What is the Lambda Calculus?

1: λ-expressions

Definition: A λ-expression is a string of one of the following forms:

x	(or any other single symbol) some variable
$\lambda x.M$	where M is a λ-expression (function abstraction)
AB	where A and B are λ-expressions (function application)

Notes: function application is left-associative: $ABC \equiv (AB)C$.
We can add parentheses for clarity or to provoke right-associative behavior: $ABC \not\equiv A(BC)$.

Examples: $\lambda x.x$ $\quad \lambda x.y$ $\quad \lambda x.\lambda y.xy$ $\quad (\lambda x.(\lambda y.xy))(\lambda x.y)(\lambda x.x)$ $\quad \lambda x.(\lambda y.(\lambda z.xzyz))$

Notation: Often $\lambda x(\lambda y.M)$ is shortened to $\lambda xy.M$. However, we should keep in mind that there are actually two nested λ-expressions in $\lambda xy.M$.

2: λ-calculus operations and β-normal form

We use the notation $[y/x]$ to denote substitution of all instances of x in a string for y. For example, $[w/z]z \equiv w$, $[w/z]xzy \equiv xwy$, and $[a/b](\lambda x.b) \equiv \lambda x.a$. A λ-expression can be changed by one of the two operations:

$\lambda x.M \to \lambda y.[y/x]M$	α-conversion: simply to avoid name collisions
$(\lambda x.M)A \to [A/x]M$	β-reduction: computation of a function application

Examples: $\lambda x.(\lambda y.xy) \to \lambda w.(\lambda y.wy)$ $\quad \lambda x.(\lambda yz.zyx) \to \lambda x.(\lambda yw.wyx)$ $\quad \lambda x.((\lambda y.y)x) \to \lambda x.x$
$(\lambda fx.f(f(fx)))gy \to (\lambda x.g(g(gx)))y \to g(g(gy))$ $\quad (\lambda x.xx)(\lambda x.xx) \to (\lambda x.xx)(\lambda x.xx)$ $\quad (\lambda x.y)z \to y$

Definition: A λ-expression is in **β-normal form** if no β-reduction operation can be performed. For example, $\lambda x.x$ and y are in β-normal form while $(\lambda x.x)y$ is not because $(\lambda x.x)y \to [y/x]x \equiv y$ via β-reduction.

Definition: A λ-expression **halts** if, after a finite number of operations, it reaches a β-normal form. For example, $(\lambda x.x)(\lambda x.x)$ halts while $(\lambda x.xx)(\lambda x.xx)$ doesn't.

Notes: When using α-conversion, there must not be any y in M (i.e. we cannot create a name collision).
It is often reasonable to think of a λ-expression in β-normal form as a function (algorithm) acting on a λ-expression.

3: Boolean algebra in the λ-calculus

We can define the Boolean values "True" and "False" in the following way:

$$\mathbf{T} := \lambda xy.x \qquad \mathbf{F} := \lambda xy.y$$

And we can define logical operations as follows:

$\vee := \lambda xy.x\mathbf{T}y$	Logical "or"
$\wedge := \lambda xy.xy\mathbf{F}$	Logical "and"
$\neg := \lambda x.x\mathbf{FT}$	Logical "not"

For example: $\neg\mathbf{T} \to \mathbf{TFF} \to \mathbf{F}$ $\quad \wedge\mathbf{TF} \to \mathbf{TFF} \to \mathbf{F}$ $\quad \vee\mathbf{TF} \to \mathbf{TTF} \to \mathbf{T}$

4: Arithmetic in the λ-calculus (Church numerals)

We can define the natural numbers in the following way:

$$\mathbf{0} := \lambda fx.x \qquad \mathbf{1} := \lambda fx.fx \qquad \mathbf{2} := \lambda fx.f(fx) \qquad \mathbf{3} := \lambda fx.f(f(fx)) \qquad \ldots$$

Notice that $\mathbf{n}f$ β-reduces to a function which applies f n times to its argument.

