Chapter 3

A short note on lambda-notation

3.1 λ-calculus: main ideas

The λ-calculus is built around two main ideas:

- *applying* a function to an argument
- forming (anonymous) functions by *abstraction* over names

This two basic concepts allow to build a theory of functions based on rules for computation, as opposed to the classical set-theoretic view of functions as sets of pairs (argument, result).

Let us start with a simple example from arithmetic:

- a polynomial such as $x^2 - 2x + 5$
- what is the value of the above expression when x is replaced by 2?
- we compute the result by plugging in '2′ for 'x' in the expression
- we get $2^2 + 2 \cdot 2 + 5$

In λ-notation, when we want to express that the value of an expression depends on some value to be plugged in, we use abstraction, written

$$\lambda x.\ (x^2 - 2x + 5)$$

whose informal reading is: wait for a value v to replace x and then compute $v^2 - 2v + 5$.

Note that:

- the symbol λ has no particular meaning (any other symbol could have been used)
- by writing $\lambda x.\ t$ we are declaring x has a formal parameter appearing in t
- we say that λx 'binds' the variable x in t (to avoid interference from the outside)
- we want to be able to pass some actual parameter to $\lambda x.\ t$, i.e., to apply the function $\lambda x.\ t$ to some value
- we denote application by juxtaposition, i.e., $(\lambda x.\ t)\ v$ means that the function $\lambda x.\ t$ is applied to v (or in other words that the actual parameter v replaces the occurrences of x in t)

Coming back to the previous example, the application $(\lambda x.\ (x^2 - 2x + 5))\ 2$ can be computed by substituting 2 for x in $x^2 - 2x + 5$, to obtain $2^2 + 2 \cdot 2 + 5 = 5$.

Let us consider another example:

$$(\lambda x.\ \lambda y.\ (x^2 - 2y + 5))\ 2$$

This time we have a function that is waiting for two arguments (first x, then y) to which we pass one value (2). We have

$$(\lambda x.\ \lambda y.\ (x^2 - 2y + 5))\ 2 = \lambda y.\ (2^2 - 2y + 5) = \lambda y.\ (9 - 2y)$$

that is, the result of applying $\lambda x.\ \lambda y.\ (x^2 - 2y + 5)$ to 2 is still a function $(\lambda y.\ (9 - 2y))$.

In λ-calculus we can pass function as arguments and return functions as results.

Note that while we can have different terms t and t' that define the same function, in general the problem of deciding whether $t = t'$ is undecidable.

To limit the number of parentheses and keep the notation more readable, we assume that application is left-associative, and lambda-abstraction is right-associative, i.e.,

$t_1\ t_2\ t_3\ t_4$	is read	$(((t_1\ t_2)\ t_3)\ t_4)$
$\lambda x_1.\ \lambda x_2.\ \lambda x_3.\ \lambda x_4.t$	is read	$\lambda x_1.\ (\lambda x_2.\ (\lambda x_3.\ (\lambda x_4.t)))$

3.2 λ-calculus: booleans and Church numerals

In the above examples, we have enriched standard arithmetic expressions with abstraction and application.

In general, it would be possible to encode booleans and numbers (and operations over them) just using abstraction and application.

For example, let us consider the following terms:

$$T \stackrel{\text{def}}{=} \lambda x.\, \lambda y.\, x$$
$$F \stackrel{\text{def}}{=} \lambda x.\, \lambda y.\, y$$

We can assume that T represents true and F represents false.

