6

Lambda-Calculus

Terms

A term of the λ-calculus is either:

- a variable $x, y, z, \ldots$
- an abstraction $\langle x \to M \rangle$ where x is a variable and M a term
- an application $M\ N$ where M, N are terms

Examples: $x \quad \langle x \to x \rangle \quad \langle y \to \langle x \to x \rangle\rangle \quad \langle x \to x\ \langle y \to y \rangle\rangle$

Intuitively, $\langle x \to M \rangle$ represents a function with parameter x and returning M, while $M\ N$ represents an application of function M to argument N. The simplification rules below will enforce this interpretation.[1]

Just like elsewhere in mathematics, we will use parentheses freely to group terms together to affect or just clarify the order of applications. Application $M\ N$ is a binary operation that associates to the left, so that writing $M\ N\ P$ is the same as writing $(M\ N)\ P$. If you want $M\ (N\ P)$ (which means something different) then you need to use parentheses explicitly.

The *scope* of x in $\langle x \to M \rangle$ is all of M. An occurrence of a variable is said to be *bound* if it occurs in the scope of an abstraction with that variable as a parameter. More precisely, it is bound to the nearest enclosing abstraction. An occurrence of a variable is said to be *free* if it is not bound.

[1]The standard presentation of the λ-calculus uses notation $\lambda x.M$ for$\langle x \to M \rangle$, hence the name.

Examples: y is free in $\langle x \to y\rangle$; the first occurrence of x is free in $\langle y \to x\ \langle x \to x\rangle\rangle$ while the second is not; z is bound in $\langle z \to \langle x \to z\rangle\rangle$.

Bound variables can be renamed without affecting the meaning of term. Intuitively, $\langle x \to x\rangle$ and $\langle y \to y\rangle$ represent the same function, the identity function. That we happen to call the parameter x in the first and y in the second is pretty irrelevant. Two terms are α-equivalent when they are equal up to renaming of some bound variables. Thus, $\langle x \to x\ z\rangle$ and $\langle y \to y\ z\rangle$ are α-equivalent. Be careful that your renaming does not *capture* a free occurrence of a variable. For example, $\langle x \to x\ z\rangle$ and $\langle z \to z\ z\rangle$ are *not* α-equivalent. They represent different functions.

We will generally identify α-equivalent terms.

Substitution

An important operation is that of substituting a term N for a variable x inside another term M, written $M\{N/x\}$. It is defined formally as

$$x\{N/x\} = N$$

$$y\{N/x\} = y \quad \text{if } x \neq y$$

$$(M_1\ M_2)\{N/x\} = M_1\{N/x\}\ M_2\{N/x\}$$

$$\langle y \to M\rangle\{N/x\} = \langle y \to M\{N/x\}\rangle \quad \text{if } y \text{ is not free in } N$$

In the last case, if $x = y$ or if y *is* free in N, we can always find a term $\langle z \to M'\rangle$ that is α-equivalent to $\langle y \to M\rangle$ and such that $x \neq z$ and z is not free in N to perform the sustitution.

(Because we avoid capturing free variables, this form of substitution is called a *capture-avoiding substitution*.)

Simplification Rules

The main simplification rule is:

$$\langle x \to M\rangle\ N = M\{N/x\}$$

A term of the form $\langle x \to M\rangle\ N$ is called a *redex*.

Simplification can occur within the context of a larger term, of course, leading to the following three additional simplification rules:

$$M\ P = N\ P \quad \text{if } M = N$$

$$P\ M = P\ N \quad \text{if } M = N$$

$$\langle x \to M\rangle = \langle x \to N\rangle \quad \text{if } M = N$$

Examples:

$$\begin{aligned} \langle x \to x\rangle\ \langle y \to y\rangle &= x\{\langle y \to y\rangle / x\} \\ &= \langle y \to y\rangle \end{aligned}$$

$$\begin{aligned} ((\langle x \to \langle y \to x\rangle\rangle\ z_1)\ z_2 &= (\langle y \to x\rangle\{z_1/x\})\ z_2 \\ &= \langle y \to z_1\rangle\ z_2 \\ &= z_1\{z_2/y\} \\ &= z_1 \end{aligned}$$

$$\begin{aligned} ((\langle x \to \langle y \to y\rangle\rangle\ \langle z \to z\rangle)\ \langle x \to \langle y \to x\rangle\rangle &= \langle y \to y\rangle\{\langle z \to z\rangle/x\}\ \langle x \to \langle y \to x\rangle\rangle \\ &= \langle y \to y\rangle\ \langle x \to \langle y \to x\rangle\rangle \\ &= y\{\langle x \to \langle y \to x\rangle\rangle/y\} \\ &= \langle x \to \langle y \to x\rangle\rangle \end{aligned}$$

From now on, I will skip the explicit substitution step when showing simplifications.

A term is in *normal form* if it has no redex (and thus cannot be simplified any further).

Not every term can be simplified to a normal form:

$$\begin{aligned} \langle x \to x\ x\rangle\ \langle x \to x\ x\rangle &= \langle x \to x\ x\rangle\ \langle x \to x\ x\rangle \\ &= \langle x \to x\ x\rangle\ \langle x \to x\ x\rangle \end{aligned}$$

$$= \quad \ldots$$

There can be more than one redex in a term, meaning that there may be more than one applicable simplification. For instance, in the term $(\langle x \rightarrow x\rangle\ \langle y \rightarrow x\rangle)\ (\langle x \rightarrow \langle y \rightarrow x\rangle\rangle\ z_1\ z_2)$. A property of the λ-calculus is that all the ways to simplify a term down to a normal form yield the same normal form (up to renaming of bound variables). This is called the *Church-Rosser property*. It says that the order in which we perform simplifications to reach a normal form is not important.

