The λ-Calculus

Luke Collins

Malta Mathematics Society
University of Malta

18th December, 2019

Formal Languages

- An *alphabet* Σ, is any finite set (of "symbols").
- A *string* from an alphabet Σ is a tuple $(\alpha_1, \ldots, \alpha_n)$ of finite length where each $\alpha_i \in \Sigma$.
  We just denote this as $\alpha_1 \cdots \alpha_n$.
- The set of all strings from Σ is called the *Kleene closure* of Σ, denoted by Σ^*, i.e.,

  $$\Sigma^* := \bigcup_{i=0}^{\infty} \Sigma^i,$$

  where as usual

  $$\Sigma^i = \underbrace{\Sigma \times \cdots \times \Sigma}_{i \text{ times}},$$

  Σ^1 are tuples of length 1, so $\Sigma^1 \simeq \Sigma$ but $\Sigma^1 \neq \Sigma$, and $\Sigma^0 = \{\epsilon\}$, where ϵ is the empty string.
- A language L over Σ is simply a subset $L \subseteq \Sigma^*$.

Examples

- Take $\Sigma = \{\mathsf{a}, \mathsf{b}, \ldots, \mathsf{y}, \mathsf{z}\}$. Then Σ^* is the set of all possible "words", and the English language E is a language over Σ, since $E \subseteq \Sigma^*$.
- Take $\Sigma = \{0, 1\}$. The set of binary palindromes is a language over Σ.
- Take $\Sigma = \{0, \mathsf{S}\}$. Then the set

  $$\{0, \mathsf{S}0, \mathsf{SS}0, \mathsf{SSS}0, \mathsf{SSSS}0, \ldots\},$$

  i.e., $\{\mathsf{S}^n 0 : n \geqslant 0\}$, is a language over Σ.
- Take $\Sigma = \{\wedge, \vee, \neg, \Rightarrow, \Leftrightarrow, \forall, \exists, \{, \}, \in\} \cup V$ where V is a set of "variables". Then mathematics (ZFC) is a language over Σ. The set of true theorems is also a language over Σ.

Semantics

Formal languages are purely syntactic objects, and their members have no inherent semantics.

But to be interesting, we want languages to have semantics! Such as the English language, or mathematics.

Some languages have *dynamics*, which are rules to manipulate strings into other strings.

For example, the language of valid arithmetic expressions over the alphabet $\{1, 2, \ldots, 9, +, -, \times\}$ admits strings such as $1 + 2 \times 3$. Somehow, we want rules to manipulate this string:

$$(1, +, 2, \times, 3) \to (1, +, 6) \to (7)$$

The dynamics encode certain behaviours we expect, which come from the semantics. This is what a compiler/calculator does.

A History of Mathematical Logic

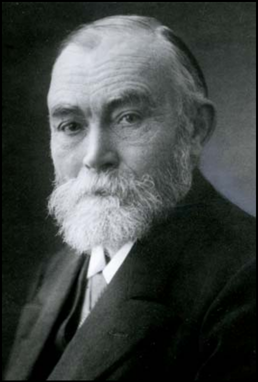
- Giuseppe Peano (1889)
- Gottlob Frege (1891)
- Bertrand Russell (1910)
- Moses Schönfinkel (1920)
- John von Neumann (1925)
- Haskell Curry (1926)
- Kurt Gödel (1931)
- Alonzo Church (1932, 36)
- Stephen Kleene & John Rosser (1932–36)
- Alan Turing (1936)

A History of Mathematical Logic

- Giuseppe Peano (1889)
- Gottlob Frege (1891)
- Bertrand Russell (1910)
- Moses Schönfinkel (1920)
- John von Neumann (1925)
- Haskell Curry (1926)
- Kurt Gödel (1931)
- Alonzo Church (1932, 36)
- Stephen Kleene & John Rosser (1932–36)
- Alan Turing (1936)

