \lambda y.\ y)$$

Church Encoding

Pairs

$$\texttt{false} = \lambda x.\, \lambda y.\, x$$

$$\texttt{true} = \lambda x.\, \lambda y.\, y$$

$$\texttt{fst} = \lambda p.\, p\, \texttt{false}$$

$$\texttt{snd} = \lambda p.\, p\, \texttt{true}$$

Church Encoding

Pairs

$$\texttt{false} = \lambda x.\, \lambda y.\, x$$

$$\texttt{true} = \lambda x.\, \lambda y.\, y$$

$$\texttt{fst} = \lambda p.\, p\, \texttt{false}$$

$$\texttt{snd} = \lambda p.\, p\, \texttt{true}$$

You can *choose* whether you want the first or second element

Church Encoding

Natural numbers

How do you count with functions?

Church Encoding

Natural numbers

How do you count with functions?

- $\lambda f.\, \lambda x.\, x$

Church Encoding

Natural numbers

How do you count with functions?

- $\lambda f.\, \lambda x.\, x$
- $\lambda f.\, \lambda x.\, f\ x$

Church Encoding

Natural numbers

How do you count with functions?

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

Church Encoding

Natural numbers

How do you count with functions?

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

Church Encoding

Natural numbers

How do you count with functions?

- $\lambda f.\ \lambda x.\ x$
- $\lambda f.\ \lambda x.\ f\ x$
- $\lambda f.\ \lambda x.\ f\ (f\ x)$
- $\lambda f.\ \lambda x.\ f\ (f\ (f\ x))$
- $\ldots$

Church Encoding

Natural numbers

How do you count with functions?

- $\lambda f.\ \lambda x.\ x$
- $\lambda f.\ \lambda x.\ f\ x$
- $\lambda f.\ \lambda x.\ f\ (f\ x)$
- $\lambda f.\ \lambda x.\ f\ (f\ (f\ x))$
- $\dots$

The natural number N is represented by N nestings of a function

Church Encoding

Natural numbers

How do you count with functions?

- $\lambda f.\ \lambda x.\ x$
- $\lambda f.\ \lambda x.\ f\ x$
- $\lambda f.\ \lambda x.\ f\ (f\ x)$
- $\lambda f.\ \lambda x.\ f\ (f\ (f\ x))$
- $\ldots$

The natural number N is represented by N nestings of a function

The essence of a natural number is *iteration*

Church Encoding

Natural numbers

- $\texttt{zero} = \lambda f.\ \lambda x.\ x$
- $\texttt{suc} = \lambda n.\ \lambda f.\ \lambda x.\ f\ (n\ f\ x)$

Church Encoding

Natural numbers

`suc zero`

Church Encoding

Natural numbers

$\texttt{suc zero}$

$$(\lambda n.\ \lambda f.\ \lambda x.\ f\ (n\ f\ x))\ \texttt{zero}$$

Church Encoding

Natural numbers

$$\texttt{suc zero}$$

$$(\lambda n.\ \lambda f.\ \lambda x.\ f\ (n\ f\ x))\ \texttt{zero}$$

$$(\lambda f.\ \lambda x.\ f\ (n\ f\ x))[n \mapsto \texttt{zero}]$$

Church Encoding

Natural numbers

$$\texttt{suc zero}$$

$$(\lambda n.\ \lambda f.\ \lambda x.\ f\ (n\ f\ x))\ \texttt{zero}$$

$$(\lambda f.\ \lambda x.\ f\ (n\ f\ x))[n \mapsto \texttt{zero}]$$

$$\lambda f.\ \lambda x.\ f\ (\texttt{zero}\ f\ x)$$

Church Encoding
Natural numbers

$$add = ?$$

Church Encoding

Natural numbers

$$add = \lambda m.\ \lambda n.\ m\ \mathtt{suc}\ n$$

Church Encoding
Natural numbers

$$add = \lambda m.\ \lambda n.\ m\ \mathtt{suc}\ n$$

Exercises:

Church Encoding

Natural numbers

$$add = \lambda m.\ \lambda n.\ m\ \mathtt{suc}\ n$$

Exercises:

- $mult = ?$ (iterated addition)

Church Encoding

Natural numbers

$$add = \lambda m.\ \lambda n.\ m\ \mathtt{suc}\ n$$

Exercises:

- $mult = ?$ (iterated addition)
- $exp = ?$ (iterated multiplication)

Church Encoding

Natural numbers

$$add = \lambda m.\ \lambda n.\ m\ \mathtt{suc}\ n$$

Exercises:

- $mult = ?$ (iterated addition)
- $exp = ?$ (iterated multiplication)
- $pred = ?$ (iterated uh... very difficult)