In practice, one often imposes an order in which to apply simplifications to avoid nondeterminisn. The *normal-order strategy*, which always simplifies the leftmost and outermost redex, is guaranteed to find a normal form if one exists.

To simplify the presentation of more complex terms, we introduce a convenient abbreviation. We write

$$\begin{aligned}
\langle x_1\ x_2 \rightarrow M\rangle &= \langle x_1 \rightarrow \langle x_2 \rightarrow M\rangle\rangle \\
\langle x_1\ x_2\ x_3 \rightarrow M\rangle &= \langle x_1 \rightarrow \langle x_2 \rightarrow \langle x_3 \rightarrow M\rangle\rangle\rangle \\
\langle x_1\ x_2\ x_3\ x_4 \rightarrow M\rangle &= \langle x_1 \rightarrow \langle x_2 \rightarrow \langle x_3 \rightarrow \langle x_4 \rightarrow M\rangle\rangle\rangle\rangle \\
&\vdots
\end{aligned}$$

Working through the abbreviations, this means that we have simplifications:

$$\begin{aligned}
\langle x_1\ x_2 \rightarrow M\rangle\ N &= \langle x_2 \rightarrow M\{N/x_1\}\rangle \\
\langle x_1\ x_2\ x_3 \rightarrow M\rangle\ N &= \langle x_2\ x_3 \rightarrow M\{N/x_1\}\rangle \\
&\vdots
\end{aligned}$$

Encoding Booleans

Even though the λ-calculus only has variables and functions, that's enough to encode all traditional data types.

Here's one way to encode Boolean values (due to Church):

$$\begin{aligned}
\textbf{true} &= \langle x\ y \rightarrow x\rangle \\
\textbf{false} &= \langle x\ y \rightarrow y\rangle
\end{aligned}$$

In what sense are these encodings of Boolean values? Booleans are useful because they allow you to select one branch or the other of a conditional expression.

$$\textbf{if} = \langle c\ x\ y \rightarrow c\ x\ y \rangle$$

The trick is that when B simplifies to either **true** or **false**, then **if** $B\ M\ N$ simplifies either to M or to N, respectively:

If B = **true**, then

$$\begin{aligned} \textbf{if}\ B\ M\ N = &\ B\ M\ N \\ = &\ \textbf{true}\ M\ N \\ = &\ \langle x\ y \rightarrow x \rangle\ M\ N \\ = &\ \langle y \rightarrow M \rangle\ N \\ = &\ M \end{aligned}$$

while if B = **false**, then

$$\begin{aligned} \textbf{if}\ B\ M\ N = &\ B\ M\ N \\ = &\ \textbf{false}\ M\ N \\ = &\ \langle x\ y \rightarrow y \rangle\ M\ N \\ = &\ \langle y \rightarrow y \rangle\ N \\ = &\ N \end{aligned}$$

Of course, these show that **if** is not strictly necessary. You should convince yourself that **true** $M\ N = M$ and that **false** $M\ N = N$.

We can define logical operators:

$$\begin{aligned} \textbf{and} &= \langle m\ n \rightarrow m\ n\ m \rangle \\ \textbf{or} &= \langle m\ n \rightarrow m\ m\ n \rangle \\ \textbf{not} &= \langle m \rightarrow \langle x\ y \rightarrow m\ y\ x \rangle \rangle \end{aligned}$$

Thus, for example:

$$\begin{aligned} \textbf{and true false} = &\ \langle m\ n \rightarrow m\ n\ m \rangle\ \textbf{true false} \\ = &\ \langle n \rightarrow \textbf{true}\ n\ \textbf{true} \rangle\ \textbf{false} \\ = &\ \textbf{true false true} \\ = &\ \langle x\ y \rightarrow x \rangle\ \textbf{false true} \end{aligned}$$

$$
\begin{aligned}
&= \langle y \to \textbf{false} \rangle \; \textbf{true} \\
&= \textbf{false}
\end{aligned}
$$

$$
\begin{aligned}
\textbf{not false} &= \langle m \to \langle x\ y \to m\ y\ x \rangle\rangle \; \textbf{false} \\
&= \langle x\ y \to \textbf{false}\ y\ x \rangle \\
&= \langle x\ y \to \langle u\ v \to v \rangle\ y\ x \rangle \\
&= \langle x\ y \to \langle v \to v \rangle\ x \rangle \\
&= \langle x\ y \to x \rangle \\
&= \textbf{true}
\end{aligned}
$$

Encoding Natural Numbers

Here is an encoding of natural numbers, again due to Church (hence the name, Church numerals):

$$
\begin{aligned}
\mathbf{0} &= \langle f\ x \to x \rangle \\
\mathbf{1} &= \langle f\ x \to f\ x \rangle \\
\mathbf{2} &= \langle f\ x \to f\ (f\ x) \rangle \\
\mathbf{3} &= \langle f\ x \to f\ (f\ (f\ x)) \rangle \\
\mathbf{4} &= \quad \ldots
\end{aligned}
$$

In general, natural number n is encoded as $\langle f\ x \to \underbrace{f\ (f\ (f\ (\ldots (f}_{n \text{ times}}\ x)))) \rangle$.

