λas.js

A Flock *of* Functions

COMBINATORS, LAMBDA CALCULUS,
& CHURCH ENCODINGS *in* JAVASCRIPT

Gabriel Lebec

FULLSTACK ACADEMY of CODE

github.com/glebec/lambda-talk

glebec

glebec

glebec

glebec

g_lebec

😭

$$\lambda a.a$$

IDENTITY

λ $\quad \mathrm{I} := \lambda a.\, a$

JS

```
I = a => a
```

λ $\quad \mathrm{I} := \lambda a.a$

JS `I = a => a`

λ $\quad \mathrm{I} := \lambda a.\, a$

JS `I = a => a`

λ $\quad \mathrm{I} := \lambda a.a$

JS `I = a => a`

λ $\quad \mathrm{I} := \lambda a.\, a$

JS `I = a => a`

$$\mathrm{I} := \lambda a.a$$

```
I = a => a
```

λ

$$\mathrm{I}\ x = ?$$

JS

```
I(x) === ?
```

$$\mathrm{I} := \lambda a.a$$

```
I = a => a
```

λ

$$\mathrm{I}\ x = x$$

JS

```
I(x) === x
```

$$\mathrm{I} := \lambda a.a$$

```
I = a => a
```

λ

$$\mathrm{I}\ \mathrm{I} = ?$$

JS

```
I(I) === ?
```

$$\mathrm{I} := \lambda a.a$$

```
I = a => a
```

λ

$$\mathrm{I}\ \mathrm{I} = \mathrm{I}$$

JS

```
I(I) === I
```

```
id 5 == 5
```

λ?

FUNCTION SIGNIFIER $\longrightarrow \lambda a.a$

PARAMETER VARIABLE

FUNCTION SIGNIFIER

$$\lambda a.\, a$$

PARAMETER VARIABLE

FUNCTION SIGNIFIER

$$\lambda a. a$$

RETURN EXPRESSION

PARAMETER VARIABLE

FUNCTION SIGNIFIER

$$\underbrace{\lambda a.\, a}_{\text{LAMBDA ABSTRACTION}}$$

RETURN EXPRESSION

λ-CALCULUS SYNTAX

expression	::=	variable	*identifier*
	|	expression expression	*application*
	|	λ variable . expression	*abstraction*
	|	(expression)	*grouping*

$\lambda \rightarrow$ JS

VARIABLES

x	`x`
(a)	`(a)`

APPLICATIONS

$f\ a$	`f(a)`
$f\ a\ b$	`f(a)(b)`
$(f\ a)\ b$	`(f(a))(b)`
$f\ (a\ b)$	`f(a(b))`

ABSTRACTIONS

$\lambda a.b$	`a => b`
$\lambda a.b\ x$	`a => b(x)`
$\lambda a.(b\ x)$	`a => (b(x))`
$(\lambda a.b)\ x$	`(a => b)(x)`
$\lambda a.\lambda b.a$	`a => b => a`
$\lambda a.(\lambda b.a)$	`a => (b => a)`

β-REDUCTION

$$((\lambda a.a)\lambda b.\lambda c.b)(x)\lambda e.f$$

β-REDUCTION

$$((\lambda a.a)\underline{\lambda b.\lambda c.b})(x)\lambda e.f$$

β-REDUCTION

$$((\lambda a.a)\underline{\lambda b.\lambda c.b})(x)\lambda e.f$$

β-REDUCTION

$$((\lambda a.a)\underline{\lambda b.\lambda c.b})(x)\lambda e.f$$

$$= \quad (\lambda b.\lambda c.b) \quad (x)\lambda e.f$$

β-REDUCTION

$$((\lambda a.a)\lambda b.\lambda c.b)(x)\lambda e.f$$

$$= \quad (\lambda b.\lambda c.b) \quad \underline{(x)}\lambda e.f$$

β-REDUCTION

$$((\lambda a.a)\lambda b.\lambda c.b)(x)\lambda e.f$$

$$= \quad (\lambda b.\lambda c.b) \quad \underline{(x)}\lambda e.f$$

β-REDUCTION

$$((\lambda a.a)\lambda b.\lambda c.b)(x)\lambda e.f$$

$$= \quad (\lambda b.\lambda c.b) \quad \underline{(x)}\lambda e.f$$

$$= \quad (\lambda c.x) \quad \lambda e.f$$

β-REDUCTION

$$((\lambda a.a)\lambda b.\lambda c.b)(x)\lambda e.f$$

$$= \quad (\lambda b.\lambda c.b) \quad (x)\lambda e.f$$

$$= \quad (\lambda c.x) \quad \underline{\lambda e.f}$$

β-REDUCTION

$$((\lambda a.a)\lambda b.\lambda c.b)(x)\lambda e.f$$

$$= \quad (\lambda b.\lambda c.b) \quad (x)\lambda e.f$$

$$= \quad (\lambda c.x) \quad \underline{\lambda e.f}$$

β-REDUCTION

$$((\lambda a.a)\lambda b.\lambda c.b)(x)\lambda e.f$$

$$= \quad (\lambda b.\lambda c.b) \quad (x)\lambda e.f$$

$$= \quad (\lambda c.x) \qquad \underline{\lambda e.f}$$

$$= \quad x$$

β-REDUCTION*

$$
\begin{aligned}
& ((\lambda a.a)\lambda b.\lambda c.b)(x)\lambda e.f \\
= \quad & (\lambda b.\lambda c.b) \quad (x)\lambda e.f \\
= \quad & (\lambda c.x) \quad \lambda e.f \\
= \quad & x
\end{aligned}
$$

β-NORMAL FORM

**not covered: evaluation order, variable collision avoidance*

$$\lambda f.ff$$

MOCKINGBIRD

λ

$$\mathrm{M} := \lambda f.ff$$

JS

```
M = f => f(f)
```

λ

$$\mathrm{M} := \lambda f.ff$$

JS

```
M = f => f(f)
```

λ

$$\mathrm{M} := \lambda f.ff$$

JS

```
M = f => f(f)
```

λ

$$\mathrm{M} := \lambda f.ff$$

JS

```
M = f => f(f)
```

$$\mathrm{M} := \lambda f.ff$$

λ

$$\mathrm{M}\ \mathrm{I} = ?$$

JS

```
M(I) === ?
```

$$\mathrm{M} := \lambda f.ff$$

λ

$$\mathrm{M}\ \mathrm{I} = \mathrm{I}\ \mathrm{I} = ?$$

JS

```
M(I) === I(I)
        && I(I) === ?
```

$$\mathrm{M} := \lambda f.ff$$

λ

$$\mathrm{M}\ \mathrm{I} = \mathrm{I}\ \mathrm{I} = \mathrm{I}$$

JS

```
M(I) === I(I)
        && I(I) === I
```

$$\mathrm{M} := \lambda f.ff$$

λ

$$\mathrm{M}\ \mathrm{M} = ?$$

JS

```
M(M) === ?
```

$$\mathrm{M} := \lambda f.ff$$

λ

$$\mathrm{M}\ \mathrm{M} = \mathrm{M}\ \mathrm{M} = ?$$

JS

```
M(M) === M(M) === ?
```

$$\mathrm{M} := \lambda f.ff$$

λ

$$\mathrm{M}\ \mathrm{M} = \mathrm{M}\ \mathrm{M} = \mathrm{M}\ \mathrm{M} = \Omega$$

JS

```
// stack overflow
M(M) === M(M) === M(M) === M(M) ===
M(M) === M(M) === M(M) === M(M) ===
M(M) === M(M) === M(M) === M(M) ===
M(M) === M(M) === M(M) === M(M) ===
M(M) === M(M) === M(M) === M(M) ===
```

$$\mathrm{M} := \lambda f.ff$$

λ

$$\omega \quad \omega = \omega \quad \omega = \omega \quad \omega = \Omega$$

JS

```
// stack overflow
M(M) === M(M) === M(M) === M(M) ===
M(M) === M(M) === M(M) === M(M) ===
M(M) === M(M) === M(M) === M(M) ===
M(M) === M(M) === M(M) === M(M) ===
M(M) === M(M) === M(M) === M(M) ===
```

ABSTRACTIONS, *again*

$$\lambda a.\lambda b.\lambda c.b$$

```
a => b => c => b
```

$$= \lambda abc.b$$

```
a => b => c => b
```

~~`(a, b, c) => b`~~

β-REDUCTION, *again*

$$((\lambda a.a)\underline{\lambda bc.b})(x)\lambda e.f$$

β-REDUCTION, *again*

$$((\lambda a.a)\underline{\lambda bc.b})(x)\lambda e.f$$

$$= \quad (\lambda bc.b) \quad (x)\lambda e.f$$

β-REDUCTION, *again*

$$((\lambda a.a)\lambda bc.b)(x)\lambda e.f$$

$$= \quad (\lambda bc.b) \quad \underline{(x)}\lambda e.f$$

β-REDUCTION, *again*

$$((\lambda a.a)\lambda bc.b)(x)\lambda e.f$$

$$= \quad (\lambda bc.b) \quad \underline{(x)}\lambda e.f$$

$$= \quad (\lambda c.x) \quad \lambda e.f$$

β-REDUCTION, *again*

$$((\lambda a.a)\lambda bc.b)(x)\lambda e.f$$

$$= \quad (\lambda bc.b) \quad (x)\lambda e.f$$

$$= \quad (\lambda c.x) \quad \lambda e.f$$

β-REDUCTION, *again*

$$
\begin{aligned}
& ((\lambda a.a)\lambda bc.b)(x)\lambda e.f \\
= \quad & (\lambda bc.b) \quad (x)\lambda e.f \\
= \quad & (\lambda c.x) \qquad \lambda e.f \\
= \quad & x
\end{aligned}
$$

β-NORMAL FORM

$\lambda ab.a$

KESTREL

λ

$$\mathrm{K} := \lambda ab.a$$
$$= \lambda a.\lambda b.a$$

JS

```
K = a => b => a
```

λ

$$\mathrm{K} := \lambda \underline{a} b.a$$

$$= \lambda \underline{a}.\lambda b.a$$

JS

```
K = a => b => a
```

λ

$$\mathrm{K} := \lambda a\underline{b}.a$$

$$= \lambda a.\lambda\underline{b}.a$$

JS

```
K = a => b => a
```

λ

$$\mathrm{K} := \lambda ab.\underline{a}$$
$$= \lambda a.\lambda b.\underline{a}$$