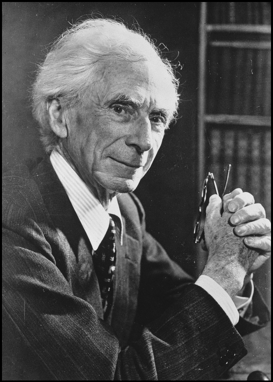
A History of Mathematical Logic

- Giuseppe Peano (1889)
- Gottlob Frege (1891)
- Bertrand Russell (1910)
- Moses Schönfinkel (1920)
- John von Neumann (1925)
- Haskell Curry (1926)
- Kurt Gödel (1931)
- Alonzo Church (1932, 36)
- Stephen Kleene & John Rosser (1932–36)
- Alan Turing (1936)



A History of Mathematical Logic

- Giuseppe Peano (1889)
- Gottlob Frege (1891)
- Bertrand Russell (1910)
- Moses Schönfinkel (1920)
- John von Neumann (1925)
- Haskell Curry (1926)
- Kurt Gödel (1931)
- Alonzo Church (1932, 36)
- Stephen Kleene & John Rosser (1932–36)
- Alan Turing (1936)



A History of Mathematical Logic

- Giuseppe Peano (1889)
- Gottlob Frege (1891)
- Bertrand Russell (1910)
- Moses Schönfinkel (1920)
- John von Neumann (1925)
- Haskell Curry (1926)
- Kurt Gödel (1931)
- Alonzo Church (1932, 36)
- Stephen Kleene & John Rosser (1932–36)
- Alan Turing (1936)

A History of Mathematical Logic

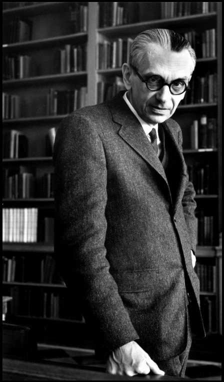
- Giuseppe Peano (1889)
- Gottlob Frege (1891)
- Bertrand Russell (1910)
- Moses Schönfinkel (1920)
- John von Neumann (1925)
- Haskell Curry (1926)
- Kurt Gödel (1931)
- Alonzo Church (1932, 36)
- Stephen Kleene & John Rosser (1932–36)
- Alan Turing (1936)

A History of Mathematical Logic

- Giuseppe Peano (1889)
- Gottlob Frege (1891)
- Bertrand Russell (1910)
- Moses Schönfinkel (1920)
- John von Neumann (1925)
- Haskell Curry (1926)
- Kurt Gödel (1931)
- Alonzo Church (1932, 36)
- Stephen Kleene & John Rosser (1932–36)
- Alan Turing (1936)

A History of Mathematical Logic

- Giuseppe Peano (1889)
- Gottlob Frege (1891)
- Bertrand Russell (1910)
- Moses Schönfinkel (1920)
- John von Neumann (1925)
- Haskell Curry (1926)
- Kurt Gödel (1931)
- Alonzo Church (1932, 36)
- Stephen Kleene & John Rosser (1932–36)
- Alan Turing (1936)

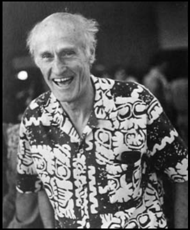


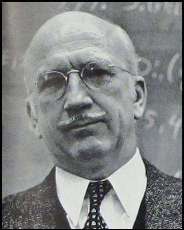
A History of Mathematical Logic

- Giuseppe Peano (1889)
- Gottlob Frege (1891)
- Bertrand Russell (1910)
- Moses Schönfinkel (1920)
- John von Neumann (1925)
- Haskell Curry (1926)
- Kurt Gödel (1931)
- Alonzo Church (1932, 36)
- Stephen Kleene & John Rosser (1932–36)
- Alan Turing (1936)