Successor operation:

$$\textbf{succ} = \langle n \to \langle f\ x \to n\ f\ (f\ x) \rangle\rangle$$

$$
\begin{aligned}
\textbf{succ 1} &= \langle n \to \langle f\ x \to n\ f\ (f\ x) \rangle\rangle\ \langle f\ x \to f\ x \rangle \\
&= \langle f\ x \to \langle f\ x \to f\ x \rangle\ f\ (f\ x) \rangle \\
&= \langle f\ x \to \langle x \to f\ x \rangle\ (f\ x) \rangle \\
&= \langle f\ x \to f\ (f\ x) \rangle \\
&= \mathbf{2}
\end{aligned}
$$

Other operations:

$$\begin{aligned}
\textbf{plus} &= \langle m\ n \rightarrow m\ \textbf{succ}\ n \rangle \\
\textbf{times} &= \langle m\ n \rightarrow \langle f\ x \rightarrow m\ (n\ f)\ x \rangle\rangle \\
\textbf{iszero}? &= \langle n \rightarrow n\ \langle x \rightarrow \textbf{false} \rangle\ \textbf{true} \rangle
\end{aligned}$$

$$\begin{aligned}
\textbf{plus 2 1} &= \langle m\ n \rightarrow m\ \textbf{succ}\ n \rangle\ \textbf{2 1} \\
&= \langle n \rightarrow \textbf{2 succ}\ n \rangle\ \textbf{1} \\
&= \textbf{2 succ 1} \\
&= \langle f\ x \rightarrow f\ (f\ x) \rangle\ \textbf{succ 1} \\
&= \langle x \rightarrow \textbf{succ}\ (\textbf{succ}\ x) \rangle\ \textbf{1} \\
&= \textbf{succ}\ (\textbf{succ 1}) \\
&= \textbf{succ}\ (\langle n \rightarrow \langle f\ x \rightarrow n\ f\ (f\ x) \rangle\rangle\ \textbf{1}) \\
&= \textbf{succ}\ \langle f\ x \rightarrow \textbf{1}\ f\ (f\ x) \rangle \\
&= \textbf{succ}\ \langle f\ x \rightarrow \langle f\ x \rightarrow f\ x \rangle\ f\ (f\ x) \rangle \\
&= \textbf{succ}\ \langle f\ x \rightarrow \langle x \rightarrow f\ x \rangle\ (f\ x) \rangle \\
&= \textbf{succ}\ \langle f\ x \rightarrow f\ (f\ x) \rangle \\
&= \langle n \rightarrow \langle f\ x \rightarrow n\ f\ (f\ x) \rangle\rangle\ \langle f\ x \rightarrow f\ (f\ x) \rangle \\
&= \langle f\ x \rightarrow \langle f\ x \rightarrow f\ (f\ x) \rangle\ f\ (f\ x) \rangle \\
&= \langle f\ x \rightarrow \langle x \rightarrow f\ (f\ x) \rangle\ (f\ x) \rangle \\
&= \langle f\ x \rightarrow f\ (f\ (f\ x)) \rangle \\
&= \textbf{3}
\end{aligned}$$

$$\begin{aligned}
\textbf{times 2 3} &= \langle m\ n \rightarrow \langle f\ x \rightarrow m\ (n\ f)\ x \rangle\rangle\ \textbf{2 3} \\
&= \langle n \rightarrow \langle f\ x \rightarrow \textbf{2}\ (n\ f)\ x \rangle\rangle\ \textbf{3} \\
&= \langle f\ x \rightarrow \textbf{2}\ (\textbf{3}\ f)\ x \rangle \\
&= \langle f\ x \rightarrow \textbf{2}\ (\langle f\ x \rightarrow f\ (f\ (f\ x)) \rangle\ f)\ x \rangle \\
&= \langle f\ x \rightarrow \textbf{2}\ \langle x \rightarrow f\ (f\ (f\ x)) \rangle\ x \rangle \\
&= \langle f\ x \rightarrow \langle f\ x \rightarrow f\ (f\ x) \rangle\ \langle x \rightarrow f\ (f\ (f\ x)) \rangle\ x \rangle \\
&= \langle f\ x \rightarrow \langle x \rightarrow \langle x \rightarrow f\ (f\ (f\ x)) \rangle\ (\langle x \rightarrow f\ (f\ (f\ x)) \rangle\ x) \rangle\ x \rangle \\
&= \langle f\ x \rightarrow \langle x \rightarrow \langle x \rightarrow f\ (f\ (f\ x)) \rangle\ (f\ (f\ (f\ x))) \rangle\ x \rangle
\end{aligned}$$

$$\begin{aligned}
&= \langle f\ x \to \langle x \to f\ (f\ (f\ (f\ (f\ (f\ x)))))\rangle\ x\rangle \\
&= \langle f\ x \to f\ (f\ (f\ (f\ (f\ (f\ x)))))\rangle \\
&= \mathbf{6}
\end{aligned}$$

