# Untyped $\lambda$-calculus exercises

Adrián Enríquez Ballester

January 9, 2022

## Foundations

### Exercise 1

a) Classify the following terms according to $\alpha$-equivalence:

| | | |
|---|---|---|
| $\lambda x.x\ y$ | $\lambda z.z\ z$ | $\lambda f.f\ f$ |
| $\lambda x.x\ z$ | $\lambda z.z\ y$ | $\lambda y.\lambda x.x\ y$ |
| $\lambda y.y\ z$ | $\lambda f.f\ y$ | $\lambda z.\lambda y.y\ z$ |

**Solution:** we can perform an $\alpha$-conversion to the following terms in order to reach another one of the list:

$$\begin{aligned}
\lambda x.x\ y &\rightsquigarrow_\alpha \lambda z.(x\ y)[z/x] \equiv \lambda z.z\ y \\
\lambda x.x\ y &\rightsquigarrow_\alpha \lambda f.(x\ y)[f/x] \equiv \lambda f.f\ y \\
\lambda x.x\ z &\rightsquigarrow_\alpha \lambda y.(x\ z)[y/x] \equiv \lambda y.y\ z \\
\lambda z.z\ z &\rightsquigarrow_\alpha \lambda f.(z\ z)[f/z] \equiv \lambda f.f\ f
\end{aligned}$$

The last one requires of two $\alpha$-conversions:

$$\begin{aligned}
\lambda y.\lambda x.x\ y &\rightsquigarrow_\alpha \lambda z.(\lambda x.x\ y)[z/y] \equiv \lambda z.\lambda x.x\ z \\
&\rightsquigarrow_\alpha \lambda z.\lambda y.(x\ z)[y/x] \equiv \lambda z.\lambda y.y\ z
\end{aligned}$$

To sum up, we have found the following $\alpha$-equivalences, grouping the terms into four equivalence classes:

$$\begin{gathered}
\lambda x.x\ y =_\alpha \lambda z.z\ y =_\alpha \lambda f.f\ y \\
\lambda x.x\ z =_\alpha \lambda y.y\ z \\
\lambda z.z\ z =_\alpha \lambda f.f\ f \\
\lambda y.\lambda x.x\ y =_\alpha \lambda z.\lambda y.y\ z
\end{gathered}$$

b) Provide an $\alpha$-equivalent term where each abstraction uses a different variable name:

$$\lambda x.((x\ (\lambda y.x\ y))\ (\lambda x.x))\ (\lambda y.y\ x)$$

**Solution:** it suffices to perform an $\alpha$-conversion to two of its abstraction subterms using a new variable name:

$$\begin{aligned}
&\lambda x.((x\ (\lambda y.x\ y))\ (\lambda x.x))\ (\lambda y.y\ x)\\
&\quad \rightsquigarrow_\alpha \lambda x.((x\ (\lambda y.x\ y))\ (\lambda z.x[z/x]))\ (\lambda y.y\ x)\\
&\quad \equiv \lambda x.((x\ (\lambda y.x\ y))\ (\lambda z.z))\ (\lambda y.y\ x)\\
&\quad \rightsquigarrow_\alpha \lambda x.((x\ (\lambda y.x\ y))\ (\lambda z.z))\ (\lambda w.(y\ x)[w/y])\\
&\quad \equiv \lambda x.((x\ (\lambda y.x\ y))\ (\lambda z.z))\ (\lambda w.w\ x)
\end{aligned}$$

The resulting term has a different variable name for each one of its abstraction subterms variable names.

## Exercise 2

Normalize the following term:

$$(\lambda x.(\lambda y.x\ y))\ y$$

**Solution:** although it seems to be a simple $\beta$-reduction, we must perform capture-avoiding substitution carefully in order to get the right normalized term:

$$\begin{aligned}
&(\lambda x.(\lambda y.x\ y))\ y\\
&\quad \rightsquigarrow_\beta (\lambda y.x\ y)[y/x]\\
&\quad \equiv \lambda z.(x\ y)[z/y][y/x]\\
&\quad \equiv \lambda z.(x\ z)[y/x]\\
&\quad \equiv \lambda z.y\ z
\end{aligned}$$

## Exercise 3

The de Bruijn index notation is a way of avoiding the problems related to substitution and variable capture in Church's original presentation of the $\lambda$-calculus, thus facilitating its mechanized treatment. The key idea is to replace variable names by numbers denoting the depth of the scope of that variable. For example, the familiar terms $\lambda x.x$, $\lambda x.\lambda y.x$ and $\lambda x.\lambda y.y$ are represented as $\lambda 1$, $\lambda\lambda 2$ and $\lambda\lambda 1$ in de Bruijn's notation. Free variables are represented by numbers higher than the maximum depth in its location. For example, $\lambda\lambda 3$ is a possible representation for $\lambda x.\lambda y.w$.

a) Represent the term $(\lambda x.\lambda y.\lambda z.x\ z\ y)(\lambda x.\lambda y.x)$ in de Brujin 's notation.