Under this convention, we can define the usual logical operations by letting:

$$\begin{aligned} \text{AND} &\stackrel{\text{def}}{=} \lambda p.\, \lambda q.\, p\ q\ p \\ \text{OR} &\stackrel{\text{def}}{=} \lambda p.\, \lambda q.\, p\ p\ q \\ \text{NOT} &\stackrel{\text{def}}{=} \lambda p.\, \lambda x.\, \lambda y.\, p\ y\ x \end{aligned}$$

In fact, suppose we want to compute AND $F\ T$, we have:

$$\begin{aligned} \text{AND}\ F\ T &= (\lambda p.\, \lambda q.\, p\ q\ p)\ F\ T \\ &= (\lambda q.\, F\ q\ F)\ T \\ &= F\ T\ F \\ &= (\lambda x.\, \lambda y.\, y)\ T\ F \\ &= (\lambda y.\, y)\ F \\ &= F \end{aligned}$$

As another example:

$$\begin{aligned} \text{AND}\ T\ T &= (\lambda p.\, \lambda q.\, p\ q\ p)\ T\ T \\ &= (\lambda q.\, T\ q\ T)\ T \\ &= T\ T\ T \\ &= (\lambda x.\, \lambda y.\, x)\ T\ T \\ &= (\lambda y.\, T)\ T \\ &= T \end{aligned}$$

The reader is invited to try other calculations, like OR $T\ F$ or NOT T.

Now suppose that P will reduce either to T or to F. The expression $P\ A\ B$ can be read as 'if P then A else B'.

For natural numbers, we can adopt the convention that the number n is represented by a function that takes a function f and an argument x and applies f to x for n times

consecutively. For example:

$$\begin{aligned} 0 &\stackrel{\text{def}}{=} \lambda f.\, \lambda x.\, x \\ 1 &\stackrel{\text{def}}{=} \lambda f.\, \lambda x.\, f\ x \\ 2 &\stackrel{\text{def}}{=} \lambda f.\, \lambda x.\, f\ (f\ x) \end{aligned}$$

Then, the operations for successor, sum, multiplication could be defined by letting:

$$\begin{aligned} \text{SUCC} &\stackrel{\text{def}}{=} \lambda n.\, \lambda f.\, \lambda x.\, f\ (n\ f\ x) \\ \text{SUM} &\stackrel{\text{def}}{=} \lambda n.\, \lambda m.\, \lambda f.\, \lambda x.\, m\ f\ (n\ f\ x) \\ \text{MUL} &\stackrel{\text{def}}{=} \lambda n.\, \lambda m.\, \lambda f.\, n\ (m\ f) \end{aligned}$$

In the following, when needed, we will assume that the data types of booleans and integers, together with the common operations on them, can appear in our λ-expressions.

3.3 Alpha-conversion, free variables and capture-avoiding substitution

The names of the formal parameters we choose for a given function should not matter. Therefore, any two expressions that differ just for the particular choice of λ-abstracted variables and have the same structure otherwise, should be considered as equal.

For example, we do not want to distinguish between the terms

$$\lambda x.\, (x^2 - 2x + 5) \qquad\qquad \lambda y.\, (y^2 - 2y + 5)$$

On the other hand, the expressions

$$x^2 - 2x + 5 \qquad\qquad y^2 - 2y + 5$$

must be distinguished, because depending on the context where they are used, the symbols x and y could have a different meaning.

We say that two terms are α-convertible if one is obtained from the other by renaming some λ-abstracted variables. We call *free* the variables x whose occurrences are not under the scope of a λ binder.

Formally, assume the following syntax for λ-terms:

$$t ::= x \mid \lambda x.t \mid tt \mid t \to t, t$$

where x is a variable, $\lambda x.t$ is the abstraction of x in t, $t_0\ t_1$ is the application of t_0 to t_1 and $t \to t_0, t_1$ is the conditional expression (if t is true then it behaves as t_0, otherwise as t_1).

We define α-conversion as the equivalence induced by letting

$$\lambda x.\ t = \lambda y.\ (t[y/x]) \quad \text{if } y \notin \mathrm{fv}(t)$$

Note the side condition $y \notin \mathrm{fv}(t)$, which is needed to avoid 'capturing' other free variables appearing in t. For example,

$$\lambda z.\ z^2 - 2y + 5 = \lambda x.\ x^2 - 2y + 5 \neq \lambda y.\ y^2 - 2y + 5$$

For example, the identity function can be written $\lambda x.\ x$ or (alpha-)equivalently $\lambda z.\ z$.