$$\begin{aligned}
\mathbf{iszero?}\ \mathbf{0} &= \langle n \to n\ \langle x \to \mathbf{false}\rangle\ \mathbf{true}\rangle\ \langle f\ x \to x\rangle \\
&= \langle f\ x \to x\rangle\ \langle x \to \mathbf{false}\rangle\ \mathbf{true} \\
&= \langle x \to x\rangle\ \mathbf{true} \\
&= \mathbf{true}
\end{aligned}$$

$$\begin{aligned}
\mathbf{iszero?}\ \mathbf{2} &= \langle n \to n\ \langle x \to \mathbf{false}\rangle\ \mathbf{true}\rangle\ \langle f\ x \to f\ (f\ x)\rangle \\
&= \langle f\ x \to f\ (f\ x)\rangle\ \langle x \to \mathbf{false}\rangle\ \mathbf{true} \\
&= \langle x \to \langle x \to \mathbf{false}\rangle\ (\langle x \to \mathbf{false}\rangle\ x)\rangle\ \mathbf{true} \\
&= \langle x \to \langle x \to \mathbf{false}\rangle\ \mathbf{false}\rangle\ \mathbf{true} \\
&= \langle x \to \mathbf{false}\rangle\ \mathbf{true} \\
&= \mathbf{false}
\end{aligned}$$

(An alternative way to define **times** is as $\langle m\ n \to m\ (\mathbf{plus}\ n)\ \mathbf{0}\rangle$. Check that **times 2 3** = **6** with this definition.)

Defining a predecessor function is a bit more challenging. Predecessor takes a nonzero natural number n and returning $n - 1$. There are several ways of defining such a function; here is probably the simplest:

$$\mathbf{pred} = \langle n \to \langle f\ x \to n\ \langle g\ h \to h\ (g\ f)\rangle\ \langle u \to x\rangle\ \langle u \to u\rangle\rangle\rangle$$

$$\begin{aligned}
\mathbf{pred}\ \mathbf{2} &= \langle n \to \langle f\ x \to n\ \langle g\ h \to h\ (g\ f)\rangle\ \langle u \to x\rangle\ \langle u \to u\rangle\rangle\rangle\ \langle f\ x \to f\ (f\ x)\rangle \\
&= \langle f\ x \to \langle f\ x \to f\ (f\ x)\rangle\ \langle g\ h \to h\ (g\ f)\rangle\ \langle u \to x\rangle\ \langle u \to u\rangle\rangle \\
&= \langle f\ x \to \langle x \to \langle g\ h \to h\ (g\ f)\rangle\ (\langle g\ h \to h\ (g\ f)\rangle\ x)\rangle\ \langle u \to x\rangle\ \langle u \to u\rangle\rangle \\
&= \langle f\ x \to \langle g\ h \to h\ (g\ f)\rangle\ (\langle g\ h \to h\ (g\ f)\rangle\ \langle u \to x\rangle)\rangle\ \langle u \to u\rangle \\
&= \langle f\ x \to \langle g\ h \to h\ (g\ f)\rangle\ (\langle h \to h\ (\langle u \to x\rangle\ f)\rangle)\rangle\ \langle u \to u\rangle \\
&= \langle f\ x \to \langle g\ h \to h\ (g\ f)\rangle\ \langle h \to h\ x\rangle\rangle\ \langle u \to u\rangle \\
&= \langle f\ x \to \langle h \to h\ (\langle h \to h\ x\rangle\ f)\rangle\rangle\ \langle u \to u\rangle
\end{aligned}$$

$$\begin{aligned}
&= \langle f\ x \to \langle h \to h\ (f\ x)\rangle\rangle\ \langle u \to u\rangle \\
&= \langle f\ x \to \langle u \to u\rangle\ (f\ x)\rangle \\
&= \langle f\ x \to f\ x\rangle \\
&= \mathbf{1}
\end{aligned}$$

Note that **pred 0** is just **0**:

$$\begin{aligned}
\mathbf{pred\ 0} &= \langle n \to \langle f\ x \to n\ \langle g\ h \to h\ (g\ f)\rangle\ \langle u \to x\rangle\ \langle u \to u\rangle\rangle\rangle\ \langle f\ x \to x\rangle \\
&= \langle f\ x \to \langle f\ x \to x\rangle\ \langle g\ h \to h\ (g\ f)\rangle\ \langle u \to x\rangle\ \langle u \to u\rangle\rangle \\
&= \langle f\ x \to \langle x \to x\rangle\ \langle u \to x\rangle\ \langle u \to u\rangle\rangle \\
&= \langle f\ x \to \langle u \to x\rangle\ \langle u \to u\rangle\rangle \\
&= \langle f\ x \to x\rangle \\
&= \mathbf{0}
\end{aligned}$$

Encoding Pairs

A pair is just a packaging up of two terms in such a way that we can recover the two terms later on.

$$\begin{aligned}
\mathbf{pair} &= \langle x\ y \to \langle s \to s\ x\ y\rangle\rangle \\
\mathbf{first} &= \langle p \to p\ \langle x\ y \to x\rangle\rangle \\
\mathbf{second} &= \langle p \to p\ \langle x\ y \to y\rangle\rangle
\end{aligned}$$