**Solution:**

$$(\lambda\lambda\lambda 3\ 1\ 2)\ (\lambda\lambda 2)$$

b) Explain how $\beta$-reduction of terms in de Brujin notation can be implemented.

**Solution:** we are going to explain it informally and show an example in c). Given an application of an abstraction $N$ to a term $M$ (i.e. $N\ M$):

- The variables in the body of $N$ which are bounded by its formal parameter must be replaced by $M$.
- In each particular replacement, the free variables in $M$ must be incremented by the scope depth at the variable subterm where each replacement is taking place.
- The free variables which were free in the body of $N$ before the reduction must be decremented by one because the depth of its scope is being decremented.

c) Apply your ideas to the application in a).

**Solution:**

$$\begin{aligned}(\lambda\lambda\lambda 3\ 1\ 2)\ (\lambda\lambda 2)& \\ &\rightsquigarrow_\beta \lambda\lambda(\lambda\lambda 2)\ 1\ 2 && (1)\\ &\rightsquigarrow_\beta \lambda\lambda(\lambda 2)\ 2 && (2)\\ &\rightsquigarrow_\beta \lambda\lambda 1 && (3)\end{aligned}$$

In the $\beta$-reduction at (1), the bound variable was 3 and there were not free variables in the abstraction body, so it was just replaced by $\lambda\lambda 2$, which also has not free variables.

In the $\beta$-reduction at (2), the bound variable was the innermost 2, which has been replaced by 1 incremented by 1. This is due to 1 being free in itself and replaced inside an abstraction of depth 1.

Finally, in the $\beta$-reduction at (3), there were no occurrences of the bound variable of the abstraction being reduced, so the body remains the same but decrementing its single free variable.

## Exercise 4

Combinators can be seen as $\lambda$-terms without free variables, although they were actually proposed independently from the $\lambda$-calculus. Given the combinators

$$\mathsf{S} \triangleq \lambda x.\lambda y.\lambda z.(x\ z)\ (y\ z)$$
$$\mathsf{K} \triangleq \lambda x.\lambda y.x$$
$$\mathsf{I} \triangleq \lambda x.x$$

prove the equivalence $\mathsf{SKK} = \mathsf{I}$.

**Solution:** an advantage of applying an abstraction to a combinator is that, as a combinator has not free variables, there is no chance for variable capture to happen.

$$\begin{aligned}
&\mathsf{SKK}\\
&\quad \equiv (\lambda x.\lambda y.\lambda z.(x\ z)\ (y\ z))\ (\lambda x.\lambda y.x)\ (\lambda x.\lambda y.x)\\
&\rightsquigarrow_\beta (\lambda y.\lambda z.((x\ z)\ (y\ z)))[(\lambda x.\lambda y.x)/x]\ (\lambda x.\lambda y.x)\\
&\quad \equiv (\lambda y.\lambda z.((\lambda x.\lambda y.x)\ z)\ (y\ z))\ (\lambda x.\lambda y.x)\\
&\rightsquigarrow_\beta (\lambda z.((\lambda x.\lambda y.x)\ z)\ (y\ z))[(\lambda x.\lambda y.x)/y]\\
&\quad \equiv \lambda z.((\lambda x.\lambda y.x)\ z)\ ((\lambda x.\lambda y.x)\ z)\\
&\rightsquigarrow_\beta \lambda z.(\lambda y.x)[z/x]\ ((\lambda x.\lambda y.x)\ z)\\
&\quad \equiv \lambda z.(\lambda y.z)\ ((\lambda x.\lambda y.x)\ z)\\
&\rightsquigarrow_\beta \lambda z.z[((\lambda x.\lambda y.x)\ z)/y]\\
&\quad \equiv \lambda z.z\\
&\quad =_\alpha \mathsf{I}
\end{aligned}$$

## Exercise 5

Combinator systems are commonly presented as equational theories where a combinator base is defined using oriented equational rules and new combinators are created by means of application alone (i.e. no abstraction is required). For example, the aforementioned combinators would be defined by the equations $\mathsf{S}\ m\ n\ o = m\ o\ (n\ o)$, $\mathsf{K}\ a\ b = a$ and $\mathsf{I}\ x = x$. As variables only appear in definitions where no confusion of scopes can happen, combinators solve in a natural way many of the nuisances associated with variable names in Church's formulation of the $\lambda$-calculus.

Take as base the following two combinators:

$$\mathsf{B}\ f\ g\ x = f\ (g\ x)$$
$$\mathsf{M}\ x = x\ x$$

Using $\mathsf{B}$ and $\mathsf{M}$ alone prove the existence of a narcissistic combinator $n$ such that $nn = n$.