A History of Mathematical Logic

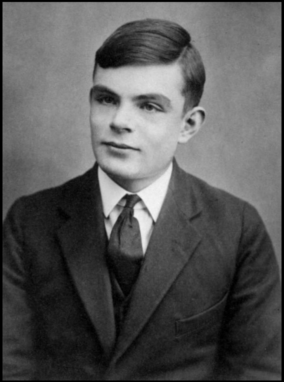
- Giuseppe Peano (1889)
- Gottlob Frege (1891)
- Bertrand Russell (1910)
- Moses Schönfinkel (1920)
- John von Neumann (1925)
- Haskell Curry (1926)
- Kurt Gödel (1931)
- Alonzo Church (1932, 36)
- Stephen Kleene & John Rosser (1932–36)
- Alan Turing (1936)





A History of Mathematical Logic

- Giuseppe Peano (1889)
- Gottlob Frege (1891)
- Bertrand Russell (1910)
- Moses Schönfinkel (1920)
- John von Neumann (1925)
- Haskell Curry (1926)
- Kurt Gödel (1931)
- Alonzo Church (1932, 36)
- Stephen Kleene & John Rosser (1932–36)
- Alan Turing (1936)



What is the λ-calculus?

What is the λ-calculus?

The set (language) of λ-terms is defined by the following BNF.

What is the λ-calculus?

The set (language) of λ-terms is defined by the following BNF.

$$expression ::=$$

What is the λ-calculus?

The set (language) of λ-terms is defined by the following BNF.

$$\textit{expression} ::= x \qquad\qquad \text{(variable)}$$

What is the λ-calculus?

The set (language) of λ-terms is defined by the following BNF.

$$
\begin{aligned}
\mathit{expression} ::=\ & x && \text{(variable)} \\
\mid\ & \lambda x \cdot \mathit{expression} && \text{(abstraction)}
\end{aligned}
$$

What is the λ-calculus?

The set (language) of λ-terms is defined by the following BNF.

$$\begin{aligned} \textit{expression} ::= &\ x && \text{(variable)} \\ \mid &\ \lambda x \cdot \textit{expression} && \text{(abstraction)} \\ \mid &\ \textit{expression}\ \ \textit{expression} && \text{(application)} \end{aligned}$$

What is the λ-calculus?

The set (language) of λ-terms is defined by the following BNF.

$$
\begin{aligned}
\mathit{expression} ::=\ & x && \text{(variable)} \\
\mid\ & \lambda x \cdot \mathit{expression} && \text{(abstraction)} \\
\mid\ & \mathit{expression}\ \ \mathit{expression} && \text{(application)} \\
\mid\ & (\mathit{expression}) && \text{(grouping)}
\end{aligned}
$$

Examples

Examples

VARIABLES

Examples

VARIABLES

x

Examples

VARIABLES

$$x$$

$$(x)$$

Examples

VARIABLES	APPLICATIONS
x	
(x)	

Examples

VARIABLES	APPLICATIONS
x	$f\,x$
(x)	

Examples

VARIABLES	APPLICATIONS
x	$f\,x$
(x)	$f\,x\,y$

Examples

VARIABLES	APPLICATIONS
x	$f\,x$
(x)	$f\,x\,y$
	$(f\,x)\,y$

Examples

VARIABLES	APPLICATIONS
x	$f\,x$
(x)	$f\,x\,y$
	$(f\,x)\,y$

Left associative: $abc \equiv (ab)c$

Examples

VARIABLES	APPLICATIONS
x	$f\ x$
(x)	$f\ x\ y$
	$(f\ x)\ y$
	$f\ (x\ y)$

Left associative: $abc \equiv (ab)c$

Examples

VARIABLES	APPLICATIONS
x	$f\,x$
(x)	$f\,x\,y$
	$(f\,x)\,y$
	$f\,(\underline{x\,y})$