Repetition

Repetition

Omega

$$\Omega = (\lambda x.\ x\ x)\ (\lambda x.\ x\ x)$$

Repetition

Omega

$$\Omega = (\lambda x.\ x\ x)\ (\lambda x.\ x\ x)$$

$$(\lambda x.\ x\ x)\ (\lambda x.\ x\ x)$$

Repetition

Omega

$$\Omega = (\lambda x.\ x\ x)\ (\lambda x.\ x\ x)$$

$$(\lambda x.\ x\ x)\ (\lambda x.\ x\ x)$$
$$(x\ x)[x \mapsto \lambda x.\ x\ x]$$

Repetition

Omega

$$\Omega = (\lambda x.\ x\ x)\ (\lambda x.\ x\ x)$$

$$(\lambda x.\ x\ x)\ (\lambda x.\ x\ x)$$
$$(x\ x)[x \mapsto \lambda x.\ x\ x]$$
$$(\lambda x.\ x\ x)\ (\lambda x.\ x\ x)$$

Repetition

The Y-combinator

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

Repetition

The Y-combinator

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

$$Y\ g$$

Repetition

The Y-combinator

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

$$Y\ g$$

$$(\lambda f.\, (\lambda x.\, f\ (x\ x))\ (\lambda x.\, f\ (x\ x)))\ g$$

Repetition

The Y-combinator

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

$$Y\ g$$

$$(\lambda f.\, (\lambda x.\, f\ (x\ x))\ (\lambda x.\, f\ (x\ x)))\ g$$

$$((\lambda x.\, f\ (x\ x))\ (\lambda x.\, f\ (x\ x)))[f \mapsto g]$$

Repetition

The Y-combinator

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

$$Y\ g$$
$$(\lambda f.\, (\lambda x.\, f\ (x\ x))\ (\lambda x.\, f\ (x\ x)))\ g$$
$$((\lambda x.\, f\ (x\ x))\ (\lambda x.\, f\ (x\ x)))[f \mapsto g]$$
$$(\lambda x.\, g\ (x\ x))\ (\lambda x.\, g\ (x\ x))$$

Repetition

The Y-combinator

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

$$Y\ g$$
$$(\lambda f.\, (\lambda x.\, f\ (x\ x))\ (\lambda x.\, f\ (x\ x)))\ g$$
$$((\lambda x.\, f\ (x\ x))\ (\lambda x.\, f\ (x\ x)))[f \mapsto g]$$
$$(\lambda x.\, g\ (x\ x))\ (\lambda x.\, g\ (x\ x))$$
$$(g\ (x\ x))[x \mapsto \lambda x.\, g\ (x\ x)]$$

Repetition

The Y-combinator

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

$$Y\ g$$
$$(\lambda f.\, (\lambda x.\, f\ (x\ x))\ (\lambda x.\, f\ (x\ x)))\ g$$
$$((\lambda x.\, f\ (x\ x))\ (\lambda x.\, f\ (x\ x)))[f \mapsto g]$$
$$(\lambda x.\, g\ (x\ x))\ (\lambda x.\, g\ (x\ x))$$
$$(g\ (x\ x))[x \mapsto \lambda x.\, g\ (x\ x)]$$
$$g\ ((\lambda x.\, g\ (x\ x))\ (\lambda x.\, g\ (x\ x)))$$

Repetition

The Y-combinator

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

$$\begin{gathered} Y\ g \\ (\lambda f.\, (\lambda x.\, f\ (x\ x))\ (\lambda x.\, f\ (x\ x)))\ g \\ ((\lambda x.\, f\ (x\ x))\ (\lambda x.\, f\ (x\ x)))[f \mapsto g] \\ (\lambda x.\, g\ (x\ x))\ (\lambda x.\, g\ (x\ x)) \\ (g\ (x\ x))[x \mapsto \lambda x.\, g\ (x\ x)] \\ g\ ((\lambda x.\, g\ (x\ x))\ (\lambda x.\, g\ (x\ x))) \end{gathered}$$

$$Y\ g \rightsquigarrow g(g(g(g(g \ldots))))$$

Repetition

Ackermann function

$$A(m,n) = \begin{cases} m = 0 & \mapsto n+1 \\ m > 0, n = 0 & \mapsto A(m-1,1) \\ m > 0, n > 0 & \mapsto A(m-1, A(m, n-1)) \end{cases}$$

Repetition

Ackermann function

$$pred = \ldots$$

$$is0 = \lambda n.\ n\ (\lambda x.\ \texttt{false})\ \texttt{true}$$

Repetition

Ackermann function

$$
\begin{array}{l}
A = \\
\quad Y \quad (\lambda f.\, \lambda m.\, \lambda n. \\
\qquad\quad is0\ m \\
\qquad\qquad (is0\ n \\
\qquad\qquad\quad (f\ (pred\ m)\ (f\ m\ (pred\ n))) \\
\qquad\qquad\quad (f\ (pred\ m)\ (\texttt{suc}\ \texttt{zero}))) \\
\qquad\qquad (\texttt{suc}\ n))
\end{array}
$$