JS

```
K = a => b => a
```

$$\mathrm{K} := \lambda ab.a$$

```
K = a => b => a
```

λ

$$\mathrm{K\ M\ I} = ?$$

JS

```
K(M)(I) === ?
```

$$\mathrm{K} := \lambda ab.a$$

```
K = a => b => a
```

λ

$$\mathrm{K\ M\ I} = \mathrm{M}$$

JS

```
K(M)(I) === M
```

$$\mathrm{K} := \lambda ab.a$$

```
K = a => b => a
```

λ

$$\mathrm{K\ M\ I} = \mathrm{M}$$

$$\mathrm{K\ I\ M} = ?$$

JS

```
K(M)(I) === M
K(I)(M) === ?
```

$$\mathrm{K} := \lambda ab.a$$

```
K = a => b => a
```

λ

$$\mathrm{K\ M\ I} = \mathrm{M}$$

$$\mathrm{K\ I\ M} = \mathrm{I}$$

JS

```
K(M)(I) === M
K(I)(M) === I
```

```
const 7 2 == 7
```

$$\mathrm{K} := \lambda ab.a$$

```
K = a => b => a
```

λ

$$\mathrm{K}\ \mathrm{I}\ x \quad = \mathrm{I}$$

JS

```
K(I)(x)     === I
```

$$\mathrm{K} := \lambda ab.a$$

```
K = a => b => a
```

λ

$$\mathrm{K}\ \mathrm{I}\ x\ y = \mathrm{I}\ y = ?$$

JS

```
K(I)(x)(y) === I(y)
              && I(y) === ?
```

$$\mathrm{K} := \lambda ab.a$$

```
K = a => b => a
```

λ

$$\mathrm{K}\ \mathrm{I}\ x\ y = \mathrm{I}\ y = y$$

JS

```
K(I)(x)(y) === I(y)
              && I(y) === y
```

$$\mathrm{K} := \lambda ab.a$$

```
K = a => b => a
```

λ

$$\mathrm{K}\ \mathrm{I}\ x\ y = \mathrm{I}\ y = y$$

JS

```
K(I)(x)(y) === I(y)
              && I(y) === y
```

$$\mathrm{K} := \lambda ab.a$$

```
K = a => b => a
```

λ

$$\underline{\mathrm{K\ I}}\ x\ y = \mathrm{I}\ y = y$$

JS

```
K(I)(x)(y) === I(y)
              && I(y) === y
```

$\lambda ab.b$

KITE

λ

$$\mathrm{KI} := \lambda ab.b$$

$$= \mathrm{K}\ \mathrm{I}$$

JS

```
KI = a => b => b
KI = K(I)
```

$$\mathrm{KI} := \lambda ab.b$$

```
KI = a => b => b
```

λ

$$\mathrm{KI}\ \mathrm{M}\ \mathrm{K} = ?$$

JS

```
KI(M)(K) === ?
```

$$\mathrm{KI} := \lambda ab.b$$

```
KI = a => b => b
```

λ

$$\mathrm{KI\ M\ K} = \mathrm{K}$$

JS

```
KI(M)(K) === K
```

$$\mathrm{KI} := \lambda ab.b$$

```
KI = a => b => b
```

λ

$$\mathrm{KI\ M\ K} = \mathrm{K}$$

$$\mathrm{KI\ K\ M} = ?$$

JS

```
KI(M)(K) === K
KI(K)(M) === ?
```

$$\mathrm{KI} := \lambda ab.b$$

```
KI = a => b => b
```

λ

$$\mathrm{KI\ M\ K} = \mathrm{K}$$

$$\mathrm{KI\ K\ M} = \mathrm{M}$$

JS

```
KI(M)(K) === K
KI(K)(M) === M
```

?

SCHÖNFINKEL

CURRY

SMULLYAN

Identitätsfunktion	I	Idiot → *Ibis?*
Konstante Funktion	K	Kestrel
verSchmelzungsfunktion	S	Starling
verTauschungsfunktion	C	Cardinal
Zusammensetzungsf.	B	Bluebird

Mathematics Teacher Enjoys Bird Watching

University Park—Looking for a hobby? One that might give you the chance to rub elbows with royalty, get you out in the fresh air, and provide some cliff-hanging adventures, all at little or no expense? Then try bird watching.

That's the suggestion of Dr. Haskell B. Curry, Evan Pugh research professor of mathematics at the Pennsylvania State University.

Dr. Curry, founder of the State College Bird Watchers Club, has been observing the antics of the feathered creatures for more than 50 years now and is probably one of the longest-term bird watchers in the state.

And he has a stack of records nearly three feet tall to prove it, some dating back to 1916, reporting the sighting of nearly 150,000 separate birds.

But the true adventure story of decades of trampling through woods in search of birds isn't shown in the records.

Recalls Oslo Visit

For instance, Dr. Curry remembers the time he visited Oslo, Norway, in 1928 and decided to do some bird watching. In persuit of his quarry, Dr. Curry jumped over a high fence onto a sprawling estate.

"No sooner had I crossed the fence than a gardener came running toward me with two enormous dogs," Dr. Curry remembers. "I tried to explain what I was doing and then I got out of there as quickly as I could."

Later, Dr. Curry learned that he had crossed onto a royal estate. It wasn't until 10 years later, however, after coming across a picture in a newspaper, that Dr. Curry learned that the "gardener" who ejected him was King Olaf of Norway.

Then there was the time Dr. Curry and a friend were trapped in the woods by nightfall. After groping and stumbling their way, the two men came to the edge of a cliff and decided to wait for daybreak. The "cliff" at daybreak, turned out to be a four-foot deep gully.

Once while out bird watching, Dr. Curry began to sing at the top of his lungs "out of sheer joy." His song was answered by a howling dog who followed Dr. Curry through the woods for the rest of the day.

DR. HASKELL B. CURRY

The fact that Dr. Curry has had minor mishaps on several occasions hasn't dampened his enthusiasm for bird watching one bit.

"I try to get out at least once a week, very early in the morning. I've been watching birds since I was 11."

All you need for bird-watching, according to Dr. Curry, are a good set of legs, a pair of binoculars and a note book.

Dr. Curry's field of mathematics and his hobby of bird-watching have nothing in common.

"That's why I like bird-watching," says Dr. Curry. "It gives me a chance to get outdoors, to use my eyes and to deal with concrete facts instead of with theories."

This fall, Dr. Curry returned to the university after a six-month leave of absence during which time he visited 25 countries to give lectures on mathematics at various universities throughout the world.

In his spare time during the trip, Dr. Curry went bird-watching. He spotted over 600 species of birds during the entire globe-encircling trip, 270 of them in Australia, which Dr. Curry considers to be the most interesting bird-watching country.

The Formalization of

PEANO

RUSSELL

VON NEUMANN

GÖDEL

KLEENE

ROSSER

FREGE

SCHÖNFINKEL

CURRY

CHURCH

TURING

Mathematical Logic

Formal Notation for Functions

1889
PEANO

RUSSELL

VON NEUMANN

GÖDEL

KLEENE

ROSSER

FREGE

SCHÖNFINKEL

CURRY

CHURCH

TURING

Peano Arithmetic

Axiomatic Logic · Fn Notation

PEANO

RUSSELL

VON NEUMANN

GÖDEL

KLEENE

ROSSER

FREGE
1891

SCHÖNFINKEL

CURRY

CHURCH

TURING

Functions as Graphs · Currying

Principia Mathematica

1910

PEANO

RUSSELL

VON NEUMANN

GÖDEL

KLEENE

ROSSER

FREGE

SCHÖNFINKEL

CURRY

CHURCH

TURING

Russell's Paradox · Fn Notation

Combinatory Logic

PEANO

RUSSELL

VON NEUMANN

GÖDEL

KLEENE

ROSSER

FREGE

SCHÖNFINKEL
1920

CURRY

CHURCH

TURING

Combinators · Currying

Functional System of Set Theory

1925

PEANO

RUSSELL

VON NEUMANN

GÖDEL

KLEENE

ROSSER

FREGE

SCHÖNFINKEL

CURRY

CHURCH

TURING

(overlapped with Combinatory Logic)

Combinatory Logic (again)

PEANO

RUSSELL

VON NEUMANN

GÖDEL

KLEENE

ROSSER

FREGE

SCHÖNFINKEL

CURRY
1926

CHURCH

TURING

Combinators · many contributions

DISCOVERS SCHÖNFINKEL

PEANO

RUSSELL

VON NEUMANN

GÖDEL

KLEENE

ROSSER

FREGE

SCHÖNFINKEL

CURRY
1927

CHURCH

TURING

"This paper anticipates much of what I have done."

Incompleteness Theorems

1931

PEANO RUSSELL VON NEUMANN GÖDEL KLEENE ROSSER

FREGE SCHÖNFINKEL CURRY CHURCH TURING

Ending the Search for Sufficient Axioms

λ-Calculus

PEANO

RUSSELL

VON NEUMANN

GÖDEL

KLEENE

ROSSER

FREGE

SCHÖNFINKEL

CURRY

CHURCH
1932

TURING

An Effective Model of Computation

INCONSISTENCY OF SPECIALIZED λ

1931–1936

PEANO

RUSSELL

VON NEUMANN

GÖDEL

KLEENE

ROSSER

FREGE

SCHÖNFINKEL

CURRY

CHURCH

TURING

CONSISTENCY OF PURE λ

Solves the Decision Problem

PEANO

RUSSELL

VON NEUMANN

GÖDEL

KLEENE

ROSSER

FREGE

SCHÖNFINKEL

CURRY

CHURCH
1936

TURING

via the λ-Calculus

Solves the Decision Problem

PEANO

RUSSELL

VON NEUMANN

GÖDEL

KLEENE

ROSSER

FREGE

SCHÖNFINKEL

CURRY

CHURCH

TURING
1936

via the Turing Machine

Establishes the Church-Turing Thesis

Peano

Russell

von Neumann

Gödel

Kleene

Rosser

Frege

Schönfinkel

Curry

Church

Turing
1936

λ-Calculus $\Leftrightarrow$ Turing Machine

Obtains PhD under Church

Peano

Russell

von Neumann

Gödel

Kleene

Rosser

Frege

Schönfinkel

Curry

Church

Turing

1936–1938

Publishes 1st Fixed-Point Combinator

COMBINATORS

functions with no free variables

$\lambda b.b$	combinator
$\lambda b.a$	not a combinator
$\lambda ab.a$	combinator
$\lambda a.ab$	not a combinator
$\lambda abc.c(\lambda e.b)$	combinator

COMBINATORS

Sym.	Bird	λ-Calculus	Use	Haskell
I	Idiot	$\lambda a.a$	identity	`id`
M	Mockingbird	$\lambda f.ff$	self-application	(cannot define)
K	Kestrel	$\lambda ab.a$	first, const	`const`
KI	Kite	$\lambda ab.b = \mathrm{KI}$	second	`const id`

$$\lambda fab.fba$$

CARDINAL

λ

$$\mathrm{C} := \lambda fab.fba$$

JS

```
C = f => a => b => f(b)(a)
```

$$\mathrm{C} := \lambda fab.fba$$

```
C = f => a => b => f(b)(a)
```

λ

$$\mathrm{C\ K\ I\ M} = ?$$

JS

```
C(K)(I)(M) === ?
```

$$\mathrm{C} := \lambda fab.fba$$

```
C = f => a => b => f(b)(a)
```

λ

$$\mathrm{C\ K\ I\ M} = \mathrm{M}$$

JS

```
C(K)(I)(M) === M
```

$$\mathrm{C} := \lambda fab.fba$$

```
C = f => a => b => f(b)(a)
```

λ

$$\underline{\mathrm{C\ K}}\ \mathrm{I\ M} = \mathrm{M}$$

JS

```
C(K)(I)(M) === M
```

$$\mathrm{C} := \lambda fab.fba$$

```
C = f => a => b => f(b)(a)
```

λ

$$\underline{\mathrm{KI}}\ \mathrm{I}\ \mathrm{M} = \mathrm{M}$$

JS

```
KI(I)(M) === M
```

COMBINATORS

Sym.	Bird	λ-Calculus	Use	Haskell
I	Idiot	$\lambda a.a$	identity	`id`
M	Mockingbird	$\lambda f.ff$	self-application	(cannot define)
K	Kestrel	$\lambda ab.a$	first, const	`const`
KI	Kite	$\lambda ab.b = \mathrm{KI}$	second	`const id`
C	Cardinal	$\lambda fab.fba$	reverse arguments	`flip`