Left associative: $abc \equiv (ab)c$

Examples

VARIABLES	APPLICATIONS	ABSTRACTIONS
x	$f\,x$	
(x)	$f\,x\,y$	
	$(f\,x)\,y$	
	$f\,(x\,y)$	

Left associative: $abc \equiv (ab)c$

Examples

VARIABLES	APPLICATIONS	ABSTRACTIONS
x	$f\,x$	$\lambda x \cdot y$
(x)	$f\,x\,y$	
	$(f\,x)\,y$	
	$f\,(x\,y)$	

Left associative: $abc \equiv (ab)c$

Examples

VARIABLES	APPLICATIONS	ABSTRACTIONS
x	$f\,x$	$\lambda x \cdot y$
(x)	$f\,x\,y$	$\lambda x \cdot x\,y\,z$
	$(f\,x)\,y$	
	$f\,(x\,y)$	

Left associative: $abc \equiv (ab)c$

Examples

VARIABLES	APPLICATIONS	ABSTRACTIONS
x	$f\,x$	$\lambda x \cdot y$
(x)	$f\,x\,y$	$\lambda x \cdot (x\,y\,z)$
	$(f\,x)\,y$	
	$f\,(x\,y)$	

Left associative: $abc \equiv (ab)c$

Examples

VARIABLES	APPLICATIONS	ABSTRACTIONS
x	$f\ x$	$\lambda x \cdot y$
(x)	$f\ x\ y$	$\lambda x \cdot (x\ y\ z)$
	$(f\ x)\ y$	
	$f\ (x\ y)$	

Left associative: $abc \equiv (ab)c$

Abstraction is greedy: $\lambda x \cdots$ 

Examples

VARIABLES	APPLICATIONS	ABSTRACTIONS
x	$f\ x$	$\lambda x \cdot y$
(x)	$f\ x\ y$	$\lambda x \cdot (x\ y\ z)$
	$(f\ x)\ y$	$(\lambda x \cdot x)\ y$
	$f\ (x\ y)$	

Left associative: $abc \equiv (ab)c$

Abstraction is greedy: $\lambda x \cdots$ 

Examples

VARIABLES	APPLICATIONS	ABSTRACTIONS
x	$f\ x$	$\lambda x \cdot y$
(x)	$f\ x\ y$	$\lambda x\ \cdot (x\ y\ z)$
	$(f\ x)\ y$	$(\lambda x \cdot x)\ y$
	$f\ (x\ y)$	$\lambda x \cdot \lambda y \cdot xy$

Left associative: $abc \equiv (ab)c$

Abstraction is greedy: $\lambda x \cdots$ 

Normal Order Reduction

The *dynamics* of the λ-calculus.

Normal Order Reduction

The *dynamics* of the λ-calculus.

Examples $(\lambda x \cdot x)y$

Normal Order Reduction

The *dynamics* of the λ-calculus.

Examples $(\lambda x \cdot x)y \to y$

Normal Order Reduction

The *dynamics* of the λ-calculus.

Examples $(\lambda x \cdot x)y \to y$

$(\lambda x \cdot y)z$

Normal Order Reduction

The *dynamics* of the λ-calculus.

Examples $(\lambda x \cdot x)y \to y$

$(\lambda x \cdot y)z \to y$

Normal Order Reduction

The *dynamics* of the λ-calculus.

Examples $(\lambda x \cdot x)y \to y$

$(\lambda x \cdot y)z \to y$

$(\lambda x \cdot \lambda y \cdot yx)(\lambda a \cdot b)(\lambda z \cdot z)$

Normal Order Reduction

The *dynamics* of the λ-calculus.

Examples $(\lambda x \cdot x)y \to y$

$(\lambda x \cdot y)z \to y$

$(\lambda x \cdot \lambda y \cdot yx)(\lambda a \cdot b)(\lambda z \cdot z) \to \lambda a \cdot b$

Normal Order Reduction

The *dynamics* of the λ-calculus.