It is easy to check that this works as advertised:

$$\begin{aligned}
\mathbf{first}\ (\mathbf{pair}\ a\ b) &= \langle p \to p\ \langle x\ y \to x\rangle\rangle\ (\langle x\ y \to \langle s \to s\ x\ y\rangle\rangle\ a\ b) \\
&= \langle p \to p\ \langle x\ y \to x\rangle\rangle\ (\langle y \to \langle s \to s\ a\ y\rangle\rangle\ b) \\
&= \langle p \to p\ \langle x\ y \to x\rangle\rangle\ \langle s \to s\ a\ b\rangle \\
&= \langle s \to s\ a\ b\rangle\ \langle x\ y \to x\rangle \\
&= \langle x\ y \to x\rangle\ a\ b \\
&= \langle y \to a\rangle\ b \\
&= a
\end{aligned}$$

$$\mathbf{second}\ (\mathbf{pair}\ a\ b) = \langle p \to p\ \langle x\ y \to y\rangle\rangle\ (\langle x\ y \to \langle s \to s\ x\ y\rangle\rangle\ a\ b)$$

$$
\begin{aligned}
&= \langle p \to p \; \langle x \; y \to y\rangle\rangle \; (\langle y \to \langle s \to s \; a \; y\rangle\rangle \; b) \\
&= \langle p \to p \; \langle x \; y \to y\rangle\rangle \; \langle s \to s \; a \; b\rangle \\
&= \langle s \to s \; a \; b\rangle \; \langle x \; y \to y\rangle \\
&= \langle x \; y \to y\rangle \; a \; b \\
&= \langle y \to y\rangle \; b \\
&= b
\end{aligned}
$$

It is an exercise to extend this encoding to lists, that is, structures that can record an arbitrary number of elements.

Recursion

With conditionals and basic data types, we are very close to having enough to code up a simulator for Turing machines — basically code that replicates the Turing machine simulator that we implemented in OCaml.

All that is missing is a way to do loops. It turns out we can write recursive functions in the λ-calculus, which as we know is sufficient to give us loops.

Consider factorial. Intuitively, we would like to define **fact** by

$$\textbf{fact} = \langle n \to (\textbf{iszero?}\; n)\; \textbf{1}\; (\textbf{times}\; n\; (\textbf{fact}\; (\textbf{pred}\; n)))\rangle \qquad (6.1)$$

The problem is that this is not a valid definition: the right-hand side refers to the term being defined. It is really an *equation*, the same way $x = 3x + 1$ is an equation.

Consider that equation, $x = 3x + 1$. Using algebraic manipulations, we can derive a solution to this equations, $x = -1/2$. So the equation $x = 3x + 1$ indirectly yields a definition of $x = -1/2$ that satisfies the equation.

Intuitively, we want to do the same with equation (6.1). That equation describes a property we need **fact** to have. A solution of that equation will give us a function that satisfies that property, that is, a factorial function. So how can we derive a definition of a function that satisfies (6.1)?

Unfortunately, it is not really possible to perform algebraic manipulations the way we can to solve arithmetic equations. Instead, we'll take a slightly different tack.

Let's return to equation $x = 3x + 1$. Define a function F to capture the right hand side, so that we can write $x = F(x)$ where F is a function such that

$F(x) = 3x + 1$. Call this F the *generating function* of the equation. I claim that a value x_0 is a solution of $x = 3x + 1$ exactly when x_0 is a *fixed point* of F. (A fixed point of a function $f : A \longrightarrow A$ is a value $a \in A$ such that $f(a) = a$.)

> Proof: x_0 is a solution of $x = 3x + 1$ exactly when $x_0 = 3x_0 + 1$ which is just $x_0 = F(x_0$ which is just x_0 being a fixed point of F.

We are going to find a solution to (6.1) by finding a fixed point of its generating function. What is the generating function of (6.1)? Well, we can write (6.1) as

$$\mathbf{fact} = F_{\mathit{fact}}(\mathbf{fact})$$

where F_{fact} is the function such that

$$F_{\mathit{fact}}(f) = \langle n \longrightarrow (\mathbf{iszero?}\ n)\ \mathbf{1}\ (\mathbf{times}\ n\ (f\ (\mathbf{pred}\ n)))\rangle$$

, that is,

$$F_{\mathit{fact}} = \langle f \longrightarrow \langle n \longrightarrow (\mathbf{iszero?}\ n)\ \mathbf{1}\ (\mathbf{times}\ n\ (f\ (\mathbf{pred}\ n)))\rangle\rangle$$

Let's check that if we had a fixed point of F_{fact}, it would indeed act like a factorial function. Let $\mathbf{f}$ be a fixed point of F_{fact}, that is, $F_{\mathit{fact}}(\mathbf{f}) = \mathbf{f}$.