COMBINATORS

Sym.	Bird	λ-Calculus	Use	Haskell
I	Idiot	$\lambda a.a$	identity	`id`
M	Mockingbird	$\lambda f.ff$	self-application	(cannot define)
K	Kestrel	$\lambda ab.a$	first, const	`const`
KI	Kite	$\lambda ab.b = \mathrm{KI} = \mathrm{CK}$	second	`const id`
C	Cardinal	$\lambda fab.fba$	reverse arguments	`flip`

```
flip const 1 8 == 8
```

λ-CALCULUS

abstract symbol rewriting

functional computation

$$(\lambda f.ff)\lambda a.a$$

TURING MACHINE

hypothetical device

state-based computation

real computers

machine code

assembly languages

higher-level machine-centric languages

higher-level abstract stateful languages

purely functional programming languages

$$\lambda \longleftrightarrow \boxed{\text{TM}}$$

EVERYTHING

CAN BE

FUNCTIONS

• • •

TRUE FALSE NOT AND OR BEQ

λ

JS

```
const result = bool ? exp1 : exp2
```

λ

JS

```
const result = bool ? exp1 : exp2
// true
```

λ

JS

```
const result = bool ? exp1 : exp2
// false
```

λ $\text{result} := ?$

JS

```
const result = bool ? exp1 : exp2
```

λ

$$\text{result} := \text{bool} \; ? \; \text{exp1} : \text{exp2}$$

JS

```
const result = bool ? exp1 : exp2
```

λ $\quad$ $\text{result} := \text{bool} \quad \text{exp1} \quad \text{exp2}$

JS

```
const result = bool ? exp1 : exp2
```

λ $\quad$ $\text{result} := \text{func} \quad \text{exp1} \quad \text{exp2}$

JS

```
const result = bool  (exp1) (exp2)
```

λ

$$\text{result} := \text{func} \quad \text{exp1} \quad \text{exp2}$$

JS

```
const result = bool  (exp1) (exp2)
                // true
```

λ

$$\text{result} := \text{func} \quad \text{exp1} \quad \text{exp2}$$

JS

```
const result = bool  (exp1) (exp2)
                // false
```

λ

$$\text{result} := \text{func} \quad \text{exp1} \quad \text{exp2}$$

TRUE

FALSE

JS

```
const result = bool  (exp1) (exp2)
```

λ

$$\text{result} := \text{func} \quad \text{exp1} \quad \text{exp2}$$

K

KI

JS

```
const result = bool  (exp1) (exp2)
```

CHURCH ENCODINGS: BOOLEANS

λ

$$\textsc{True} := \mathrm{K}$$

$$\textsc{False} := \mathrm{KI} = \mathrm{C}\ \mathrm{K}$$

JS

```
const T = K
const F = KI
```

λ $!p$

JS !p

λ $\textsc{Not}\ p$

JS `not(p)`

λ

$$\textsc{Not}\ p \begin{cases} \mathrm{F} \\ \mathrm{T} \end{cases}$$

JS

```
not(p) { F
       { T
```

λ

NOT T → F
T

JS

`not(T)` → `F`
`T`

λ

NOT F → F, T

JS

`not(F)` → F, T

λ

$$p \quad \mathrm{F} \quad \mathrm{T}$$

JS

```
p (F)(T)
```

λ

K KI

T F T

JS

K KI

T (F) (T)

λ

KI K

F F T

JS

KI K

F (F) (T)

λ

$$\textsc{Not} := \lambda p.p\mathrm{FT}$$

JS

```
const not = p => p(F)(T)
```

CHURCH ENCODINGS: BOOLEANS

Sym.	Name	λ-Calculus	Use
T	True	$\lambda ab.a = \mathrm{K}$	encoding for true
F	False	$\lambda ab.b = \mathrm{KI} = \mathrm{CK}$	encoding for false
	Not	$\lambda p.p\mathrm{FT}$	negation

λ

$$\textsc{Not}\ \mathrm{T} = \mathrm{F}$$
$$\textsc{Not}\ \mathrm{F} = \mathrm{T}$$

JS

```
not(T) === F
not(F) === T
```

λ

$$\lambda ab.a \quad \lambda ba.a$$

$$\textsc{Not}\ \mathrm{K} = \mathrm{KI}$$

$$\textsc{Not}\ (\mathrm{KI}) = \mathrm{K}$$

$$\lambda ba.a \quad \lambda ab.a$$

JS

```
not(K) === KI
not(KI) === K
```

λ

$$\mathrm{C}\ \mathrm{K} = \mathrm{KI}$$

$$\mathrm{C}\ (\mathrm{KI}) = \mathrm{K}$$

JS

```
C(K) (chirp)(tweet) === tweet
C(KI)(chirp)(tweet) === chirp
```

λ

$$\mathrm{C}\ \mathrm{T} = \mathrm{F}$$
$$\mathrm{C}\ \mathrm{F} = \mathrm{T}$$

JS

```
C(T) (chirp)(tweet) === tweet
C(F) (chirp)(tweet) === chirp
```

CHURCH ENCODINGS: BOOLEANS

Sym.	Name	λ-Calculus	Use
T	True	$\lambda ab.a = \mathrm{K}$	encoding for true
F	False	$\lambda ab.b = \mathrm{KI} = \mathrm{CK}$	encoding for false
	Not	$\lambda p.p\mathrm{FT}$	negation

CHURCH ENCODINGS: BOOLEANS

Sym.	Name	λ-Calculus	Use
T	True	$\lambda ab.a = \mathrm{K}$	encoding for true
F	False	$\lambda ab.b = \mathrm{KI} = \mathrm{CK}$	encoding for false
	Not	$\lambda p.p\mathrm{FT}$ or C	negation

λ $\quad \textsc{And} := ?$

JS

```
const and = ?
```

λ

$$\textsc{And} := \lambda pq.?$$

JS

```
const and = p => q => ?
```

λ

$$\textsc{And} := \lambda pq.p?¿$$

JS

```
const and = p => q => p(?)(¿)
```

λ

$$\textsc{And} := \lambda pq.p^{\overset{\mathrm{F}}{?}}\,¿$$

JS

```
const and = p => q => p(?)(¿)
```

F

λ

$$\textsc{And} := \lambda pq.p?\mathrm{F}$$

JS

```
const and = p => q => p(?)(F)
```

λ

$$\textsc{And} := \lambda pq.p\overset{\mathrm{T}}{?}\mathrm{F}$$

JS

```
const and = p => q => p(?)(F)
```

T

λ

$$\textsc{And} := \lambda pq.pq\mathrm{F}$$

JS

```
const and = p => q => p(q)(F)
```

$$\lambda pq.pq\mathrm{F}$$

$$\lambda pq.p \begin{cases} \to q \\ \to \mathrm{F} \end{cases}$$

$$\lambda pq.p$$

$$q$$

$$p$$

$$\lambda pq.pqp$$

CHURCH ENCODINGS: BOOLEANS

Sym.	Name	λ-Calculus	Use
T	True	$\lambda ab.a = \mathrm{K}$	encoding for true
F	False	$\lambda ab.b = \mathrm{KI} = \mathrm{CK}$	encoding for false
	Not	$\lambda p.p\mathrm{FT}$ or C	negation
	And	$\lambda pq.pq\mathrm{F}$ or $\lambda pq.pqp$	conjunction

λ $\textsc{Or} := ?$

JS `const or = ?`

λ

$$\textsc{Or} := \lambda pq. \ldots$$

JS

```
const or = p => q => …
```

λ

$$\textsc{Or} := \lambda pq.p?¿$$

JS

```
const or = p => q => p(?)(¿)
```

λ

$$\textsc{Or} := \lambda pq.p\mathrm{T}¿$$

JS

```
const or = p => q => p(T)(¿)
```

λ

$$\textsc{Or} := \lambda pq.p\mathrm{T}q$$

JS

```
const or = p => q => p(T)(q)
```

λ

$$\textsc{Or} := \lambda pq.ppq$$

JS

```
const or = p => q => p(p)(q)
```

CHURCH ENCODINGS: BOOLEANS

Sym.	Name	λ-Calculus	Use
T	True	$\lambda ab.a = \mathrm{K}$	encoding for true
F	False	$\lambda ab.b = \mathrm{KI} = \mathrm{CK}$	encoding for false
	Not	$\lambda p.p\mathrm{FT}$ or C	negation
	And	$\lambda pq.pq\mathrm{F}$ or $\lambda pq.pqp$	conjunction
	Or	$\lambda pq.p\mathrm{T}q$ or $\lambda pq.ppq$	disjunction

$$\lambda pq.ppq$$

$$(\lambda pq.ppq)xy = ?$$

$$(\lambda pq.ppq)xy = xxy$$

$$(\lambda pq.ppq)xy = xxy$$

$$(\quad ? \quad)xy = xxy$$

$$(\lambda pq.ppq)xy = xxy$$

$$(\quad ? \quad)xy = xxy$$

$$(\lambda pq.ppq)xy = xxy$$

$$\mathrm{M}\,xy = xxy$$

$$(\lambda pq.ppq)xy = xxy$$

$$\mathrm{M}\,xy = xxy$$

CHURCH ENCODINGS: BOOLEANS

Sym.	Name	λ-Calculus	Use
T	True	$\lambda ab.a = \mathrm{K}$	encoding for true
F	False	$\lambda ab.b = \mathrm{KI} = \mathrm{CK}$	encoding for false
	Not	$\lambda p.p\mathrm{FT}$ or C	negation
	And	$\lambda pq.pq\mathrm{F}$ or $\lambda pq.pqp$	conjunction
	Or	$\lambda pq.p\mathrm{T}q$ or $\lambda pq.ppq$	disjunction

CHURCH ENCODINGS: BOOLEANS

Sym.	Name	λ-Calculus	Use
T	True	$\lambda ab.a = \mathrm{K}$	encoding for true
F	False	$\lambda ab.b = \mathrm{KI} = \mathrm{CK}$	encoding for false
	Not	$\lambda p.p\mathrm{FT}$ or C	negation
	And	$\lambda pq.pq\mathrm{F}$ or $\lambda pq.pqp$	conjunction
	Or	$\lambda pq.p\mathrm{T}q$ or $\lambda pq.ppq = \mathrm{M}^*$	disjunction

$$\lambda pq.p\,(\,q\,\mathrm{T}\,\mathrm{F}\,)(\,q\,\mathrm{F}\,\mathrm{T}\,)$$

```
p => q => p(q(T)(F))(q(F)(T))
```

$$\lambda pq.p \begin{cases} (q\ \mathrm{T}\ \mathrm{F}) \\ (q\ \mathrm{F}\ \mathrm{T}) \end{cases}$$

$$\lambda pq.p \begin{cases} (q\ \mathrm{T}\ \mathrm{F}) \\ (q\ \mathrm{F}\ \mathrm{T}) \end{cases}$$