**Solution:** I have not found a better way for finding such a term than 'brute force'. As it was none of the possible combinators of size one, two and three, I

wrote a program that generates all the possible combinators of increasing size and tests for this property to hold. The smallest result was

$$n \triangleq \mathsf{M}\ (\mathsf{B}\ \mathsf{M}\ \mathsf{M})$$

which can be verified to hold the property:

$$\begin{aligned} n &= \mathsf{M}\ (\mathsf{B}\ \mathsf{M}\ \mathsf{M}) \\ &= \mathsf{B}\ \mathsf{M}\ \mathsf{M}\ (\mathsf{B}\ \mathsf{M}\ \mathsf{M}) \\ &= \mathsf{M}\ (\mathsf{M}\ (\mathsf{B}\ \mathsf{M}\ \mathsf{M})) \\ &= \mathsf{M}\ (\mathsf{B}\ \mathsf{M}\ \mathsf{M})\ (\mathsf{M}\ (\mathsf{B}\ \mathsf{M}\ \mathsf{M})) \\ &= n\ n \end{aligned}$$

Just for fun, these are some other bigger solutions that the program has found:

$$\mathsf{M}\ (\mathsf{B}\ (\mathsf{B}\ \mathsf{M})\ \mathsf{M})\ \mathsf{B}$$

$$\mathsf{M}\ (\mathsf{B}\ (\mathsf{B}\ \mathsf{M})\ \mathsf{M})\ \mathsf{M}$$

$$\mathsf{M}\ (\mathsf{B}\ (\mathsf{B}\ \mathsf{M})\ \mathsf{M})\ (\mathsf{B}\ \mathsf{B})$$

$$\mathsf{M}\ (\mathsf{B}\ (\mathsf{B}\ \mathsf{M})\ \mathsf{M})\ (\mathsf{B}\ \mathsf{M})$$

$$\mathsf{M}\ (\mathsf{B}\ (\mathsf{B}\ \mathsf{M})\ \mathsf{M})\ (\mathsf{M}\ \mathsf{B})$$

$$\mathsf{M}\ (\mathsf{B}\ (\mathsf{B}\ \mathsf{M})\ \mathsf{M})\ (\mathsf{M}\ \mathsf{M})$$

$$\mathsf{M}\ (\mathsf{B}\ \mathsf{B}\ \mathsf{B}\ \mathsf{M}\ \mathsf{M})\ \mathsf{B}$$

$$\mathsf{M}\ (\mathsf{B}\ \mathsf{B}\ \mathsf{B}\ \mathsf{M}\ \mathsf{M})\ \mathsf{M}$$

$$\mathsf{M}\ (\mathsf{M}\ \mathsf{B}\ \mathsf{B}\ \mathsf{M}\ \mathsf{M})\ \mathsf{B}$$

$$\mathsf{M}\ (\mathsf{M}\ \mathsf{B}\ \mathsf{B}\ \mathsf{M}\ \mathsf{M})\ \mathsf{M}$$

By observing these results, it is possible that

$$\mathsf{M}\ (\mathsf{B}\ (\mathsf{B}\ \mathsf{M})\ \mathsf{M})\ x$$

may be narcissistic for any $\mathsf{BM}$-combinator $x$, which is indeed the case:

$$\begin{aligned} \mathsf{M}\ (\mathsf{B}\ (\mathsf{B}\ \mathsf{M})\ \mathsf{M})\ x &= \mathsf{B}\ (\mathsf{B}\ \mathsf{M})\ \mathsf{M}\ (\mathsf{B}\ (\mathsf{B}\ \mathsf{M})\ \mathsf{M})\ x \\ &= \mathsf{B}\ \mathsf{M}\ (\mathsf{M}\ (\mathsf{B}\ (\mathsf{B}\ \mathsf{M})\ \mathsf{M}))\ x \\ &= \mathsf{M}\ (\mathsf{M}\ (\mathsf{B}\ (\mathsf{B}\ \mathsf{M})\ \mathsf{M})\ x) \\ &= \mathsf{M}\ (\mathsf{B}\ (\mathsf{B}\ \mathsf{M})\ \mathsf{M})\ x\ (\mathsf{M}\ (\mathsf{B}\ (\mathsf{B}\ \mathsf{M})\ \mathsf{M})\ x) \end{aligned}$$

# Constructive mathematics in the $\lambda$-calculus

All the encodings and terms which represent operations appearing in this section have been automatically tested in a more extensive way than some of the examples shown here. The corresponding code is due to be presented in an upcoming exercises delivery.

Also, the reductions in this section are shown in a less explicit way than in the previous one, avoiding the intermediate steps of safely replacing variables and even performing more than one reduction at a time between steps (e.g. two $\beta$-reductions are indicated as $\rightsquigarrow_\beta^2$). For long reductions we have used the syntax $\boxed{abstraction}$ $\boxed{parameter}$ to show the application term(s) being $\beta$-reduced in order to improve its readability.