$$\begin{aligned}
\mathbf{f\ 3} &= F_{\mathit{fact}}\ \mathbf{f\ 3} \\
&= \langle f \longrightarrow \langle n \longrightarrow (\mathbf{iszero?}\ n)\ \mathbf{1}\ (\mathbf{times}\ n\ (f\ (\mathbf{pred}\ n)))\rangle\rangle\ \mathbf{f\ 3} \\
&= \langle n \longrightarrow (\mathbf{iszero?}\ n)\ \mathbf{1}\ (\mathbf{times}\ n\ (\mathbf{f}\ (\mathbf{pred}\ n)))\rangle\ \mathbf{3} \\
&= (\mathbf{iszero?\ 3})\ \mathbf{1}\ (\mathbf{times\ 3}\ (\mathbf{f}\ (\mathbf{pred\ 3})) \\
&= \mathbf{times\ 3}\ (\mathbf{f}\ (\mathbf{pred\ 3})) \\
&= \mathbf{times\ 3}\ (\mathbf{f\ 2}) \\
&= \mathbf{times\ 3}\ (F_{\mathit{fact}}\ \mathbf{f\ 2}) \\
&= \mathbf{times\ 3}\ (\mathbf{times\ 2}\ (\mathbf{f\ 1})) \\
&= \mathbf{times\ 3}\ (\mathbf{times\ 2}\ (F_{\mathit{fact}}\ \mathbf{f\ 1})) \\
&= \mathbf{times\ 3}\ (\mathbf{times\ 2}\ (\mathbf{times\ 1}\ (\mathbf{f\ 1}))) \\
&= \mathbf{times\ 3}\ (\mathbf{times\ 2}\ (\mathbf{times\ 1}\ (F_{\mathit{fact}}\ \mathbf{f\ 1}))) \\
&= \mathbf{times\ 3}\ (\mathbf{times\ 2}\ (\mathbf{times\ 1\ 1})) \\
&= \mathbf{6}
\end{aligned}$$

(I coalesced together quite a few sequences of simplification steps in the above, for the sake of space.)

It's not hard to convince yourself that **f** n simplies to the factorial of n, and therefore a fixed point of F_{fact} is indeed the factorial function.

The only thing we need now is to figure out how to find fixed-points in the λ-calculus. The following term Θ does just that, for *any* term of the λ-calculus:

$$\begin{aligned}
\Theta_0 &= \langle x\ y \rightarrow y\ ((x\ x)\ y)\rangle \\
\Theta &= \Theta_0\ \Theta_0
\end{aligned}$$

Theorem: ΘG if a fixed point of G for any term G.

It's an easy check, by simplification:

$$\begin{aligned}
\Theta G = &\quad (\Theta_0\ \Theta_0)\ G \\
= &\quad (\langle x\ y \rightarrow y\ ((x\ x)\ y)\rangle\ G \\
= &\quad \langle y \rightarrow y\ ((\Theta_0\ \Theta_0)\ y)\rangle\ G \\
= &\quad G\ ((\Theta_0\ \Theta_0)\ G) \\
= &\quad G\ (\Theta G)
\end{aligned}$$

Therefore, $G(\Theta G) = YG$, showing that ΘG is a fixed point of G.

Since a fixed point of F_{fact} is the factorial function we're looking for, we can define factorial as:

$$\mathbf{fact} = \Theta F_{\mathit{fact}}$$

We can check directly that **fact 3** = **6**, or that **fact 4** = **24**, or rely on our simplifications above that used the fact that **fact** was a fixed point of F_{fact}.

With basic types, conditionals, and recursion, it is a simple matter to implement a Turing machine simulator, showing the λ-calculus Turing-complete, and therefore able to express all computable decision problems.

Combinatory Logic

A λ-calculus term without any free identifiers is said to be *closed*. Closed terms are also called *combinators*. All of the definitions above — the encoding of Booleans, natural numbers, pairs, and operations on those encodings – are combinators. The simulation of Turing machines can be done all with combinators. Combinators, in a sense, are the interesting terms of the λ-calculus.

Every combinator in the λ-calculus can be rewritten into a term that uses only the following three combinators:

$$\begin{aligned}
\mathbf{I} &= \langle x \rightarrow x \rangle \\
\mathbf{K} &= \langle x\ y \rightarrow x \rangle \\
\mathbf{S} &= \langle x\ y\ z \rightarrow x\ z\ (y\ z) \rangle
\end{aligned}$$

To do this rewriting, we can apply the following transformations, successively, until the whole term is rewritten. The transformations should be applied inside out (from the innermost terms to the outermost terms):

$$\begin{aligned}
\langle x \rightarrow x \rangle &\Longrightarrow \mathbf{I} \\
\langle x \rightarrow M \rangle &\Longrightarrow \mathbf{K}\ M \quad \text{when } x \text{ is not free in } M \\
\langle x \rightarrow M\ N \rangle &\Longrightarrow \mathbf{S}\ \langle x \rightarrow M \rangle\ \langle x \rightarrow N \rangle
\end{aligned}$$

As examples, let's see what some of our encodings rewrite into:

$$\begin{aligned}
\textbf{true} &= \langle x \rightarrow \langle y \rightarrow x \rangle \rangle \\
&= \langle x \rightarrow \mathbf{K}\ x \rangle \\
&= \mathbf{S}\ \langle x \rightarrow \mathbf{K} \rangle\ \langle x \rightarrow x \rangle \\
&= \mathbf{S}\ (\mathbf{K}\ \mathbf{K})\ \mathbf{I}
\end{aligned}$$

$$\begin{aligned}
\textbf{false} &= \langle x \rightarrow \langle y \rightarrow y \rangle \rangle \\
&= \langle x \rightarrow \mathbf{I} \rangle \\
&= \mathbf{K}\ \mathbf{I}
\end{aligned}$$

$$\begin{aligned}
\mathbf{0} &= \langle f \rightarrow \langle x \rightarrow x \rangle \rangle \\
&= \langle f \rightarrow \mathbf{I} \rangle \\
&= \mathbf{K}\ \mathbf{I}
\end{aligned}$$