$\lambda pq.p$
$(q$ T $)$
$(q$ F $)$

$\lambda pq.p$
$(q$ T F $)$
$(q$ F T $)$

$\lambda pq.p$
$(q$ T F $)$
$(q$ F T $)$

$$\lambda pq.p$$

$$(q\ \mathrm{T}\ \mathrm{F})$$

$$(q\ \mathrm{F}\ \mathrm{T})$$

BOOLEAN EQUALITY

$$\lambda pq.p \begin{cases} (q\ \mathrm{T}\ \mathrm{F}) \\ (q\ \mathrm{F}\ \mathrm{T}) \end{cases}$$

$$\lambda pq.p \begin{cases} q \\ (q \begin{cases} \mathrm{F} \\ \mathrm{T} \end{cases}) \end{cases}$$

$$\lambda pq.p \begin{cases} \to q \\ \to (\textsc{Not}\ q) \end{cases}$$

$$\lambda pq.p\ q\ (\textsc{Not}\ q)$$

```
p => q => p(q)(not(q))
```

CHURCH ENCODINGS: BOOLEANS

Sym.	Name	λ-Calculus	Use
T	TRUE	$\lambda ab.a = \mathrm{K}$	encoding for true
F	FALSE	$\lambda ab.b = \mathrm{KI} = \mathrm{CK}$	encoding for false
	NOT	$\lambda p.p\mathrm{FT}$ or C	negation
	AND	$\lambda pq.pq\mathrm{F}$ or $\lambda pq.pqp$	conjunction
	OR	$\lambda pq.p\mathrm{T}q$ or $\lambda pq.ppq = \mathrm{M}^*$	disjunction
	BEQ	$\lambda pq.p\ q\ (\mathrm{NOT}\ q)$	equality

(ONE OF) DE MORGAN'S LAWS

$$\neg(P \wedge Q) = (\neg P) \vee (\neg Q)$$

$$\textsc{Beq}\ (\textsc{Not}\ (\textsc{And}\ p\ q))\ (\textsc{Or}\ (\textsc{Not}\ p)\ (\textsc{Not}\ q))$$

```
!(p && q) === (!p) || (!q)
```

$$\textsc{Beq}\ (\textsc{Not}\ (\textsc{And}\ p\ q))\ (\textsc{Or}\ (\textsc{Not}\ p)\ (\textsc{Not}\ q))$$

$$(\lambda xy.x\ y\ ((\lambda fab.fba)\ y))$$

$$((\lambda fab.fba)\ ((\lambda xy.xyx)\ p\ q))$$

$$((\lambda f.ff)\ ((\lambda fab.fba)\ p)\ ((\lambda fab.fba)\ q))$$

$$\textsc{Beq}\ (\textsc{Not}\ (\textsc{And}\ p\ q))\ (\textsc{Or}\ (\textsc{Not}\ p)\ (\textsc{Not}\ q))$$

$$(\lambda xy.x\ y\ ((\lambda fab.fba)\ y))$$

$$((\lambda fab.fba)\ ((\lambda xy.xyx)\ p\ q))$$

$$((\lambda f.ff)\ ((\lambda fab.fba)\ p)\ ((\lambda fab.fba)\ q))$$

$$\textsc{Beq}\ (\textsc{Not}\ (\textsc{And}\ p\ q))\ (\textsc{Or}\ (\textsc{Not}\ p)\ (\textsc{Not}\ q))$$

$$(\lambda xy.x\ y\ ((\lambda fab.fba)\ y))$$

$$((\lambda fab.fba)\ ((\lambda xy.xyx)\ p\ q))$$

$$((\lambda f.ff)\ ((\lambda fab.fba)\ p)\ ((\lambda fab.fba)\ q))$$

$$\textsc{Beq}\ (\textsc{Not}\ (\textsc{And}\ p\ q))\ (\textsc{Or}\ (\textsc{Not}\ p)\ (\textsc{Not}\ q))$$

$$(\lambda xy.x\ y\ ((\lambda fab.fba)\ y))$$

$$((\lambda fab.fba)\ ((\lambda xy.xyx)\ p\ q))$$

$$((\lambda f.ff)\ ((\lambda fab.fba)\ p)\ ((\lambda fab.fba)\ q))$$

$$\textsc{Beq}\ (\textsc{Not}\ (\textsc{And}\ p\ q))\ (\textsc{Or}\ (\textsc{Not}\ p)\ (\textsc{Not}\ q))$$

$$(\lambda xy.x\ y\ ((\lambda fab.fba)\ y))$$

$$((\lambda fab.fba)\ ((\lambda xy.xyx)\ p\ q))$$

$$((\lambda f.ff)\ ((\lambda fab.fba)\ p)\ ((\lambda fab.fba)\ q))$$

$$\textsc{Beq}\ (\textsc{Not}\ (\textsc{And}\ p\ q))\ (\textsc{Or}\ (\textsc{Not}\ p)\ (\textsc{Not}\ q))$$

$$(\lambda xy.x\ y\ ((\lambda fab.fba)\ y))$$

$$((\lambda fab.fba)\ ((\lambda xy.xyx)\ p\ q))$$

$$((\lambda f.ff)\ ((\lambda fab.fba)\ p)\ ((\lambda fab.fba)\ q))$$

$$\textsc{Beq}\ (\textsc{Not}\ (\textsc{And}\ p\ q))\ (\textsc{Or}\ (\textsc{Not}\ p)\ (\textsc{Not}\ q))$$

$$(\lambda xy.x\ y\ ((\lambda fab.fba)\ y))$$

$$((\lambda fab.fba)\ ((\lambda xy.xyx)\ p\ q))$$

$$((\lambda f.ff)\ ((\lambda fab.fba)\ p)\ ((\lambda fab.fba)\ q))$$

• • •

ZERO ONE TWO THREE SUCC

• • •

ZERO ONCE TWICE THRICE SUCC

λ

$$\mathrm{N1} := \lambda fa.fa$$

$$\mathrm{N2} := \lambda fa.f(fa)$$

$$\mathrm{N3} := \lambda fa.\underline{f}(\underline{f}(\underline{f}a))$$

JS

```
n1 = f => a => f(a)
n2 = f => a => f(f(a))
n3 = f => a => f(f(f(a)))
```

$$\textsc{n1} := \lambda fa.fa$$

```
N1 = f => a => f(a)
```

λ

$$\textsc{n1}\ \textsc{Not}\ \mathrm{T} = \textsc{Not}\ \mathrm{T} = ?$$

JS

```
n1(not)(T) = not(T) = ?
```

$$\text{N1} := \lambda fa.fa$$

```
N1 = f => a => f(a)
```

λ

$$\text{N1 NOT T} = \text{NOT T} = \text{F}$$

JS

```
n1(not)(T) = not(T) = F
```

$$\textsc{n2} := \lambda f a.f(fa)$$

```
N2 = f => a => f(f(a))
```

λ

$$\textsc{n2}\ \textsc{Not}\ \mathrm{T} = \textsc{Not}\ (\textsc{Not}\ \mathrm{T}) = ?$$

JS

```
n2(not)(T) = not(not(T)) = ?
```

$$\textsc{n2} := \lambda fa.f(fa)$$

```
N2 = f => a => f(f(a))
```

λ

$$\textsc{n2}\ \textsc{Not}\ \mathrm{T} = \textsc{Not}\ (\textsc{Not}\ \mathrm{T}) = \mathrm{T}$$

JS

```
n2(not)(T) = not(not(T)) = T
```

$$\textsc{n3} := \lambda fa.f(f(fa))$$

```
N3 = f => a => f(f(f(a)))
```

λ

$$\textsc{n3}\ \textsc{Not}\ \mathrm{T} = \textsc{Not}\ (\textsc{Not}\ (\textsc{Not}\ \mathrm{T})) = \mathrm{F}$$

JS

```
n3(not)(T) = not(not(not(T))) = F
```

λ

$$\mathrm{N0} := \lambda fa.a$$

$$\mathrm{N1} := \lambda fa.fa$$

$$\mathrm{N2} := \lambda fa.f(fa)$$

$$\mathrm{N3} := \lambda fa.f(f(fa))$$

JS

```
n0 = f => a => a
n1 = f => a => f(a)
n2 = f => a => f(f(a))
n3 = f => a => f(f(f(a)))
```

$$\textsc{n0} := \lambda f a.\, a$$

```
N3 = f => a => a
```

λ

$$\textsc{n0}\ \textsc{Not}\ \mathrm{T} = ?$$

JS

```
n0(not)(T) = ?
```

$$\textsc{n0} := \lambda f a.a$$

```
N3 = f => a => a
```

λ

$$\textsc{n0}\ \textsc{Not}\ \mathrm{T} = \mathrm{T}$$

JS

```
n0(not)(T) = T
```

CHURCH ENCODINGS: NUMERALS

Sym.	Name	λ-Calculus	Use
N0	Zero	$\lambda fa.a = \mathrm{F}$	apply f no times to a
N1	Once	$\lambda fa.f\ a = \mathrm{I}^*$	apply f once to a
N2	Twice	$\lambda fa.f\ (f\ a)$	apply 2-fold f to a
N3	Thrice	$\lambda fa.f\ (f\ (f\ a))$	apply 3-fold f to a
N4	Fourfold	$\lambda fa.f\ (f\ (f\ (f\ a)))$	apply 4-fold f to a
N5	Fivefold	$\lambda fa.f\ (f\ (f\ (f\ (f\ a)))))$	apply 5-fold f to a

$$+1\ ?$$

PEANO NUMBERS

$$\textsc{Succ}\ \textsc{n1} = \textsc{n2}$$

$$\textsc{Succ}\ \textsc{n2} = \textsc{n3}$$

$$\textsc{Succ}\ (\textsc{Succ}\ \textsc{n1}) = \textsc{n3}$$

λ

$$\textsc{Succ} := \lambda n.?$$

$$\textsc{Succ}\ \textsc{n1} = \textsc{n2}$$

JS

```
succ = n => ?
```

λ

$$\textsc{Succ} := \lambda n.?$$

$$\textsc{Succ}\ \lambda fa.fa = \lambda fa.f(fa)$$

JS

```
succ = n => ?
```

λ

$$\textsc{Succ} := \lambda n\underline{fa.f(nfa)}$$

$$\textsc{Succ}\ \lambda fa.fa = \lambda fa.f(fa)$$

JS

```
succ = n => f => a => f(n(f)(a))
```

λ

$$\textsc{Succ} := \lambda nfa.f(\underline{nf}a)$$

$$\textsc{Succ}\ \lambda fa.fa = \lambda fa.f(fa)$$

JS

```
succ = n => f => a => f(n(f)(a))
```

λ

$$\textsc{Succ} := \lambda nfa.\underline{f}(nfa)$$

$$\textsc{Succ}\ \lambda fa.fa = \lambda fa.f(fa)$$

JS

```
succ = n => f => a => f(n(f)(a))
```

$$
\begin{aligned}
\textsc{Succ}\ \textsc{n2} &= (\lambda nfa.f(nfa))\ \textsc{n2} \\
&= \lambda fa.f(\textsc{n2}\ f\ a) \\
&= \lambda fa.f(f(f\ a) \\
&= \textsc{n3}
\end{aligned}
$$

CHURCH ARITHMETIC

Name	λ-Calculus	Use
SUCC	$\lambda nfa.f(nfa)$	successor of n

$$\lambda fga.f(ga)$$

BLUEBIRD

λ

$$\mathrm{B} := \lambda fga.f(ga)$$

JS

```
B = f => g => a => f(g(a))
```

$$\mathrm{B} := \lambda fga.f(\underline{ga})$$

```
B = f => g => a => f(g(a))
```

λ

$$\underline{\mathrm{B\ \textsc{Not}\ \textsc{Not}}}\ \mathrm{T} = ?$$

JS

```
B(not)(not)(T) === ?
```

$$\mathrm{B} := \lambda fga.f(\underline{ga})$$

```
B = f => g => a => f(g(a))
```