## Exercise 6

Using Church's encoding of booleans in the pure $\lambda$-calculus, define normalized $\lambda$-terms CONJ, DISJ and NEG to represent conjunction, disjunction and negation, respectively.

**Solution:** recall the definition of the boolean terms as

$$\mathsf{TRUE} \triangleq \lambda x.\lambda y.x$$
$$\mathsf{FALSE} \triangleq \lambda x.\lambda y.y$$

First, NEG must be a term which when applied to a boolean term becomes the other one. This can be achieved by embedding it into an abstraction which applies it to its parameters in reverse order:

$$\mathsf{NEG} \triangleq \lambda b.\lambda x.\lambda y.b\ y\ x$$

Its behavior is the following:

$$\begin{aligned}
&\mathsf{NEG\ TRUE}\\
&\quad \equiv (\lambda b.\lambda x.\lambda y.b\ y\ x)\ (\lambda x.\lambda y.x)\\
&\rightsquigarrow_\beta \lambda x.\lambda y.(\lambda x.\lambda y.x)\ y\ x\\
&\rightsquigarrow_\beta^2 \lambda x.\lambda y.y\\
&\quad \equiv \mathsf{FALSE}
\end{aligned}$$

NEG FALSE

$$\begin{aligned}
&\equiv(\lambda b.\lambda x.\lambda y.b\ y\ x)\ (\lambda x.\lambda y.y)\\
&\rightsquigarrow_\beta \lambda x.\lambda y.(\lambda x.\lambda y.y)\ y\ x\\
&\rightsquigarrow^2_\beta \lambda x.\lambda y.x\\
&\equiv \mathsf{TRUE}
\end{aligned}$$

Second, CONJ must be a term which when applied to two boolean terms becomes into FALSE if the first one also is, and into the second one otherwise. It can be defined as

$$\mathsf{CONJ} \triangleq \lambda b.\lambda c.\lambda x.\lambda y.b\ (c\ x\ y)\ y$$

because

CONJ FALSE

$$\begin{aligned}
&\equiv(\lambda b.\lambda c.\lambda x.\lambda y.b\ (c\ x\ y)\ y)\ (\lambda x.\lambda y.y)\\
&\rightsquigarrow_\beta \lambda c.\lambda x.\lambda y.(\lambda x.\lambda y.y)\ (c\ x\ y)\ y\\
&\rightsquigarrow^2_\beta \lambda c.\lambda x.\lambda y.y\\
&\equiv \lambda c.\mathsf{FALSE}
\end{aligned}$$

and

CONJ TRUE

$$\begin{aligned}
&\equiv(\lambda b.\lambda c.\lambda x.\lambda y.b\ (c\ x\ y)\ y)\ (\lambda x.\lambda y.x)\\
&\rightsquigarrow_\beta \lambda c.\lambda x.\lambda y.(\lambda x.\lambda y.x)\ (c\ x\ y)\ y\\
&\rightsquigarrow^2_\beta \lambda c.\lambda x.\lambda y.c\ x\ y\\
&\rightsquigarrow^2_\eta \lambda c.c
\end{aligned}$$

where we have used also $\eta$-reduction for the ease of reasoning about the property.

Finally, DISJ must be a term which when applied to two boolean terms becomes into TRUE if the first one also is, and into the second one otherwise. It can be defined as

$$\mathsf{DISJ} \triangleq \lambda b.\lambda c.\lambda x.\lambda y.b\ x\ (c\ x\ y)$$

because

$$
\begin{aligned}
&\mathsf{DISJ\ TRUE}\\
&\quad \equiv (\lambda b.\lambda c.\lambda x.\lambda y.b\ x\ (c\ x\ y))\ (\lambda x.\lambda y.x)\\
&\rightsquigarrow_{\beta} \lambda c.\lambda x.\lambda y.(\lambda x.\lambda y.x)\ x\ (c\ x\ y)\\
&\rightsquigarrow^{2}_{\beta} \lambda c.\lambda x.\lambda y.x\\
&\quad \equiv \lambda c.\mathsf{TRUE}
\end{aligned}
$$

and

$$
\begin{aligned}
&\mathsf{DISJ\ FALSE}\\
&\quad \equiv (\lambda b.\lambda c.\lambda x.\lambda y.b\ x\ (c\ x\ y))\ (\lambda x.\lambda y.y)\\
&\rightsquigarrow_{\beta} \lambda c.\lambda x.\lambda y.(\lambda x.\lambda y.y)\ x\ (c\ x\ y)\\
&\rightsquigarrow^{2}_{\beta} \lambda c.\lambda x.\lambda y.c\ x\ y\\
&\rightsquigarrow^{2}_{\eta} \lambda c.c
\end{aligned}
$$

where, again, we have used also $\eta$-reduction for the ease of reasoning about the property.