$$\begin{aligned}
\mathbf{1} &= \langle f \to \langle x \to f\ x\rangle\rangle \\
&= \langle f \to \mathbf{S}\ \langle x \to f\rangle\ \langle x \to x\rangle\rangle \\
&= \langle f \to \mathbf{S}\ (\mathbf{K}\ f)\ \langle x \to x\rangle\rangle \\
&= \langle f \to \mathbf{S}\ (\mathbf{K}\ f)\ \mathbf{I}\rangle \\
&= \mathbf{S}\ \langle f \to \mathbf{S}\ (\mathbf{K}\ f)\rangle\ \langle f \to \mathbf{I}\rangle \\
&= \mathbf{S}\ \langle f \to \mathbf{S}\ (\mathbf{K}\ f)\rangle\ (\mathbf{K}\ \mathbf{I}) \\
&= \mathbf{S}\ (\mathbf{S}\ \langle f \to \mathbf{S}\rangle\ \langle f \to \mathbf{K}\ f\rangle)\ (\mathbf{K}\ \mathbf{I}) \\
&= \mathbf{S}\ (\mathbf{S}\ (\mathbf{K}\ \mathbf{S})\ \langle f \to \mathbf{K}\ f\rangle)\ (\mathbf{K}\ \mathbf{I}) \\
&= \mathbf{S}\ (\mathbf{S}\ (\mathbf{K}\ \mathbf{S})\ (\mathbf{S}\ \langle f \to \mathbf{K}\rangle\ \langle f \to f\rangle))\ (\mathbf{K}\ \mathbf{I}) \\
&= \mathbf{S}\ (\mathbf{S}\ (\mathbf{K}\ \mathbf{S})\ (\mathbf{S}\ (\mathbf{K}\ \mathbf{K})\ \langle f \to f\rangle))\ (\mathbf{K}\ \mathbf{I}) \\
&= \mathbf{S}\ (\mathbf{S}\ (\mathbf{K}\ \mathbf{S})\ (\mathbf{S}\ (\mathbf{K}\ \mathbf{K})\ \mathbf{I}))\ (\mathbf{K}\ \mathbf{I})
\end{aligned}$$

$$\begin{aligned}
\mathbf{plus} &= \langle m \to \langle n \to \langle f \to \langle x \to m\ f\ (n\ f\ x)\rangle\rangle\rangle\rangle \\
&= \langle m \to \langle n \to \langle f \to \mathbf{S}\ \langle x \to m\ f\rangle\ \langle x \to n\ f\ x\rangle\rangle\rangle\rangle \\
&= \langle m \to \langle n \to \langle f \to \mathbf{S}\ (\mathbf{K}\ (m\ f))\ \langle x \to n\ f\ x\rangle\rangle\rangle\rangle \\
&= \langle m \to \langle n \to \langle f \to \mathbf{S}\ (\mathbf{K}\ (m\ f))\ (\mathbf{S}\ \langle x \to n\ f\rangle\ \langle x \to x\rangle)\rangle\rangle\rangle \\
&= \langle m \to \langle n \to \langle f \to \mathbf{S}\ (\mathbf{K}\ (m\ f))\ (\mathbf{S}\ (\mathbf{K}\ (n\ f))\ \langle x \to x\rangle)\rangle\rangle\rangle \\
&= \langle m \to \langle n \to \langle f \to \mathbf{S}\ (\mathbf{K}\ (m\ f))\ (\mathbf{S}\ (\mathbf{K}\ (n\ f))\ \mathbf{I})\rangle\rangle\rangle \\
&= \langle m \to \langle n \to \mathbf{S}\ \langle f \to \mathbf{S}\ (\mathbf{K}\ (m\ f))\rangle\ \langle f \to \mathbf{S}\ (\mathbf{K}\ (n\ f))\ \mathbf{I}\rangle\rangle\rangle \\
&= \langle m \to \langle n \to \mathbf{S}\ (\mathbf{S}\ \langle f \to \mathbf{S}\rangle\ \langle f \to \mathbf{K}\ (m\ f)\rangle)\ \langle f \to \mathbf{S}\ (\mathbf{K}\ (n\ f))\ \mathbf{I}\rangle\rangle\rangle \\
&= \langle m \to \langle n \to \mathbf{S}\ (\mathbf{S}\ (\mathbf{K}\ \mathbf{S})\ \langle f \to \mathbf{K}\ (m\ f)\rangle)\ \langle f \to \mathbf{S}\ (\mathbf{K}\ (n\ f))\ \mathbf{I}\rangle\rangle\rangle \\
&= \langle m \to \langle n \to \mathbf{S}\ (\mathbf{S}\ (\mathbf{K}\ \mathbf{S})\ (\mathbf{S}\ \langle f \to \mathbf{K}\rangle\ \langle f \to m\ f\rangle))\ \langle f \to \mathbf{S}\ (\mathbf{K}\ (n\ f))\ \mathbf{I}\rangle\rangle\rangle \\
&= \langle m \to \langle n \to \mathbf{S}\ (\mathbf{S}\ (\mathbf{K}\ \mathbf{S})\ (\mathbf{S}\ (\mathbf{K}\ \mathbf{K})\ \langle f \to m\ f\rangle))\ \langle f \to \mathbf{S}\ (\mathbf{K}\ (n\ f))\ \mathbf{I}\rangle\rangle\rangle \\
&= \langle m \to \langle n \to \mathbf{S}\ (\mathbf{S}\ (\mathbf{K}\ \mathbf{S})\ (\mathbf{S}\ (\mathbf{K}\ \mathbf{K})\ (\mathbf{S}\ \langle f \to m\rangle\ \langle f \to f\rangle)))\ \langle f \to \mathbf{S}\ (\mathbf{K}\ (n\ f))\ \mathbf{I}\rangle\rangle\rangle \\
&= \langle m \to \langle n \to \mathbf{S}\ (\mathbf{S}\ (\mathbf{K}\ \mathbf{S})\ (\mathbf{S}\ (\mathbf{K}\ \mathbf{K})\ (\mathbf{S}\ (\mathbf{K}\ m)\ \langle f \to f\rangle)))\ \langle f \to \mathbf{S}\ (\mathbf{K}\ (n\ f))\ \mathbf{I}\rangle\rangle\rangle \\
&= \langle m \to \langle n \to \mathbf{S}\ (\mathbf{S}\ (\mathbf{K}\ \mathbf{S})\ (\mathbf{S}\ (\mathbf{K}\ \mathbf{K})\ (\mathbf{S}\ (\mathbf{K}\ m)\ \mathbf{I})))\ \langle f \to \mathbf{S}\ (\mathbf{K}\ (n\ f))\ \mathbf{I}\rangle\rangle\rangle \\
&= \langle m \to \langle n \to \mathbf{S}\ (\mathbf{S}\ (\mathbf{K}\ \mathbf{S})\ (\mathbf{S}\ (\mathbf{K}\ \mathbf{K})\ (\mathbf{S}\ (\mathbf{K}\ m)\ \mathbf{I})))\ (\mathbf{S}\ \langle f \to \mathbf{S}\ (\mathbf{K}\ (n\ f))\rangle\ \langle f \to \mathbf{I}\rangle)\rangle\rangle \\
&= \langle m \to \langle n \to \mathbf{S}\ (\mathbf{S}\ (\mathbf{K}\ \mathbf{S})\ (\mathbf{S}\ (\mathbf{K}\ \mathbf{K})\ (\mathbf{S}\ (\mathbf{K}\ m)\ \mathbf{I})))\ (\mathbf{S}\ (\mathbf{S}\ \langle f \to \mathbf{S}\rangle\ \langle f \to \mathbf{K}\ (n\ f)\rangle)\ \langle f \to \mathbf{I}\rangle)\rangle\rangle \\
&= \ldots
\end{aligned}$$