Examples $(\lambda x \cdot x)y \to y$

$(\lambda x \cdot y)z \to y$

$(\lambda x \cdot \lambda y \cdot yx)(\lambda a \cdot b)(\lambda z \cdot z) \to \lambda a \cdot b$

$(\lambda x \cdot (\lambda y \cdot k)x)((\lambda z \cdot zz)(\lambda z \cdot zz))$

Normal Order Reduction

The *dynamics* of the λ-calculus.

Examples $(\lambda x \cdot x)y \to y$

$(\lambda x \cdot y)z \to y$

$(\lambda x \cdot \lambda y \cdot yx)(\lambda a \cdot b)(\lambda z \cdot z) \to \lambda a \cdot b$

$(\lambda x \cdot (\lambda y \cdot k)x)((\lambda z \cdot zz)(\lambda z \cdot zz)) \to k$

Normal Order Reduction

The *dynamics* of the λ-calculus.

Examples $(\lambda x \cdot x)y \to y$

$(\lambda x \cdot y)z \to y$

$(\lambda x \cdot \lambda y \cdot yx)(\lambda a \cdot b)(\lambda z \cdot z) \to \lambda a \cdot b$

$(\lambda x \cdot (\lambda y \cdot k)x)((\lambda z \cdot zz)(\lambda z \cdot zz)) \to k$

Some caveats: variable capture, order of evaluation makes a difference

$$\lambda x \cdot x$$

THE IDENTITY

$$\lambda x \cdot xx$$

THE MOCKINGBIRD

A relaxation of notation

We abbreviate $\lambda x_1 \cdot \ldots \cdot \lambda x_n \cdot M$ to $\lambda x_1 \ldots x_n \cdot M$.

A relaxation of notation

We abbreviate $\lambda x_1 \cdot \ldots \cdot \lambda x_n \cdot M$ to $\lambda x_1 \ldots x_n \cdot M$.

For example, $\lambda x \cdot \lambda y \cdot xy$ becomes $\lambda xy \cdot xy$.

A relaxation of notation

We abbreviate $\lambda x_1 \cdot \ldots \cdot \lambda x_n \cdot M$ to $\lambda x_1 \ldots x_n \cdot M$.

For example, $\lambda x \cdot \lambda y \cdot xy$ becomes $\lambda xy \cdot xy$.

This is only notational relaxation, everything is still unary!

$$\lambda xy \cdot x$$

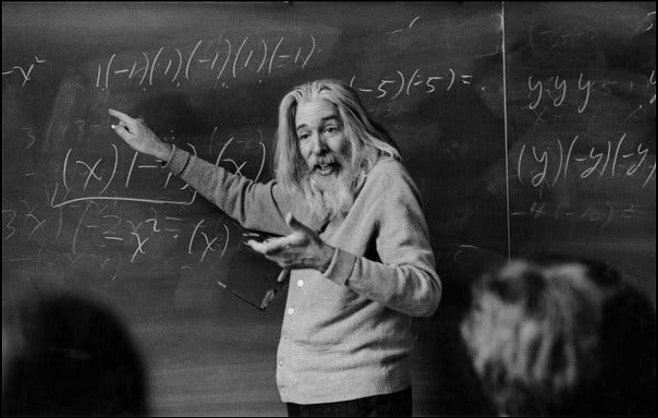
THE KESTREL

$$\lambda xy \cdot y$$

THE KITE

Why all these birds?

Why all these birds?



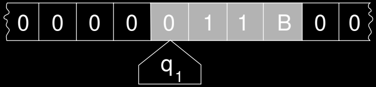
Raymond Smullyan (1919–2017)

$$\lambda fxy \cdot fyx$$

THE CARDINAL

A bit more about Turing...

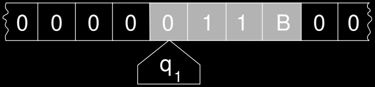
A *Turing machine* is a mathematical object $(Q, \Gamma, b, \delta, q_0, F)$ which represents a hypothetical "machine".