## Exercise 7

Using Church's encoding of natural numbers define addition, multiplication and exponentiation.

**Solution:** recall that natural numbers are encoded as

$$\lambda s.\lambda z.s\ (s\ (...\ s(z)\ ...))$$

where the number of times that $s$ is applied determines the natural number it represents. For example, we have the following encodings:

$$
\begin{aligned}
&\mathsf{0} \triangleq \lambda s.\lambda z.z\\
&\mathsf{1} \triangleq \lambda s.\lambda z.s\ z\\
&\mathsf{2} \triangleq \lambda s.\lambda z.s\ (s\ z)
\end{aligned}
$$

First, $\mathsf{ADD}$ must be a natural number term that given two natural number terms $m$ and $n$, which respectively represent two natural numbers $a$ and $b$, must represent the natural number $a + b$. It can be defined as

$$\mathsf{ADD} \triangleq \lambda m.\lambda n.\lambda s.\lambda z.n\ s\ (m\ s\ z)$$

because, under the initial explanation conditions, it applies $b$ times $s$ to the result of applying $a$ times $s$ starting with $z$ and therefore $s$ is applied $a+b$ times.

This is an example of its behavior:

ADD 3 2

$$
\begin{aligned}
&\equiv \boxed{(\lambda m.\lambda n.\lambda s.\lambda z.n\ s\ (m\ s\ z))}\ \boxed{(\lambda s.\lambda z.s\ (s\ (s\ z)))}\ \boxed{(\lambda s.\lambda z.s\ (s\ z))}\\
&\rightsquigarrow_\beta^2 \lambda s.\lambda z.\boxed{(\lambda s.\lambda z.s\ (s\ z))}\ \boxed{s}\ (\boxed{(\lambda s.\lambda z.s\ (s\ (s\ z)))}\ \boxed{s}\ \boxed{z})\\
&\rightsquigarrow_\beta^3 \lambda s.\lambda z.\boxed{(\lambda z.s\ (s\ z))}\ \boxed{(s\ (s\ (s\ z)))}\\
&\rightsquigarrow_\beta \lambda s.\lambda z.s\ (s\ (s\ (s\ (s\ z))))\\
&\equiv 5
\end{aligned}
$$

Second, MULT must be a natural number term that given two natural number terms $m$ and $n$, which respectively represent two natural numbers $a$ and $b$, must represent the natural number $a \cdot b$. It can be defined as

$$\mathsf{MULT} \triangleq \lambda m.\lambda n.\lambda s.\lambda z.n\ (m\ s)\ z$$

because, under the initial explanation conditions, it applies $b$ times a term which applies $a$ times $s$ starting with $z$ and therefore $s$ is applied $a$ times $b$ times.

An example of its behavior is as follows:

MULT 3 2

$$
\begin{aligned}
&\equiv \boxed{(\lambda m.\lambda n.\lambda s.\lambda z.n\ (m\ s)\ z)}\ \boxed{(\lambda s.\lambda z.s\ (s\ (s\ z)))}\ \boxed{(\lambda s.\lambda z.s\ (s\ z))}\\
&\rightsquigarrow_\beta^2 \lambda s.\lambda z.(\lambda s.\lambda z.s\ (s\ z))\ (\boxed{(\lambda s.\lambda z.s\ (s\ (s\ z)))}\ \boxed{s})\ z\\
&\rightsquigarrow_\beta \lambda s.\lambda z.\boxed{(\lambda s.\lambda z.s\ (s\ z))}\ \boxed{(\lambda z.s\ (s\ (s\ z)))}\ z\\
&\rightsquigarrow_\beta \lambda s.\lambda z.\boxed{(\lambda z.(\lambda z.\ s\ (s\ (s\ z)))\ ((\lambda z.s\ (s\ (s\ z)))\ z))}\ \boxed{z}\\
&\rightsquigarrow_\beta \lambda s.\lambda z.(\lambda z.\ s\ (s\ (s\ z)))\ (\boxed{(\lambda z.s\ (s\ (s\ z)))}\ \boxed{z})\\
&\rightsquigarrow_\beta \lambda s.\lambda z.\boxed{(\lambda z.\ s\ (s\ (s\ z)))}\ \boxed{(s\ (s\ (s\ z)))}\\
&\rightsquigarrow_\beta \lambda s.\lambda z.s\ (s\ (s\ (s\ (s\ (s\ z)))))\\
&\equiv 6
\end{aligned}
$$

Finally, EXPO must be a natural number term that given two natural number terms $m$ and $n$, which respectively represent two natural numbers $a$ and $b$, must represent the natural number $a^b$. It can be defined as

$$\mathsf{EXPO} \triangleq \lambda m.\lambda n.\lambda s.\lambda z.n\ m\ s\ z$$

because, under the initial explanation conditions, it applies $b$ times $m$ starting with $s$ and, as $m$ applies $a$ times its parameter, the result is $s^{a^b}$ (i.e. $s$ applied $a^b$ times).