You can complete it, if you have the patience. Better: write a program to perform the transformation. (It's an interesting exercise, though it probably helps to use a parse tree representation of terms rather than a string representation...) This is what I get for **plus**:

$$\begin{aligned}&\mathbf{S\ (S\ (K\ S)\ (S\ (K\ K)\ (S\ (K\ S)\ (S\ (K\ (S\ (K\ S)))\ (S\ (K\ (S\ (K\ K)))}\\ &\quad\mathbf{(S\ (S\ (K\ S)\ (S\ (K\ K)\ I))\ (K\ I)))))))\ (K\ (S\ (S\ (K\ S)\ (S\ (K\ (S\ (K\ S)))}\\ &\qquad\mathbf{(S\ (K\ (S\ (K\ K)))\ (S\ (S\ (K\ S)\ (S\ (K\ K)\ I))\ (K\ I)))))\ (K\ (K\ I))))}\end{aligned}$$

It is admittedly a mouthful.

The Θ combinator can be rewritten as:

$$\begin{aligned}&\mathbf{S\ (K\ (S\ I))\ (S\ (S\ (K\ S)\ (S\ (K\ K)\ (S\ I\ I)))\ (K\ I))\ (S\ (K\ (S\ I))}\\ &\quad\mathbf{(S\ (S\ (K\ S)\ (S\ (K\ K)\ (S\ I\ I)))\ (K\ I)))}\end{aligned}$$

The point is not that it's long or short, the point is that we can rewrite any combinator of the λ-calculus into applications of **S**, **K**, and **I**. In fact, it's easy to check that we can also replace **I** by **S K K** — though we can't simplify **S K K** into **I**, we can check that **S K K** acts exactly like the identify function, that is, (**S K K**) a simplifies to a, for all terms a. Therefore we can rewrite any λ-calculus combinator into applications of **S** and **K** and get a term that simplifies to the same normal forms.

This means that really, **S** and **K** is all we need to implement computations.

Combinatory Logic (CL) is the algebraic system that you obtain in this way. It is defined very simply. The terms of CL are as follows:

- **K** is a CL term.
- **S** is a CL term.
- If α and β are CL terms, then $(\alpha\ \beta)$ is a CL term.

You can simplify CL terms using two simplification rules:

$$(\mathbf{K}\ \alpha)\ \beta = \alpha$$
$$((\mathbf{S}\ \alpha)\ \beta)\ \gamma) = (\alpha\ \gamma)\ (\beta\ \gamma)$$

with the understanding that these simplifications may occur anywhere in a term.

CL is Turing-complete. Since we already know that any combinator of the λ-calculus can be rewritten into a CL term, and any Turing machine can be implemented using a combinator of the λ-calculus, we get that any Turing machine can be implemented using a term of CL.

This is one of the simplest Turing-complete systems.