The set of free variables occurring in a term is defined by structural recursion:

$$\begin{aligned}
\mathrm{fv}(x) &= \{x\} \\
\mathrm{fv}(\lambda x.t) &= \mathrm{fv}(t) \setminus \{x\} \\
\mathrm{fv}(t_0\ t_1) &= \mathrm{fv}(t_0) \cup \mathrm{fv}(t_1) \\
\mathrm{fv}(t \to t_0, t_1) &= \mathrm{fv}(t) \cup \mathrm{fv}(t_0) \cup \mathrm{fv}(t_1)
\end{aligned}$$

Let us now try to define substitutions (but be aware that the following is a wrong attempt):

$$\begin{aligned}
y[t/x] &= \begin{cases} t & \text{if } y = x \\ y & \text{if } y \neq x \end{cases} \\
(\lambda y.t')[t/x] &= \begin{cases} \lambda y.t' & \text{if } y = x \\ \lambda y.(t'[t/x]) & \text{if } y \neq x \end{cases} \\
(t_0\ t_1)[t/x] &= (t_0[t/x])\ (t_1[t/x]) \\
(t' \to t_0, t_1)[t/x] &= (t'[t/x]) \to (t_0[t/x]), (t_1[t/x])
\end{aligned}$$

What is wrong with the above attempt?

Consider the term $t = \lambda x.\ \lambda y.\ (x^2 - 2y + 5)$ and $t' = y$. Then, take $t\ t'$:

$$\begin{aligned}
t\ t' &= (\lambda x.\ \lambda y.\ (x^2 - 2y + 5))\ y \\
&= (\lambda y.\ (x^2 - 2y + 5))[y/x] \\
&= \lambda y.\ ((x^2 - 2y + 5)[y/x]) \\
&= \lambda y.\ (y^2 - 2y + 5)
\end{aligned}$$

It happens that the free variable $y \in \mathrm{fv}(t\ t')$ has been 'captured' by the lambda-abstraction λy. Instead, free variables occurring in t should remain free during the application of the substitution $[t/x]$.

Thus we need to correct the above version of substitution for the case related to $(\lambda y.t')[t/x]$ by applying first the alpha-conversion to $\lambda y.t'$ (to make sure that if $y \in \mathrm{fv}(t)$, then the free occurrences of y in t will not be captured by λy when replacing x in t') and then the substitution $[t/x]$. Formally, we let:

$$\begin{array}{rcl} y[t/x] & = & \begin{cases} t & \text{if } y = x \\ y & \text{if } y \neq x \end{cases} \\ (\lambda y.t')[t/x] & = & \lambda z.((t'[z/y])[t/x]) \quad \text{if } z \notin \mathrm{fv}(\lambda y.\ t') \cup \mathrm{fv}(t) \cup \{x\} \\ (t_0\ t_1)[t/x] & = & (t_0[t/x])\ (t_1[t/x]) \\ (t' \to t_0, t_1)[t/x] & = & (t'[t/x]) \to (t_0[t/x]), (t_1[t/x]) \end{array}$$

3.4 Beta-rule

We have now all ingredients to define the basic computational rule, called β-rule, which explain how to apply a function to an argument:

$$(\lambda y.\ t')\ t = t'[t/x]$$

3.5 Exercises

1. Is $\lambda x.\ \lambda x.\ x$ α-convertible to one or more of the following expressions?
   (a) $\lambda y.\ \lambda x.\ x$
   (b) $\lambda y.\ \lambda x.\ y$
   (c) $\lambda y.\ \lambda y.\ y$
   (d) $\lambda x.\ \lambda y.\ x$
   (e) $\lambda z.\ \lambda w.\ w$
2. Is $(\lambda x.\ \lambda y.\ x)\ y$ equivalent to one or more of the following expressions?
   (a) $\lambda y.\ \lambda y.\ y$
   (b) $\lambda y.\ y$

(c) $\lambda y.\ z$

(d) $\lambda z.\ y$

(e) $\lambda x.\ y$