We also show an example of its behavior:

$\mathsf{EXPO}\ 3\ 2$

$$
\begin{aligned}
&\equiv \boxed{(\lambda m.\lambda n.\lambda s.\lambda z.n\ m\ s\ z)}\ \boxed{(\lambda s.\lambda z.s\ (s\ z)))}\ \boxed{(\lambda s.\lambda z.s\ (s\ z))} \\
&\rightsquigarrow_\beta^2 \lambda s.\lambda z.\ \boxed{(\lambda s.\lambda z.s\ (s\ z))}\ \boxed{(\lambda s.\lambda z.s\ (s\ (s\ z)))}\ s\ z \\
&\rightsquigarrow_\beta \lambda s.\lambda z.\ \boxed{(\lambda z.(\lambda s.\lambda z.s\ (s\ (s\ z)))\ ((\lambda s.\lambda z.s\ (s\ (s\ z)))\ z))}\ \boxed{s}\ z \\
&\rightsquigarrow_\beta \lambda s.\lambda z.((\lambda s.\lambda z.s\ (s\ (s\ z)))\ (\ \boxed{(\lambda s.\lambda z.s\ (s\ (s\ z)))}\ \boxed{s}\ ))\ z \\
&\rightsquigarrow_\beta \lambda s.\lambda z.(\ \boxed{(\lambda s.\lambda z.s\ (s\ (s\ z)))}\ \boxed{(\lambda z.s\ (s\ (s\ z)))}\ )\ z \\
&\rightsquigarrow_\beta \lambda s.\lambda z.\ \boxed{(\lambda z.(\lambda z.s\ (s\ (s\ z)))\ ((\lambda z.s\ (s\ (s\ z)))\ ((\lambda z.s\ (s\ (s\ z)))\ z)))}\ \boxed{z} \\
&\rightsquigarrow_\beta \lambda s.\lambda z.(\lambda z.s\ (s\ (s\ z)))\ ((\lambda z.s\ (s\ (s\ z)))\ (\ \boxed{(\lambda z.s\ (s\ (s\ z)))}\ \boxed{z}\ )) \\
&\rightsquigarrow_\beta \lambda s.\lambda z.(\lambda z.s\ (s\ (s\ z)))\ (\ \boxed{(\lambda z.s\ (s\ (s\ z)))}\ \boxed{(s\ (s\ (s\ z)))}\ ) \\
&\rightsquigarrow_\beta \lambda s.\lambda z.\ \boxed{(\lambda z.s\ (s\ (s\ z)))}\ \boxed{(s\ (s\ (s\ (s\ (s\ (s\ z))))))} \\
&\rightsquigarrow_\beta \lambda s.\lambda z.s\ (s\ (s\ (s\ (s\ (s\ (s\ (s\ z))))))) \\
&\equiv \mathsf{9}
\end{aligned}
$$

Although $\mathsf{MULT}$ and $\mathsf{EXPO}$ as defined here are $\beta$-normal terms, they are not $\eta$-normal terms and can be expressed in a more concise way:

$$
\begin{aligned}
\mathsf{MULT} &\rightsquigarrow_\eta \lambda m.\lambda n.\lambda s.m\ (n\ s) \\
\mathsf{EXPO} &\rightsquigarrow_\eta^2 \lambda m.\lambda n.n\ m
\end{aligned}
$$

## Exercise 8

Devise an encoding for pairs in the $\lambda$-calculus. Provide $\lambda$-terms $\mathsf{PAIR}$ (a pair constructor) and the projections $\mathsf{FST}$ and $\mathsf{SND}$. What properties would be required in order to check the correctness of the encoding?

**Solution:** a pair can be represented as

$$\lambda f.f\ a\ b$$

which applies a given term $f$ to the pair contents $a$ and $b$.
Such a term can be constructed waiting for its contents to be provided as

$$\mathsf{PAIR} \triangleq \lambda a.\lambda b.\lambda f.f\ a\ b$$

and its respective projections, which allow to retrieve them as

$$\mathsf{FST} \triangleq \lambda p.p(\lambda x.\lambda y.x)$$

$$\mathsf{SND} \triangleq \lambda p.p(\lambda x.\lambda y.y)$$

This encoding must satisfy the properties

$$\mathsf{FST}\ (\mathsf{PAIR}\ a\ b) \rightsquigarrow_\beta^* a$$
$$\mathsf{SND}\ (\mathsf{PAIR}\ a\ b) \rightsquigarrow_\beta^* b$$

which state that the projections over a pair must effectively allow to retrieve the corresponding terms used when constructing it.