Real computers are basically an attempt at creating a physical model.

A bit more about Turing...

A *Turing machine* is a mathematical object $(Q, \Gamma, b, \delta, q_0, F)$ which represents a hypothetical “machine”.



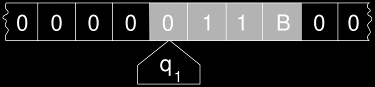
Real computers are basically an attempt at creating a physical model.

Ladder of abstraction:

Real computers

A bit more about Turing...

A *Turing machine* is a mathematical object $(Q, \Gamma, b, \delta, q_0, F)$ which represents a hypothetical "machine".



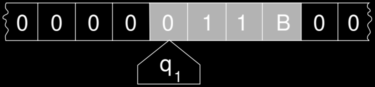
Real computers are basically an attempt at creating a physical model.

Ladder of abstraction:

Real computers $\rightarrow$ machine code

A bit more about Turing...

A *Turing machine* is a mathematical object $(Q, \Gamma, b, \delta, q_0, F)$ which represents a hypothetical "machine".



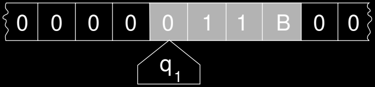
Real computers are basically an attempt at creating a physical model.

Ladder of abstraction:

Real computers $\rightarrow$ machine code $\rightarrow$ assembly language

A bit more about Turing. . .

A *Turing machine* is a mathematical object $(Q, \Gamma, b, \delta, q_0, F)$ which represents a hypothetical "machine".



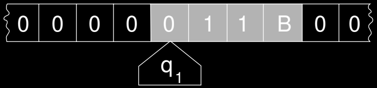
Real computers are basically an attempt at creating a physical model.

Ladder of abstraction:

Real computers $\rightarrow$ machine code $\rightarrow$ assembly language $\rightarrow$ low-level languages

A bit more about Turing. . .

A *Turing machine* is a mathematical object $(Q, \Gamma, b, \delta, q_0, F)$ which represents a hypothetical “machine”.



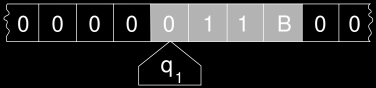
Real computers are basically an attempt at creating a physical model.

Ladder of abstraction:

Real computers $\rightarrow$ machine code $\rightarrow$ assembly language $\rightarrow$ low-level languages $\rightarrow$ high-level languages

A bit more about Turing. . .

A *Turing machine* is a mathematical object $(Q, \Gamma, b, \delta, q_0, F)$ which represents a hypothetical "machine".



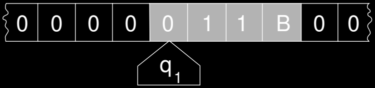
Real computers are basically an attempt at creating a physical model.

Ladder of abstraction:

Real computers → machine code → assembly language → low-level languages → high-level languages → functional programming

A bit more about Turing...

A *Turing machine* is a mathematical object $(Q, \Gamma, b, \delta, q_0, F)$ which represents a hypothetical "machine".



Real computers are basically an attempt at creating a physical model.

Ladder of abstraction:

Real computers $\rightarrow$ machine code $\rightarrow$ assembly language $\rightarrow$ low-level languages $\rightarrow$ high-level languages $\rightarrow$ functional programming $= \lambda$

We can do everything with λ

We can do EVERYTHING with λ

We can do EVERYTHING with λ

Let's start with logic...