Combinator Calculus

Combinator Calculus

Four expression forms:

Combinator Calculus

Four expression forms:

- $a\ b$ (application - nests to the left)

Combinator Calculus

Four expression forms:

- a b (application - nests to the left)
- `S` (**S**ubsitution)

Combinator Calculus

Four expression forms:

- *a b* (application - nests to the left)
- `S` (**S**ubsitution)
- `K` (**K**onstant)

Combinator Calculus

Four expression forms:

- *a b* (application - nests to the left)
- `S` (**S**ubsitution)
- `K` (**K**onstant)
- `I` (**I**dentity)

Combinator Calculus

Three reduction rules:

Combinator Calculus

Three reduction rules:

- $\mathtt{S}\ a\ b\ c \rightsquigarrow a\ c\ (b\ c)$

Combinator Calculus

Three reduction rules:

- $\mathtt{S}\ a\ b\ c \rightsquigarrow a\ c\ (b\ c)$
- $\mathtt{K}\ a\ b \rightsquigarrow a$

Combinator Calculus

Three reduction rules:

- $\mathtt{S}\ a\ b\ c \rightsquigarrow a\ c\ (b\ c)$
- $\mathtt{K}\ a\ b \rightsquigarrow a$
- $\mathtt{I}\ a \rightsquigarrow a$

Combinator Calculus

- Equivalent to λ-calculus, but without variable binding
- $\texttt{SKI} \rightarrow \lambda$

Combinator Calculus

- Equivalent to λ-calculus, but without variable binding
- $\texttt{SKI} \rightarrow \lambda$
  - $\texttt{S} \rightarrow \lambda a.\ \lambda b.\ \lambda c.\ a\ c\ (b\ c)$

Combinator Calculus

- Equivalent to λ-calculus, but without variable binding
- $\texttt{SKI} \to \lambda$
  - $\texttt{S} \to \lambda a.\ \lambda b.\ \lambda c.\ a\ c\ (b\ c)$
  - $\texttt{K} \to \lambda a.\ \lambda b.\ a$

Combinator Calculus

- Equivalent to λ-calculus, but without variable binding
- $\texttt{SKI} \to \lambda$
  - $\texttt{S} \to \lambda a.\ \lambda b.\ \lambda c.\ a\ c\ (b\ c)$
  - $\texttt{K} \to \lambda a.\ \lambda b.\ a$
  - $\texttt{I} \to \lambda a.\ a$

Combinator Calculus

- Equivalent to λ-calculus, but without variable binding
- $\texttt{SKI} \to \lambda$
  - $\texttt{S} \to \lambda a.\ \lambda b.\ \lambda c.\ a\ c\ (b\ c)$
  - $\texttt{K} \to \lambda a.\ \lambda b.\ a$
  - $\texttt{I} \to \lambda a.\ a$
- $\lambda \to \texttt{SKI}$

Combinator Calculus

- Equivalent to λ-calculus, but without variable binding
- $\texttt{SKI} \to \lambda$
  - $\texttt{S} \to \lambda a.\ \lambda b.\ \lambda c.\ a\ c\ (b\ c)$
  - $\texttt{K} \to \lambda a.\ \lambda b.\ a$
  - $\texttt{I} \to \lambda a.\ a$
- $\lambda \to \texttt{SKI}$
  - ???

Combinator Calculus

- Equivalent to λ-calculus, but without variable binding
- $\mathtt{SKI} \to \lambda$
  - $\mathtt{S} \to \lambda a.\ \lambda b.\ \lambda c.\ a\ c\ (b\ c)$
  - $\mathtt{K} \to \lambda a.\ \lambda b.\ a$
  - $\mathtt{I} \to \lambda a.\ a$
- $\lambda \to \mathtt{SKI}$
  - ???
  - Used to compile efficient λ programs

Combinator Calculus

- Equivalent to λ-calculus, but without variable binding
- $\mathtt{SKI} \to \lambda$
  - $\mathtt{S} \to \lambda a.\ \lambda b.\ \lambda c.\ a\ c\ (b\ c)$
  - $\mathtt{K} \to \lambda a.\ \lambda b.\ a$
  - $\mathtt{I} \to \lambda a.\ a$
- $\lambda \to \mathtt{SKI}$
  - ???
  - Used to compile efficient λ programs
  - SKI is like ‘machine code’ for λ

Practical λ?

Practical λ?

Is λ-calculus only a theoretical tool?

Practical λ?

Is λ-calculus only a theoretical tool?

- No!

Practical λ?

Is λ-calculus only a theoretical tool?

- No!
- Anonymous functions

Practical λ?

Is λ-calculus only a theoretical tool?

- No!
- Anonymous functions
- Functional programming languages
  - Haskell
  - PureScipt
  - Elm