We can define some mathematical operations on the natural numbers as follows:

$S := \lambda n.(\lambda fx.nf(fx))$	Successor function
$+ := \lambda nm.nSm$	Addition
$\times := \lambda nm.n(+m)\mathbf{0}$	Multiplication

For example: $S\mathbf{2} \to \lambda fx.\mathbf{2}f(fx) \to \lambda fx.f(f(fx)) \equiv \mathbf{3}$ $\quad +(\mathbf{3})(\mathbf{2}) \to \mathbf{3}S\mathbf{2} \to S(S(S\mathbf{2})) \to S(S\mathbf{3}) \to S\mathbf{4} \to \mathbf{5}$ $\quad \times(\mathbf{3})(\mathbf{2}) \to \mathbf{3}(+\mathbf{2})\mathbf{0} \to (+\mathbf{2})((+\mathbf{2})((+\mathbf{2})\mathbf{0})) \to (+\mathbf{2})((+\mathbf{2})\mathbf{2}) \to (+\mathbf{2})\mathbf{4} \to \mathbf{6}$

It is often useful to have an operator which checks if a given number is zero, returning a Boolean value:

$$Z := \lambda n.n\mathbf{F}\neg\mathbf{F}$$

For example: $Z\mathbf{0} \to \mathbf{0F}\neg\mathbf{F} \to \neg\mathbf{F} \to \mathbf{T}$ $\quad Z\mathbf{1} \to \mathbf{1F}\neg\mathbf{F} \to \mathbf{F}\neg\mathbf{F} \to \mathbf{F}$ $\quad Z\mathbf{3} \to \mathbf{3F}\neg\mathbf{F} \to \mathbf{F}(\mathbf{F}(\mathbf{F}\neg))\mathbf{F} \to \mathbf{F}$

5: Computable functions

Definitions:

A function $f : \mathbb{N}^k \to \mathbb{N}$ is λ-*computable* if there is a λ-expression F so that $F\mathbf{n}_1 \ldots \mathbf{n}_k \to^* \mathbf{m}$ iff $f(n_1, \ldots, n_k) = m$.
A set $A \subseteq \mathbb{N}^k$ is λ-*recognizable* if there is a λ-expression L so that $L\mathbf{n}_1 \ldots \mathbf{n}_k \to^* \mathbf{T}$ iff $(n_1, \ldots, n_k) \in A$.
If there is an L as above so that $L\mathbf{n}_1 \ldots \mathbf{n}_k$ halts for every $(n_1, \ldots, n_k) \in \mathbb{N}^k$, then A is λ-*decidable*.

6: Encodings of λ-expressions

We will need to slightly restrict our definition of a λ-expression by only allowing variable names to be x or x followed by any number of $'$s: x, x', x'', x''', etc. We will also require that λ-expressions be written out "in full" (i.e. in terms of only the six basic symbols required, which are listed in the table below, and with only one variable per λ). Now we can encode any λ-expression by a natural number by translating symbols to digits as follows:

Symbol	λ	.	x	$'$	(	)
Digit	1	2	3	4	5	6

So, for example, our + algorithm defined earlier, when written out in our more restrictive notation, looks like this:

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

Which means its encoding, denoted $<+>$, is 132134235134421344421344442344344453444344446634.

7: The Halting problem

Given any λ-expression M and Church numeral $\mathbf{w}$, can we decide if $M\mathbf{w}$ halts? More precisely, is the set

$$\{(< M >, w) \mid M\mathbf{w} \text{ halts}\} \subseteq \mathbb{N}^2$$

λ-decidable? As it turns out, the answer is no. If some λ-expression H λ-decides this set then we can define a new λ-expression $G := \lambda m.Hmm((\lambda x.xx)(\lambda x.xx))\mathbf{1}$. Now, does $G < G >$ halt? Well, if $G < G >$ halts then we have $H < G >< G > \to^* \mathbf{T}$, so $G < G > \to H < G >< G > ((\lambda x.xx)(\lambda x.xx))\mathbf{1} \to^* \mathbf{T}((\lambda x.xx)(\lambda x.xx))\mathbf{1} \to (\lambda x.xx)(\lambda x.xx)$, which of course does not halt. On the other hand, if $G < G >$ does not halt then $H < G >< G > \to^* \mathbf{F}$ since H λ-decides the halting set. this means $G < G > \to H < G >< G > ((\lambda x.xx)(\lambda x.xx))\mathbf{1} \to^* \mathbf{F}((\lambda x.xx)(\lambda x.xx))\mathbf{1} \to \mathbf{1}$ which is in β-normal form, showing that $G < G >$ halted. This contradiction shows that G (and thus H) cannot exist.

References

[1] Church, A. (1932). "*A* set of *P*ostulates for the foundation of logic". Annals of Mathematics. Series 2. 33 (2): 346–366. JSTOR 1968337. doi:10.2307/1968337.

[2] Rojas, R. (2015). A tutorial introduction to the lambda calculus. arXiv preprint arXiv:1503.09060.

[3] Sipser, M. (2012). Introduction to the Theory of Computation. Cengage Learning.

[4] Turing, A. M. (December 1937). "Computability and λ-Definability". The Journal of Symbolic Logic. 2 (4): 153–163. JSTOR 2268280. doi:10.2307/2268280.