λ

$$\underline{\mathrm{B}\ \textsc{Not}\ \textsc{Not}}\ \mathrm{T} = \mathrm{T}$$

JS

```
B(not)(not)(T) === T
```

```
odd = not . even
```

COMBINATORS

Sym.	Bird	λ-Calculus	Use	Haskell
I	Idiot	$\lambda a.a$	identity	`id`
M	Mockingbird	$\lambda f.ff$	self-application	(cannot define)
K	Kestrel	$\lambda ab.a$	true, first, const	`const`
KI	Kite	$\lambda ab.b = \mathrm{KI} = \mathrm{CK}$	false, second	`const id`
C	Cardinal	$\lambda fab.fba$	reverse arguments	`flip`
B	Bluebird	$\lambda fga.f(ga)$	$1^\circ \leftarrow 1^\circ$ composition	`(.)`

λ

$$\textsc{Succ} := \lambda nfa.f(nfa)$$

JS

```
succ = n => f => a => f(n(f)(a))
```

λ

$$\textsc{Succ} := \lambda nf.\mathrm{B}f(nf)$$

JS

```
succ = n => f => B(f)(n(f))
```

CHURCH ARITHMETIC

Name	λ-Calculus	Use
SUCC	$\lambda nf.\mathrm{B}\ f\ (nf) = \lambda nfa.f(nfa)$	successor of n

• • •

ADD MULT POW

λ

$$\textsc{Add} := \lambda nk.?$$

$$\textsc{Add}\ \textsc{n3}\ \textsc{n5} = \textsc{Succ}\ (\textsc{Succ}\ (\textsc{Succ}\ \textsc{n5}))$$

JS

```
add = n => k => ?
```

λ

$$\textsc{Add} := \lambda nk.?$$

$$\textsc{Add}\ \textsc{n3}\ \textsc{n5} = (\textsc{Succ} \circ \textsc{Succ} \circ \textsc{Succ})\ \textsc{n5}$$

JS

```
add = n => k => ?
```

λ

$$\textsc{Add} := \lambda nk.?$$

$$\textsc{Add}\ \textsc{n3}\ \textsc{n5} = \textsc{n3}\ \textsc{Succ}\ \textsc{n5}$$

JS

```
add = n => k => ?
```

λ

$$\textsc{Add} := \lambda nk.n\ \textsc{Succ}\ k$$

$$\textsc{Add}\ \textsc{n3}\ \textsc{n5} = \textsc{n3}\ \textsc{Succ}\ \textsc{n5}$$

JS

```
add = n => k => n(succ)(k)
```

$$
\begin{aligned}
&\textsc{Add}\ \textsc{n3}\ \textsc{n5} = \textsc{n3}\ \textsc{Succ}\ \textsc{n5} \\
&= \textsc{Thrice}\ \textsc{Succ}\ \textsc{Fivefold} \\
&= \textsc{Succ}\ (\textsc{Succ}\ (\textsc{Succ}\ \textsc{Fivefold}))) \\
&= \textsc{Eightfold}
\end{aligned}
$$

CHURCH ARITHMETIC

Name	λ-Calculus	Use
SUCC	$\lambda nf.\mathrm{B}\ f\ (nf) = \lambda nfa.f(nfa)$	successor of n
ADD	$\lambda nk.n\ \textsc{Succ}\ k = \lambda nkf.\mathrm{B}\ (n\ f)\ (k\ f)$	addition of n and k

λ

$$\textsc{Mult} := \lambda nk.?$$

$$\textsc{Mult}\ \textsc{n2}\ \textsc{n3} = \textsc{n6}$$

JS

```
mult = n => k => ?
```

λ

$$\textsc{Mult} := \lambda nk.?$$

$$\textsc{Mult}\ \textsc{n2}\ \textsc{n3}\ f\ a = \textsc{n6}\ f\ a$$

JS

```
mult = n => k => ?
```

λ

$$\textsc{Mult} := \lambda nk.?$$

$$\textsc{Mult}\ \textsc{n2}\ \textsc{n3}\ f\ a = (f \circ f \circ f \circ f \circ f \circ f)\ a$$

JS

```
mult = n => k => ?
```

λ

$$\textsc{Mult} := \lambda nk.?$$

$$\textsc{Mult}\ \textsc{n2}\ \textsc{n3}\ f\ a = ((f \circ f \circ f) \circ (f \circ f \circ f))\ a$$

JS

```
mult = n => k => ?
```

λ

$$\textsc{Mult} := \lambda nk.?$$

$$\textsc{Mult}\ \textsc{n2}\ \textsc{n3}\ f\ a = ((\textsc{n3}\ f) \circ (\textsc{n3}\ f))\ a$$

JS

```
mult = n => k => ?
```

λ

$$\textsc{Mult} := \lambda nk.?$$

$$\textsc{Mult}\ \textsc{n2}\ \textsc{n3}\ f\ a = \textsc{n2}\ (\textsc{n3}\ f)\ a$$

JS

```
mult = n => k => ?
```

λ

$$\textsc{Mult} := \lambda nk.?$$

$$\textsc{Mult}\ \textsc{n2}\ \textsc{n3}\ f\ a = \textsc{n2}\ (\textsc{n3}\ f)\ a$$

JS

```
mult = n => k => ?
```

λ

$$\textsc{Mult} := \lambda nk.?$$

$$\textsc{Mult}\ \textsc{n2}\ \textsc{n3}\ f = \textsc{n2}\ (\textsc{n3}\ f)$$

JS

```
mult = n => k => ?
```

λ

$$\textsc{Mult} := \lambda nkf.n(kf)$$

$$\textsc{Mult}\ \textsc{n2}\ \textsc{n3}\ f = \textsc{n2}\ (\textsc{n3}\ f)$$

JS

```
mult = n => k => n(k(f))
```

$$
\begin{aligned}
&\textsc{Mult}\ \textsc{n2}\ \textsc{n3}\ f = \textsc{n2}\ (\textsc{n3}\ f) \\
&= \textsc{Twice}\ (\textsc{Thrice}\ f) \\
&= (f \circ f \circ f) \circ (f \circ f \circ f) \\
&= \textsc{Sixfold}\ f
\end{aligned}
$$

CHURCH ARITHMETIC

Name	λ-Calculus	Use
SUCC	$\lambda nf.\mathrm{B}\ f\ (nf) = \lambda nfa.f(nfa)$	successor of n
ADD	$\lambda nk.n\ \mathrm{SUCC}\ k = \lambda nkf.\mathrm{B}\ (n\ f)\ (k\ f)$	addition of n and k
MULT	$\lambda nkf.n(kf)$	multiplication of n and k

λ

$$\textsc{Mult} := \lambda nkf.n(kf)$$

$$\textsc{Mult}\ \textsc{n2}\ \textsc{n3}\ f = \textsc{n2}\ (\textsc{n3}\ f)$$

JS

```
mult = n => k => n(k(f))
```

λ

$$\textsc{Mult} := \lambda nkf.n(kf)$$

$$\textsc{Mult}\ \textsc{n2}\ \textsc{n3}\ f = (\textsc{n2} \circ \textsc{n3})\ f$$

JS

```
mult = n => k => n(k(f))
```

λ

$$\textsc{Mult} := \lambda nkf.n(kf)$$

$$\textsc{Mult}\ \textsc{n2}\ \textsc{n3}\ f = (\textsc{n2} \circ \textsc{n3})\ f$$

JS

```
mult = n => k => n(k(f))
```

λ

$$\textsc{Mult} := \lambda nkf.n(kf)$$

$$\textsc{Mult}\ \textsc{n2}\ \textsc{n3} = \textsc{n2} \circ \textsc{n3}$$

JS

```
mult = n => k => n(k(f))
```

λ

$$\textsc{Mult} := \lambda nkf.n(kf)$$

$$\textsc{Mult}\ \textsc{n2}\ \textsc{n3} = \mathrm{B}\ \textsc{n2}\ \textsc{n3}$$

JS

```
mult = n => k => n(k(f))
```

λ

$$\textsc{Mult} := \lambda nkf.n(kf)$$

$$\textsc{Mult}\ \textsc{n2}\ \textsc{n3} = \mathrm{B}\ \textsc{n2}\ \textsc{n3}$$

JS

```
mult = n => k => n(k(f))
```

λ

$$\textsc{Mult} := \lambda nkf.n(kf)$$

$$\textsc{Mult} = \mathrm{B}$$

JS

```
mult = n => k => n(k(f))
```

$$\textsc{Mult} := \mathrm{B}$$

α-EQUIVALENCE

$$\begin{aligned} \text{Mult} &:= \lambda\, n\, k\, f.\, n\, (k\, f) \\ &= \text{B} = \lambda\, f\, g\, a.\, f\, (g\, a) \end{aligned}$$

CHURCH ARITHMETIC

Name	λ-Calculus	Use
SUCC	$\lambda nf.\mathrm{B}\ f\ (nf) = \lambda nfa.f(nfa)$	successor of n
ADD	$\lambda nk.n\ \mathrm{SUCC}\ k = \lambda nkf.\mathrm{B}\ (n\ f)\ (k\ f)$	addition of n and k
MULT	$\lambda nkf.n(kf) = \mathrm{B}$	multiplication of n and k

λ

$$\textsc{Pow} := \lambda nk.?$$

$$\textsc{Pow}\ \textsc{n2}\ \textsc{n3} = \textsc{n8}$$

JS

```
pow = n => k => ?
```

λ

$$\textsc{Pow} := \lambda nk.?$$

$$\textsc{Pow}\ \textsc{n2}\ \textsc{n3} = \textsc{n2} \times \textsc{n2} \times \textsc{n2}$$

JS

```
pow = n => k => ?
```

λ

$$\textsc{Pow} := \lambda nk.?$$

$$\textsc{Pow}\ \textsc{n2}\ \textsc{n3} = \textsc{n2} \circ \textsc{n2} \circ \textsc{n2}$$

JS

```
pow = n => k => ?
```

λ

$$\textsc{Pow} := \lambda nk.?$$

$$\textsc{Pow}\ \textsc{n2}\ \textsc{n3} = \textsc{n3}\ \textsc{n2}$$

JS

```
pow = n => k => ?
```

λ

$$\textsc{Pow} := \lambda nk.kn$$

$$\textsc{Pow}\ \textsc{n2}\ \textsc{n3} = \textsc{n3}\ \textsc{n2}$$

JS

```
pow = n => k => k(n)
```

$$
\begin{aligned}
&\textsc{Pow n2 n3} = \textsc{n3 n2} \\
&= \textsc{Thrice Twice} \\
&= \textsc{Twice} \circ \textsc{Twice} \circ \textsc{Twice} \\
&= \textsc{Eightfold}
\end{aligned}
$$

CHURCH ARITHMETIC

Name	λ-Calculus	Use
SUCC	$\lambda nf.\mathrm{B}\ f\ (nf) = \lambda nfa.f(nfa)$	successor of n
ADD	$\lambda nk.n\ \mathrm{SUCC}\ k = \lambda nkf.\mathrm{B}\ (n\ f)\ (k\ f)$	addition of n and k
MULT	$\lambda nkf.n(kf) = \mathrm{B}$	multiplication of n and k
POW	$\lambda nk.kn$	raise n to the power of k

$\lambda af.fa$

THRUSH

COMBINATORS

Sym.	Bird	λ-Calculus	Use	Haskell
I	Idiot	$\lambda a.a$	identity	`id`
M	Mockingbird	$\lambda f.ff$	self-application	(cannot define)
K	Kestrel	$\lambda ab.a$	true, first, const	`const`
KI	Kite	$\lambda ab.b = \mathrm{KI} = \mathrm{CK}$	false, second	`const id`
C	Cardinal	$\lambda fab.fba$	reverse arguments	`flip`
B	Bluebird	$\lambda fga.f(ga)$	$1^\circ \leftarrow 1^\circ$ composition	`(.)`
$\mathrm{T_h}$	Thrush	$\lambda af.fa$	hold an argument	