We show that this is the case for the FST term, and it can also be checked in a similar way for SND:

$$\begin{aligned}
&\mathsf{FST}\ (\mathsf{PAIR}\ a\ b)\\
&\qquad \equiv (\lambda p.p\ (\lambda x.\lambda y.x))\ ((\lambda a.\lambda b.\lambda f.f\ a\ b)\ a\ b)\\
&\rightsquigarrow_\beta^2 (\lambda p.p\ (\lambda x.\lambda y.x))\ (\lambda f.f\ a\ b)\\
&\rightsquigarrow_\beta (\lambda f.f\ a\ b)\ (\lambda x.\lambda y.x)\\
&\rightsquigarrow_\beta (\lambda x.\lambda y.x)\ a\ b\\
&\rightsquigarrow_\beta^2 a
\end{aligned}$$

## Exercise 9

Define a predecessor function and subtraction for Church numerals.

**Solution:** we can define a term that given a natural number term $n$ wraps it and becomes into a natural number term that makes $n$ to avoid one of its $s$ applications. This means that, at least for our definition, the predecessor of 0 is going to be 0:

$$\begin{aligned}
&\lambda n.\lambda s.\lambda z.\mathsf{FST}\ (\\
&\qquad n\\
&\qquad (\lambda p.\mathsf{PAIR}\ (\mathsf{SND}\ p\ (\mathsf{FST}\ p))\ s) && (2)\\
&\qquad (\mathsf{PAIR}\ z\ (\lambda x.x)) && (1)\\
&\quad )
\end{aligned} \tag{3}$$

At (1), we wrap the initial element for $n$ within a pair, where the first element is $z$ and the second element is the next function to apply to it. The identity abstraction is given at first.

At (2), the function which $n$ applies expects to take a pair and yield a new pair, where the second element is $s$ and the first element is its second applied to its first. As initially we have provided the identity abstraction, the original $s$ is going to be applied the times $n$ does except for the first one.

To conclude with the explanation, the application of $n$ to a function between pairs and a pair would also yield a pair, so we apply FST at (3) in order to end up with the desired result, which has been laying into the first element of the pair.

In order to achieve a concise and normalized definition, we expand the outermost FST definition and the PAIR ones, but instead of expanding SND $p$ (FST $p$), we can write directly a function which given a pair applies its second element to its first:

$$\begin{aligned}\mathsf{PRED} \triangleq & \lambda n.\lambda s.\lambda z.(\\ & \quad n\\ & \quad (\lambda p.\lambda f.f\ (p\ (\lambda x.\lambda y.y\ x))\ s)\\ & \quad (\lambda f.f\ z\ (\lambda x.x))\\ & )\ (\lambda x.\lambda y.x)\end{aligned}$$

This is an example of its behavior:

$\mathsf{PRED}\ 2$

$$\begin{aligned}
&\equiv \boxed{(\lambda n.\lambda s.\lambda z.(n\ (\lambda p.\lambda f.f\ (p\ (\lambda x.\lambda y.y\ x))\ s)\ (\lambda f.f\ z\ (\lambda x.x)))\ (\lambda x.\lambda y.x))}\ \boxed{(\lambda s.\lambda z.s\ (s\ z))}\\
&\rightsquigarrow_\beta \lambda s.\lambda z.(\boxed{(\lambda s.\lambda z.s\ (s\ z))}\ \boxed{(\lambda p.\lambda f.f\ (p\ (\lambda x.\lambda y.y\ x))\ s)}\ \boxed{(\lambda f.f\ z\ (\lambda x.x))})\ (\lambda x.\lambda y.x)\\
&\rightsquigarrow_\beta^2 \lambda s.\lambda z.((\lambda p.\lambda f.f\ (p\ (\lambda x.\lambda y.y\ x))\ s)\ (\boxed{(\lambda p.\lambda f.f\ (p\ (\lambda x.\lambda y.y\ x))\ s)}\ \boxed{(\lambda f.f\ z\ (\lambda x.x))}))\ (\lambda x.\lambda y.x)\\
&\rightsquigarrow_\beta \lambda s.\lambda z.(\boxed{(\lambda p.\lambda f.f\ (p\ (\lambda x.\lambda y.y\ x))\ s)}\ \boxed{(\lambda f.f\ ((\lambda f.f\ z\ (\lambda x.x))\ (\lambda x.\lambda y.y\ x))\ s)})\ (\lambda x.\lambda y.x)\\
&\rightsquigarrow_\beta \lambda s.\lambda z.\boxed{(\lambda f.f\ ((\lambda f.f\ ((\lambda f.f\ z\ (\lambda x.x))\ (\lambda x.\lambda y.y\ x))\ s)\ (\lambda x.\lambda y.y\ x))\ s)}\ \boxed{(\lambda x.\lambda y.x)}\\
&\rightsquigarrow_\beta \lambda s.\lambda z.\boxed{(\lambda x.\lambda y.x)}\ \boxed{((\lambda f.f\ ((\lambda f.f\ z\ (\lambda x.x))\ (\lambda x.\lambda y.y\ x))\ s)\ (\lambda x.\lambda y.y\ x))}\ \boxed{s}\\
&\rightsquigarrow_\beta^2 \lambda s.\lambda z.\boxed{(\lambda f.f\ ((\lambda f.f\ z\ (\lambda x.x))\ (\lambda x.\lambda y.y\ x))\ s)}\ \boxed{(\lambda x.\lambda y.y\ x)}\\
&\rightsquigarrow_\beta \lambda s.\lambda z.\boxed{(\lambda x.\lambda y.y\ x)}\ \boxed{((\lambda f.f\ z\ (\lambda x.x))\ (\lambda x.\lambda y.y\ x))}\ \boxed{s}\\
&\rightsquigarrow_\beta^2 \lambda s.\lambda z.s\ (\boxed{(\lambda f.f\ z\ (\lambda x.x))}\ \boxed{(\lambda x.\lambda y.y\ x)})\\
&\rightsquigarrow_\beta \lambda s.\lambda z.s\ (\boxed{(\lambda x.\lambda y.y\ x)}\ \boxed{z}\ \boxed{(\lambda x.x)})\\
&\rightsquigarrow_\beta^2 \lambda s.\lambda z.s\ (\boxed{(\lambda x.x)}\ \boxed{z})\\
&\rightsquigarrow_\beta \lambda s.\lambda z.s\ z\\
&\equiv 1
\end{aligned}$$