True

False

Not

And

Or

TRUE	K	$\lambda xy \cdot x$
FALSE		
NOT		
AND		
OR		

TRUE	K	$\lambda xy \cdot x$
FALSE	KI	$\lambda xy \cdot y$
NOT		
AND		
OR		

TRUE	K	$\lambda xy \cdot x$
FALSE	KI	$\lambda xy \cdot y$
NOT	C	$\lambda fxy \cdot fyx$
AND		
OR		

TRUE	K	$\lambda xy \cdot x$
FALSE	KI	$\lambda xy \cdot y$
NOT	C	$\lambda fxy \cdot fyx$
AND		$\lambda xy \cdot xy\mathrm{F}$
OR		

TRUE	K	$\lambda xy \cdot x$
FALSE	KI	$\lambda xy \cdot y$
NOT	C	$\lambda fxy \cdot fyx$
AND		$\lambda xy \cdot xy\mathrm{F}$
OR		$\lambda xy \cdot x\mathrm{T}y$

True	K	$\lambda xy \cdot x$
False	KI	$\lambda xy \cdot y$
Not	C	$\lambda fxy \cdot fyx$
And		$\lambda xy \cdot xy\mathrm{F}$
Or		$\lambda xy \cdot x\mathrm{T}y$
Iff		$\lambda xy \cdot xy(\textsc{Not}\ y)$

What else can we do?

Numbers:

Zero	$\lambda fx \cdot x = \mathrm{F}$
One	$\lambda fx \cdot fx$
Two	$\lambda fx \cdot f(fx)$
Three	$\lambda fx \cdot f(f(fx))$
Four	$\lambda fx \cdot f(f(f(fx)))$
$\vdots$	$\vdots$

In general, the λ-calculus representation of n takes a function f and applies it to its second argument x, n times.

What else can we do?

Arithmetic:

Succ	$\lambda nfx \cdot f(nfa)$
Add	$\lambda nm \cdot n \text{ Succ } m$
Mul	$\lambda nmf \cdot n(mf)$
Pow	$\lambda nm \cdot mn$
ZeroQ	$\lambda n \cdot n\,(\mathrm{K}\,\mathrm{F})\,\mathrm{T}$
Pre	$\lambda n{\cdot}n(\lambda g{\cdot}\text{ZeroQ}(g \text{ One})\ \mathrm{I}\ (\text{Mul Succ } g))(\mathrm{K} \text{ Zero})\ \text{Zero}$
Sub	$\lambda nm \cdot m \text{ Pre } n$
Leq	$\lambda nm \cdot \text{ZeroQ}(\text{Sub } n\ m)$
Eq	$\lambda nm \cdot \text{And } (\text{Leq } n\ m)(\text{Leq } m\ n)$

What else can we do?

Recursion / looping:

$$\lambda f \cdot (\lambda x \cdot f(xx))(\lambda x \cdot f(xx))$$

THE Y COMBINATOR

Other Interesting Stuff

- All λ combinators can be made up by composing just two others: the Kestrel K (which we've seen), and the Starling S, which is $\lambda xyz \cdot (xz)(yz)$.

  `http://www.angelfire.com/tx4/cus/combinator/birds.html`

Other Interesting Stuff

- All λ combinators can be made up by composing just two others: the Kestrel K (which we've seen), and the Starling S, which is $\lambda xyz \cdot (xz)(yz)$.

  `http://www.angelfire.com/tx4/cus/combinator/birds.html`

  Alex Farrugia wrote a great article series on Quora explaining the SK calculus.

  `https://bit.ly/2PxdQLi`

Other Interesting Stuff

- All λ combinators can be made up by composing just two others: the Kestrel K (which we've seen), and the Starling S, which is $\lambda xyz \cdot (xz)(yz)$.

  http://www.angelfire.com/tx4/cus/combinator/birds.html

  Alex Farrugia wrote a great article series on Quora explaining the SK calculus.

  https://bit.ly/2PxdQLi

- To Mock a Mockingbird is a must-have!

  https://www.amazon.co.uk/Mock-Mockingbird-Other-Logic-Puzzles/dp/0192801422

Thank you!

Luke Collins
luke.collins@um.edu.mt

Malta Mathematics Society
University of Malta