COMBINATORS

Sym.	Bird	λ-Calculus	Use	Haskell
I	Idiot	$\lambda a.a$	identity	`id`
M	Mockingbird	$\lambda f.ff$	self-application	(cannot define)
K	Kestrel	$\lambda ab.a$	true, first, const	`const`
KI	Kite	$\lambda ab.b = \mathrm{KI} = \mathrm{CK}$	false, second	`const id`
C	Cardinal	$\lambda fab.fba$	reverse arguments	`flip`
B	Bluebird	$\lambda fga.f(ga)$	$1^\circ \leftarrow 1^\circ$ composition	`(.)`
$\mathrm{T_h}$	Thrush	$\lambda af.fa = \mathrm{CI}$	hold an argument	`flip id`

CHURCH ARITHMETIC

Name	λ-Calculus	Use
SUCC	$\lambda nf.\mathrm{B}\ f\ (nf) = \lambda nfa.f(nfa)$	successor of n
ADD	$\lambda nk.n\ \mathrm{SUCC}\ k = \lambda nkf.\mathrm{B}\ (n\ f)\ (k\ f)$	addition of n and k
MULT	$\lambda nkf.n(kf) = \mathrm{B}$	multiplication of n and k
POW	$\lambda nk.kn$	raise n to the power of k

CHURCH ARITHMETIC

Name	λ-Calculus	Use
SUCC	$\lambda nf.\mathrm{B}\ f\ (nf) = \lambda nfa.f(nfa)$	successor of n
ADD	$\lambda nk.n\ \mathrm{SUCC}\ k = \lambda nkf.\mathrm{B}\ (n\ f)\ (k\ f)$	addition of n and k
MULT	$\lambda nkf.n(kf) = \mathrm{B}$	multiplication of n and k
POW	$\lambda nk.kn = \mathrm{T_h}$	raise n to the power of k

ISZERO

$$\textsc{Is0}\ \textsc{n0} = \mathrm{T}$$

$$\textsc{Is0}\ \textsc{n1} = \mathrm{F}$$

$$\textsc{Is0}\ \textsc{n2} = \mathrm{F}$$

$$\ldots$$

λ $\quad \mathrm{Is0} := \lambda n.?$

JS

```
is0 = n => ?
```

λ

$$\mathrm{Is0} := \lambda n.n\ \mathrm{func}\ \mathrm{arg}$$

JS

```
is0 = n => n(func)(arg)
```

λ

$$\text{Is0} := \lambda n.\overset{\text{N0}}{n}\ \cancel{\text{func}}\ \text{arg}$$

JS

```
is0 = n => n(func)(arg)
```

λ

$$\text{Is0} := \lambda n.\overset{\text{N0}}{n}\ \cancel{\text{func}}\ \text{T}$$

JS

```
is0 = n => n(func)(T)
```

λ

$$\text{Is0} := \lambda n.n \overset{\text{N1}}{\underline{\text{func}}}\ \text{T} \longrightarrow \text{F}$$

JS

```
is0 = n => n(func)(T)
```

λ

$$\text{Is0} := \lambda n.n\ \overset{\text{N2}}{\underline{\text{func}}}\ \text{T} \longrightarrow \text{F}$$

JS

```
is0 = n => n(func)(T)
```

λ

$$\mathrm{N} > 0$$

$$\mathrm{Is0} := \lambda n.n\ \underline{\mathrm{func}}\ \mathrm{T} \longrightarrow \mathrm{F}$$

JS

```
is0 = n => n(func)(T)
```

λ

$$\text{Is0} := \lambda n.n\ (\text{KF})\ \text{T}$$

JS

```
is0 = n => n(K(F))(T)
```

λ

$$\textsc{Is0} := \lambda n.n\ (\mathrm{KF})\ \mathrm{T}$$

$$\textsc{Is0}\ \textsc{n3} = \mathrm{KF}(\mathrm{KF}(\mathrm{KF}(\mathrm{T})))$$

JS

```
is0 = n => n(K(F))(T)
```

λ

$$\textsc{Is0} := \lambda n.n\ (\mathrm{KF})\ \mathrm{T}$$

$$\textsc{Is0}\ \textsc{n3} = \mathrm{KF}(\cancel{\mathrm{KF}(\mathrm{KF}(\mathrm{T}))})$$

JS

```
is0 = n => n(K(F))(T)
```

λ

$$\textsc{Is0} := \lambda n.n\ (\mathrm{KF})\ \mathrm{T}$$

$$\textsc{Is0}\ \textsc{n3} = \mathrm{KF}(\cancel{\mathrm{KF}(\mathrm{KF}(\mathrm{T}))}) = \mathrm{F}$$

JS

```
is0 = n => n(K(F))(T)
```

CHURCH ARITHMETIC: BOOLEAN OPS

Name	λ-Calculus	Use
Is0	$\lambda n.n\ (\mathrm{K}\ \mathrm{F})\ \mathrm{T}$	test if $n = 0$

+ × ^

$$-1\ ?$$

$$\textsc{Pred} := \lambda n.n\ (\lambda g.\mathrm{Is0}\ (g\ \mathrm{N1})\ \mathrm{I}\ (\mathrm{B}\ \textsc{Succ}\ g))\ (\mathrm{K}\ \mathrm{N0})\ \mathrm{N0}$$

$$\textsc{Pred} := \lambda n.n\ (\lambda g.\mathrm{Is0}\ (g\ \mathrm{N1})\ \mathrm{I}\ (\mathrm{B}\ \textsc{Succ}\ g))\ (\mathrm{K}\ \mathrm{N0})\ \mathrm{N0}$$

N0: $\mathrm{N0}\ (\lambda g.\mathrm{Is0}\ (g\ \mathrm{N1})\ \mathrm{I}\ (\mathrm{B}\ \textsc{Succ}\ g))\ (\mathrm{K}\ \mathrm{N0})\ \mathrm{N0}$ N0

$(\underline{\mathrm{K}\ \mathrm{N0}})$

N1: $\mathrm{N1}\ (\lambda g.\mathrm{Is0}\ (g\ \mathrm{N1})\ \mathrm{I}\ (\mathrm{B}\ \textsc{Succ}\ g))\ (\mathrm{K}\ \mathrm{N0})\ \mathrm{N0}$ N0

I

N2: $\mathrm{N2}\ (\lambda g.\mathrm{Is0}\ (g\ \mathrm{N1})\ \mathrm{I}\ (\mathrm{B}\ \textsc{Succ}\ g))\ (\mathrm{K}\ \mathrm{N0})\ \mathrm{N0}$ N1

$(\underline{\textsc{Succ} \circ \mathrm{I}})$

N3: $\mathrm{N3}\ (\lambda g.\mathrm{Is0}\ (g\ \mathrm{N1})\ \mathrm{I}\ (\mathrm{B}\ \textsc{Succ}\ g))\ (\mathrm{K}\ \mathrm{N0})\ \mathrm{N0}$ N2

PAIR FST SND PHI

λabf.fab

VIREO

λ

$$\mathrm{V} := \lambda abf.fab$$

JS

```
V = a => b => f => f(a)(b)
```

$$\mathrm{V} := \lambda abf.fab$$

```
V = a => b => f => f(a)(b)
```

λ

$$\mathrm{V\ I\ M}$$
$$= \quad \lambda f.f\ \mathrm{I\ M}$$

JS

```
V(I)(M)
//  f => f(I)(M)
```

$$\mathrm{V} := \lambda abf.fab$$

```
V = a => b => f => f(a)(b)
```

λ

$$\underline{\mathrm{V\ I\ M}}\ \mathrm{K}$$

$$= (\lambda f.f\ \mathrm{I\ M})\ \mathrm{K} \quad = ?$$

JS

```
V(I)(M)(K)
// (f => f(I)(M))(K)  === ?
```

$$\mathrm{V} := \lambda abf.fab$$

```
V = a => b => f => f(a)(b)
```

λ

$$\underline{\mathrm{V\ I\ M}}\ \mathrm{K}$$

$$= (\lambda f.f\ \mathrm{I}\ \mathrm{M})\ \mathrm{K} \quad = \mathrm{I}$$

JS

```
V(I)(M)(K)
// (f => f(I)(M))(K)  === I
```

$$\mathrm{V} := \lambda abf.fab$$

```
V = a => b => f => f(a)(b)
```

λ

$$\underline{\mathrm{V\ I\ M}}\ \mathrm{KI}$$
$$= (\lambda f.f\ \mathrm{I\ M})\ \mathrm{KI} = \mathrm{M}$$

JS

```
V(I)(M)(KI)
// (f => f(I)(M))(KI) === M
```

Copyrighted Material
Purely Functional
Data Structures
Chris Okasaki

COMBINATORS

Sym.	Bird	λ-Calculus	Use	Haskell
I	Idiot	$\lambda a.a$	identity	`id`
M	Mockingbird	$\lambda f.ff$	self-application	(cannot define)
K	Kestrel	$\lambda ab.a$	true, first, const	`const`
KI	Kite	$\lambda ab.b = \mathrm{KI} = \mathrm{CK}$	false, second	`const id`
C	Cardinal	$\lambda fab.fba$	reverse arguments	`flip`
B	Bluebird	$\lambda fga.f(ga)$	$1^\circ \leftarrow 1^\circ$ composition	`(.)`
$\mathrm{T_h}$	Thrush	$\lambda af.fa = \mathrm{CI}$	hold an argument	`flip id`
V	Vireo	$\lambda abf.fab = \mathrm{BCT}$	hold a pair of args	`flip . flip id`

CHURCH PAIRS

Sym.	Name	λ-Calculus	Use
	PAIR	$\lambda abf.fab = \mathrm{V}$	pair two arguments

$$\textsc{Fst} := \lambda p.p\mathrm{K}$$

$$\textsc{Snd} := \lambda p.p(\mathrm{KI})$$

CHURCH PAIRS

Sym.	Name	λ-Calculus	Use
	PAIR	$\lambda abf.fab = \mathrm{V}$	pair two arguments
	FST	$\lambda p.p\mathrm{K}$	extract first of pair
	SND	$\lambda p.p(\mathrm{KI})$	extract second of pair

$$\textsc{Phi} := \lambda p.\mathrm{V}\ (\textsc{Snd}\ p)\ (\textsc{Succ}\ (\textsc{Snd}\ p))$$

```
phi = p => pair (snd(p)) (succ(snd(p)))
```

$$\Phi := \lambda p.\textsc{Pair}\ (\textsc{Snd}\ p)\ (\textsc{Succ}\ (\textsc{Snd}\ p))$$

$$\Phi\ (\mathrm{M}, \textsc{n}7) = ?$$

$$\Phi := \lambda p.\underline{\textsc{Pair}}\ (\textsc{Snd}\ p)\ (\textsc{Succ}\ (\textsc{Snd}\ p))$$

$$\Phi\ (\mathrm{M}, \textsc{n}7) = \underline{(\ ,\ )}$$

$$\Phi := \lambda p.\textsc{Pair}\ (\underline{\textsc{Snd}\ p})\ (\textsc{Succ}\ (\textsc{Snd}\ p))$$

$$\Phi\ (\mathrm{M}, \underline{\textsc{n}7}) = (\underline{\textsc{n}7}, \ )$$

$$\Phi := \lambda p.\textsc{Pair}\ (\textsc{Snd}\ p)\ (\underline{\textsc{Succ}\ (\textsc{Snd}\ p)})$$

$$\Phi\ (\mathrm{M}, \underline{\textsc{n7}}) = (\textsc{n7}, \underline{\textsc{n8}})$$

$$\Phi := \lambda p.\textsc{Pair}\ (\textsc{Snd}\ p)\ (\textsc{Succ}\ (\textsc{Snd}\ p))$$

$$\Phi\ (\mathrm{M}, \textsc{n}7) = (\textsc{n}7, \textsc{n}8)$$

$$\Phi\ (\textsc{n}9, \textsc{n}2) = (\textsc{n}2, \textsc{n}3)$$

CHURCH PAIRS

Sym.	Name	λ-Calculus	Use
	PAIR	$\lambda abf.fab = \mathrm{V}$	pair two arguments
	FST	$\lambda p.p\mathrm{K}$	extract first of pair
	SND	$\lambda p.p(\mathrm{KI})$	extract second of pair
Φ	PHI	$\lambda p.\mathrm{PAIR}\ (\mathrm{SND}\ p)\ (\mathrm{SUCC}\ (\mathrm{SND}\ p)$	copy 2nd to 1st, inc 2nd

$$\Phi := \lambda p.\textsc{Pair}\ (\textsc{Snd}\ p)\ (\textsc{Succ}\ (\textsc{Snd}\ p))$$

$$\Phi\ (\mathrm{M}, \textsc{n}7) = (\textsc{n}7, \textsc{n}8)$$

$$\Phi\ (\textsc{n}9, \textsc{n}2) = (\textsc{n}2, \textsc{n}3)$$

$$\Phi\ (\textsc{n}0, \textsc{n}0) = (\textsc{n}0, \textsc{n}1)$$

$$\Phi := \lambda p.\textsc{Pair}\ (\textsc{Snd}\ p)\ (\textsc{Succ}\ (\textsc{Snd}\ p))$$

$$\Phi\ (\mathrm{M}, \textsc{n}7) = (\textsc{n}7, \textsc{n}8)$$
$$\Phi\ (\textsc{n}9, \textsc{n}2) = (\textsc{n}2, \textsc{n}3)$$
$$\Phi\ (\textsc{n}0, \textsc{n}0) = (\textsc{n}0, \textsc{n}1)$$
$$\Phi\ (\textsc{n}0, \textsc{n}1) = (\textsc{n}1, \textsc{n}2)$$

$$\Phi := \lambda p.\textsc{Pair}\ (\textsc{Snd}\ p)\ (\textsc{Succ}\ (\textsc{Snd}\ p))$$

$$\Phi\ (\mathrm{M}, \textsc{n}7) = (\textsc{n}7, \textsc{n}8)$$
$$\Phi\ (\textsc{n}9, \textsc{n}2) = (\textsc{n}2, \textsc{n}3)$$
$$\Phi\ (\textsc{n}0, \textsc{n}0) = (\textsc{n}0, \textsc{n}1)$$
$$\Phi\ (\textsc{n}0, \textsc{n}1) = (\textsc{n}1, \textsc{n}2)$$
$$\Phi\ (\textsc{n}1, \textsc{n}2) = (\textsc{n}2, \textsc{n}3)$$

$$\Phi := \lambda p.\textsc{Pair}\ (\textsc{Snd}\ p)\ (\textsc{Succ}\ (\textsc{Snd}\ p))$$

$$\Phi\ (\mathrm{M}, \textsc{n}7) = (\textsc{n}7, \textsc{n}8)$$
$$\Phi\ (\textsc{n}9, \textsc{n}2) = (\textsc{n}2, \textsc{n}3)$$
$$\Phi\ (\textsc{n}0, \textsc{n}0) = (\textsc{n}0, \textsc{n}1)$$
$$\Phi\ (\textsc{n}0, \textsc{n}1) = (\textsc{n}1, \textsc{n}2)$$
$$\Phi\ (\textsc{n}1, \textsc{n}2) = (\textsc{n}2, \textsc{n}3)$$
$$\textsc{n}8\ \Phi\ (\textsc{n}0, \textsc{n}0) = ?$$

$$\Phi := \lambda p.\textsc{Pair}\ (\textsc{Snd}\ p)\ (\textsc{Succ}\ (\textsc{Snd}\ p))$$

$$\Phi\ (\mathrm{M}, \textsc{n}7) = (\textsc{n}7, \textsc{n}8)$$
$$\Phi\ (\textsc{n}9, \textsc{n}2) = (\textsc{n}2, \textsc{n}3)$$
$$\Phi\ (\textsc{n}0, \textsc{n}0) = (\textsc{n}0, \textsc{n}1)$$
$$\Phi\ (\textsc{n}0, \textsc{n}1) = (\textsc{n}1, \textsc{n}2)$$
$$\Phi\ (\textsc{n}1, \textsc{n}2) = (\textsc{n}2, \textsc{n}3)$$
$$\textsc{n}8\ \Phi\ (\textsc{n}0, \textsc{n}0) = (\textsc{n}7, \textsc{n}8)$$

$$\Phi := \lambda p.\textsc{Pair}\ (\textsc{Snd}\ p)\ (\textsc{Succ}\ (\textsc{Snd}\ p))$$

$$\Phi\ (\mathrm{M}, \textsc{n}7) = (\textsc{n}7, \textsc{n}8)$$
$$\Phi\ (\textsc{n}9, \textsc{n}2) = (\textsc{n}2, \textsc{n}3)$$
$$\Phi\ (\textsc{n}0, \textsc{n}0) = (\textsc{n}0, \textsc{n}1)$$
$$\Phi\ (\textsc{n}0, \textsc{n}1) = (\textsc{n}1, \textsc{n}2)$$
$$\Phi\ (\textsc{n}1, \textsc{n}2) = (\textsc{n}2, \textsc{n}3)$$
$$\textsc{Fst}\ (\ \textsc{n}8\ \Phi\ (\textsc{n}0, \textsc{n}0)) = \textsc{Fst}\ (\textsc{n}7, \textsc{n}8) = \textsc{n}7$$

$$\Phi := \lambda p.\textsc{Pair}\ (\textsc{Snd}\ p)\ (\textsc{Succ}\ (\textsc{Snd}\ p))$$

$$\Phi\ (\mathrm{M}, \textsc{n}7) = (\textsc{n}7, \textsc{n}8)$$

$$\Phi\ (\textsc{n}9, \textsc{n}2) = (\textsc{n}2, \textsc{n}3)$$

$$\Phi\ (\textsc{n}0, \textsc{n}0) = (\textsc{n}0, \textsc{n}1)$$

$$\Phi\ (\textsc{n}0, \textsc{n}1) = (\textsc{n}1, \textsc{n}2)$$

$$\Phi\ (\textsc{n}1, \textsc{n}2) = (\textsc{n}2, \textsc{n}3)$$

$$\underline{\textsc{Fst}\ (\ \textsc{n}8\ \Phi\ (\textsc{n}0, \textsc{n}0))} = \textsc{Fst}\ (\textsc{n}7, \textsc{n}8) = \textsc{n}7$$

λ

$$\textsc{Pred} := \lambda n.\textsc{Fst}\ (n\ \Phi\ (\textsc{Pair}\ \textsc{n0}\ \textsc{n0}))$$

JS

```
pred = n => fst(n(phi)(pair(n0)(n0)))
```

CHURCH ARITHMETIC

Name	λ-Calculus	Use
SUCC	$\lambda nf.\mathrm{B}\ f\ (nf) = \lambda nfa.f(nfa)$	successor of n
ADD	$\lambda nk.n\ \mathrm{SUCC}\ k = \lambda nkf.\mathrm{B}\ (n\ f)\ (k\ f)$	addition of n and k
MULT	$\lambda nkf.n(kf) = \mathrm{B}$	multiplication of n and k
POW	$\lambda nk.kn = \mathrm{T_h}$	raise n to the power of k
PRED	$\lambda n.n\ (\lambda g.\mathrm{IS0}\ (g\ \mathrm{N1})\ \mathrm{I}\ (\mathrm{B}\ \mathrm{SUCC}\ g))\ (\mathrm{K}\ \mathrm{N0})\ \mathrm{N0}$	predecessor of n
PRED	$\lambda n.\mathrm{FST}\ (n\ \Phi\ (\mathrm{PAIR}\ \mathrm{N0}\ \mathrm{N0}))$	predecessor of n (easier)

SUB LEQ EQ GT

λ

$$\textsc{Sub} := \lambda nk.k\ \textsc{Pred}\ n$$

JS

```
sub = n => k => k(pred)(n)
```

CHURCH ARITHMETIC

Name	λ-Calculus	Use
SUCC	$\lambda nf.\mathrm{B}\ f\ (nf) = \lambda nfa.f(nfa)$	successor of n
ADD	$\lambda nk.n\ \textsc{Succ}\ k = \lambda nkf.\mathrm{B}\ (n\ f)\ (k\ f)$	addition of n and k
MULT	$\lambda nkf.n(kf) = \mathrm{B}$	multiplication of n and k
POW	$\lambda nk.kn = \mathrm{T_h}$	raise n to the power of k
PRED	$\lambda n.n\ (\lambda g.\mathrm{Is0}\ (g\ \mathrm{N1})\ \mathrm{I}\ (\mathrm{B}\ \textsc{Succ}\ g))\ (\mathrm{K}\ \mathrm{N0})\ \mathrm{N0}$	predecessor of n
PRED	$\lambda n.\textsc{Fst}\ (n\ \Phi\ (\textsc{Pair}\ \mathrm{N0}\ \mathrm{N0}))$	predecessor of n (easier)
SUB	$\lambda nk.k\ \textsc{Pred}\ n$	subtract k from n

λ

$$\textsc{Leq} := \lambda nk.\textsc{Is0}\ (\textsc{Sub}\ n\ k)$$

JS

```
leq = n => k => is0(sub(n)(k))
```

λ

$$\textsc{Eq} := \lambda nk.\textsc{And}(\textsc{Leq}\ n\ k)(\textsc{Leq}\ k\ n)$$

JS

```
eq = n => k => and(leq(n)(k))(leq(k)(n))
```

CHURCH ARITHMETIC: BOOLEAN OPS

Name	λ-Calculus	Use
Is0	$\lambda n.n\ (\mathrm{K}\ \mathrm{F})\ \mathrm{T}$	test if $n = 0$
Leq	$\lambda nk.\mathrm{Is0}\ (\mathrm{Sub}\ n\ k)$	test if $n \leqslant k$
Eq	$\lambda nk.\mathrm{And}\ (\mathrm{Leq}\ n\ k)\ (\mathrm{Leq}\ k\ n)$	test if $n = k$

λ

$$\textsc{Gt} := \lambda nk.\textsc{Not}\ (\textsc{Leq}\ \mathrm{n}\ \mathrm{k})$$

JS

```
eq = n => k => not(leq(n)(k))
```

λ

$$\textsc{Gt} := \lambda nk.\textsc{Not}\ (\textsc{Leq}\ \mathrm{n}\ \mathrm{k})$$

JS

```
eq = n => k => not(leq(n)(k))
```

λ

$$\textsc{Gt} := \mathrm{B}\ \textsc{Not}\ \textsc{Leq}\ ???$$

JS

```
eq = B(not)(leq) ???
```

λ

$$\textsc{Gt} := \text{Ƀ}\ \textsc{Not}\ \textsc{Leq}$$

JS

```
eq = Ƀ(not)(leq)
```

λ

$$\textsc{Gt} := \lambda nk.\textsc{Not}\ (\textsc{Leq}\ \mathrm{n}\ \mathrm{k})$$

JS

```
eq = n => k => not(leq(n)(k))
```

$$\lambda fgab.f(gab)$$

BLACKBIRD

$$\mathrm{B}_1 = \lambda fgab.f(gab)$$

λ

$$\textsc{Gt} := \mathrm{B}_1\ \textsc{Not}\ \textsc{Leq}$$

JS

```
eq = B1(not)(leq)
```

CHURCH ARITHMETIC: BOOLEAN OPS

Name	λ-Calculus	Use
Is0	$\lambda n.n\ (\mathrm{K}\ \mathrm{F})\ \mathrm{T}$	test if $n = 0$
Leq	$\lambda nk.\mathrm{Is0}\ (\mathrm{Sub}\ n\ k)$	test if $n \leqslant k$
Eq	$\lambda nk.\mathrm{And}\ (\mathrm{Leq}\ n\ k)\ (\mathrm{Leq}\ k\ n)$	test if $n = k$
Gt	$\lambda nk.\mathrm{B}_1\ \mathrm{Not}\ \mathrm{Leq}$	test if $n > k$

COMBINATORS

Sym.	Bird	λ-Calculus	Use	Haskell
I	Idiot	$\lambda a.a$	identity	`id`
M	Mockingbird	$\lambda f.ff$	self-application	(cannot define)
K	Kestrel	$\lambda ab.a$	true, first, const	`const`
KI	Kite	$\lambda ab.b = \mathrm{KI} = \mathrm{CK}$	false, second	`const id`
C	Cardinal	$\lambda fab.fba$	reverse arguments	`flip`
B	Bluebird	$\lambda fga.f(ga)$	$1^\circ \leftarrow 1^\circ$ composition	`(.)`
$\mathrm{T_h}$	Thrush	$\lambda af.fa = \mathrm{CI}$	hold an argument	`flip id`
V	Vireo	$\lambda abf.fab = \mathrm{BCT}$	hold a pair of args	`flip . flip id`
$\mathrm{B_1}$	Blackbird	$\lambda fgab.f(gab)$	$1^\circ \leftarrow 2^\circ$ composition	

COMBINATORS

Sym.	Bird	λ-Calculus	Use	Haskell
I	Idiot	$\lambda a.a$	identity	`id`
M	Mockingbird	$\lambda f.ff$	self-application	(cannot define)
K	Kestrel	$\lambda ab.a$	true, first, const	`const`
KI	Kite	$\lambda ab.b = \mathrm{KI} = \mathrm{CK}$	false, second	`const id`
C	Cardinal	$\lambda fab.fba$	reverse arguments	`flip`
B	Bluebird	$\lambda fga.f(ga)$	$1^\circ \leftarrow 1^\circ$ composition	`(.)`
$\mathrm{T_h}$	Thrush	$\lambda af.fa = \mathrm{CI}$	hold an argument	`flip id`
V	Vireo	$\lambda abf.fab = \mathrm{BCT}$	hold a pair of args	`flip . flip id`
$\mathrm{B_1}$	Blackbird	$\lambda fgab.f(gab) = \mathrm{BBB}$	$1^\circ \leftarrow 2^\circ$ composition	`(.) . (.)`

B C K I

$$\mathrm{KI} = \mathrm{K\ I} = \mathrm{C\ K}$$

$$\mathrm{B_1} = \mathrm{B\ B\ B}$$

$$\mathrm{T_h} = \mathrm{C\ I}$$

$$\mathrm{V} = \mathrm{B\ C\ T_h} = \mathrm{B\ C\ (C\ I)}$$

QUESTION

how many combinators are needed to form a <u>basis</u>?

20? 10? 5?

STARLING · KESTREL

$$\mathrm{S} := \lambda abc.ac(bc)$$

$$\mathrm{K} := \lambda ab.a$$

THE SK COMBINATOR CALCULUS

$$\mathrm{I} = \mathrm{S}\ \mathrm{K}\ \mathrm{K}$$

$$\mathrm{I} = \mathrm{S}\ \mathrm{K}\ \mathrm{K}$$

$$\mathrm{V} = (\mathrm{S}(\mathrm{K}((\mathrm{S}((\mathrm{S}(\mathrm{K}((\mathrm{S}(\mathrm{KS}))\mathrm{K})))\mathrm{S}))(\mathrm{KK}))))((\mathrm{S}(\mathrm{K}(\mathrm{S}((\mathrm{SK})\mathrm{K}))))\mathrm{K})$$

seriously though,

why?

ADDENDUM

COMBINATORS

Sym.	Bird	λ-Calculus	Use	Haskell
I	Idiot	$\lambda a.a$	identity	`id`
M	Mockingbird	$\lambda f.ff$	self-application	(cannot define)
K	Kestrel	$\lambda ab.a$	true, first, const	`const`
KI	Kite	$\lambda ab.b = \mathrm{KI} = \mathrm{CK}$	false, second	`const id`
C	Cardinal	$\lambda fab.fba$	reverse arguments	`flip`
B	Bluebird	$\lambda fga.f(ga)$	$1^\circ \leftarrow 1^\circ$ composition	`(.)`
$\mathrm{T_h}$	Thrush	$\lambda af.fa = \mathrm{CI}$	hold an argument	`flip id`
V	Vireo	$\lambda abf.fab = \mathrm{BCT}$	hold a pair of args	`flip . flip id`
$\mathrm{B_1}$	Blackbird	$\lambda fgab.f(gab) = \mathrm{BBB}$	$1^\circ \leftarrow 2^\circ$ composition	`(.) . (.)`

CHURCH ENCODINGS: BOOLEANS

Sym.	Name	λ-Calculus	Use
T	TRUE	$\lambda ab.a = \mathrm{K} = \mathrm{C}(\mathrm{KI})$	encoding for true
F	FALSE	$\lambda ab.b = \mathrm{KI} = \mathrm{CK}$	encoding for false
	NOT	$\lambda p.p\mathrm{FT}$ or C	negation
	AND	$\lambda pq.pq\mathrm{F}$ or $\lambda pq.pqp$	conjunction
	OR	$\lambda pq.p\mathrm{T}q$ or $\lambda pq.ppq = \mathrm{M}^*$	disjunction
	BEQ	$\lambda pq.p\ q\ (\mathrm{NOT}\ q)$	equality

CHURCH ENCODINGS: NUMERALS

Sym.	Name	λ-Calculus	Use
N0	Zero	$\lambda fa.a = \mathrm{F}$	apply f no times to a
N1	Once	$\lambda fa.f\ a = \mathrm{I}^*$	apply f once to a
N2	Twice	$\lambda fa.f\ (f\ a)$	apply 2-fold f to a
N3	Thrice	$\lambda fa.f\ (f\ (f\ a))$	apply 3-fold f to a
N4	Fourfold	$\lambda fa.f\ (f\ (f\ (f\ a)))$	apply 4-fold f to a
N5	Fivefold	$\lambda fa.f\ (f\ (f\ (f\ (f\ a)))))$	apply 5-fold f to a

CHURCH ARITHMETIC

Name	λ-Calculus	Use
SUCC	$\lambda nf.\mathrm{B}\ f\ (nf) = \lambda nfa.f(nfa)$	successor of n
ADD	$\lambda nk.n\ \textsc{Succ}\ k = \lambda nkf.\mathrm{B}\ (n\ f)\ (k\ f)$	addition of n and k
MULT	$\lambda nkf.n(kf) = \mathrm{B}$	multiplication of n and k
POW	$\lambda nk.kn = \mathrm{T_h}$	raise n to the power of k
PRED	$\lambda n.n\ (\lambda g.\mathrm{Is0}\ (g\ \mathrm{N1})\ \mathrm{I}\ (\mathrm{B}\ \textsc{Succ}\ g))\ (\mathrm{K}\ \mathrm{N0})\ \mathrm{N0}$	predecessor of n
PRED	$\lambda n.\textsc{Fst}\ (n\ \Phi\ (\textsc{Pair}\ \mathrm{N0}\ \mathrm{N0}))$	predecessor of n (easier)
SUB	$\lambda nk.k\ \textsc{Pred}\ n$	subtract k from n

CHURCH ARITHMETIC: BOOLEAN OPS

Name	λ-Calculus	Use
Is0	$\lambda n.n\ (\mathrm{K}\ \mathrm{F})\ \mathrm{T}$	test if $n = 0$
Leq	$\lambda nk.\mathrm{Is0}\ (\mathrm{Sub}\ n\ k)$	test if $n \leqslant k$
Eq	$\lambda nk.\mathrm{And}\ (\mathrm{Leq}\ n\ k)\ (\mathrm{Leq}\ k\ n)$	test if $n = k$
Gt	$\lambda nk.\mathrm{B}_1\ \mathrm{Not}\ \mathrm{Leq}$	test if $n > k$

CHURCH PAIRS

Sym.	Name	λ-Calculus	Use
	PAIR	$\lambda abf.fab = \mathrm{V}$	pair two arguments
	FST	$\lambda p.p\mathrm{K}$	extract first of pair
	SND	$\lambda p.p(\mathrm{KI})$	extract second of pair
Φ	PHI	$\lambda p.\textsc{Pair}\ (\textsc{Snd}\ p)\ (\textsc{Succ}\ (\textsc{Snd}\ p)$	copy 2nd to 1st, inc 2nd
	SET1ST	$\lambda cp.\textsc{Pair}\ c\ (\textsc{Snd}\ p)$	set first, immutably
	SET2ND	$\lambda cp.\textsc{Pair}\ (\textsc{Fst}\ p)\ c$	set second, immutably

$$\lambda f.\mathrm{M}(\lambda x.f(\mathrm{M}x))$$

THE Y FIXED-POINT COMBINATOR

EVALUATION STRATEGIES

CALL BY NAME

(apply to args before reduction)

$$
\begin{aligned}
& (\lambda ab.b)(\underline{(\lambda f.ff)\lambda f.ff})\ x \\
= \quad & (\lambda b.b) \qquad\qquad\qquad\qquad x \\
= \quad & x
\end{aligned}
$$

(AKA normal order; lazy)

CALL BY VALUE

(reduce args before application)

$$
\begin{aligned}
& (\lambda ab.b)((\lambda f.ff)\underline{\lambda f.ff})x \\
= \quad & (\lambda ab.b)((\lambda f.ff)\lambda f.ff)x \\
= \quad & (\lambda ab.b)((\lambda f.ff)\lambda f.ff)x \\
= \quad & (\lambda ab.b)((\lambda f.ff)\lambda f.ff)x \\
= \quad & (\lambda ab.b)((\lambda f.ff)\lambda f.ff)x
\end{aligned}
$$

(AKA applicative order; strict)

$$\lambda f.\mathrm{M}(\lambda x.f(\lambda v.\mathrm{M}xv))$$

THE Z FIXED-POINT COMBINATOR

ADDITIONAL RESOURCES

Combinator Birds · Rathman · http://bit.ly/2iudab9

To Mock a Mockingbird · Smullyan · http://amzn.to/2g9AlXl

To Dissect a Mockingbird · Keenan · http://dkeenan.com/Lambda

.:.

A Tutorial Introduction to the Lambda Calculus · Rojas · http://bit.ly/1agRC97

Lambda Calculus · Wikipedia · http://bit.ly/1TsPkGn

The Lambda Calculus · Stanford · http://stanford.io/2vtg8hp

.:.

History of Lambda-calculus and Combinatory Logic

Cardone, Hindley · http://bit.ly/2wCxv4k

.:.

An Introduction to Functional Programming through Lambda Calculus · Michaelson · http://amzn.to/2vtts56