Once we have defined the PRED term, the subtraction of two natural number terms $m$ and $n$, which respectively represent two natural numbers $a$ and $b$, can be defined by applying $b$ times PRED to $m$. We keep the PRED definition unexpanded due to the size of the term:

$$\mathsf{SUB} \triangleq \lambda m.\lambda n.\lambda s.\lambda z.n\ \mathsf{PRED}\ m\ s\ z$$

$n$ PRED $m$ reduces to the natural number term which applies its parameter $s$ $a$ times but avoiding $b$ of them, starting with $z$, so $s$ is applied $max(0, a - b)$ times.

Although SUB as defined here is a $\beta$-normal term, it is not an $\eta$-normal term, thus it can be expressed in a more concise way:

$$\mathsf{SUB} \rightsquigarrow_{\eta}^{2} \lambda m.\lambda n.n\ \mathsf{PRED}\ m$$

## Exercise 10

Imagine that the list $[a_1, a_2...a_n]$ is represented in the lambda calculus by the term

$$\lambda f.\lambda x.f\ a_1\ (f\ a_2\ (...\ f\ a_n\ x\ ...))$$

Define:

a) NIL, the empty list constructor.

**Solution:** the term which fits the stated encoding when there are no elements is

$$\mathsf{NIL} \triangleq \lambda f.\lambda x.x$$

b) APP, a function to concatenate two lists.

**Solution:** this term can be defined as

$$\mathsf{APP} \triangleq \lambda l.\lambda m.\lambda f.\lambda x.l\ f\ (m\ f\ x)$$

because, if $l$ and $m$ represent the lists $[a_1, a_2...a_n]$ and $[b_1, b_2...b_n]$, then APP $l$ $m$ can be $\beta$-reduced to

$$\lambda f.\lambda x.f\ a_1\ (f\ a_2\ (...\ f\ a_n\ \Big(b_1\ (f\ b_2\ (...\ f\ b_n\ x\ ...))\Big)\ ...))$$

which represents the list $[a_1, a_2...a_n, b_1, b_2...b_n]$.

c) HD, which returns the first element of a nonempty list.

**Solution:** this term can be defined as an abstraction which, given a list $l$, provides as the actual parameter an abstraction that returns its parameter:

$$\mathsf{HD} \triangleq \lambda l.l\ (\lambda x.\lambda y.x)\ nilhead$$

This mentioned parameter is not used when the list is NIL and the second one is used instead, which we have provided as a free variable named *nilhead* to be the result of reducing HD NIL.

These are two examples to show both cases:

$$\begin{aligned}
&\mathsf{HD\ NIL}\\
&\quad \equiv (\lambda l.l\ (\lambda x.\lambda y.x)\ nilhead)\ (\lambda f.\lambda x.x)\\
&\rightsquigarrow_\beta (\lambda f.\lambda x.x)\ (\lambda x.\lambda y.x)\ nilhead\\
&\rightsquigarrow_\beta (\lambda x.x)\ nilhead\\
&\rightsquigarrow_\beta nilhead
\end{aligned}$$

$$\begin{aligned}
&\mathsf{HD}\ (\lambda f.\lambda x.f\ a\ x)\\
&\quad \equiv (\lambda l.l\ (\lambda x.\lambda y.x)\ nilhead)\ (\lambda f.\lambda x.f\ a\ x)\\
&\rightsquigarrow_\beta (\lambda f.\lambda x.f\ a\ x)\ (\lambda x.\lambda y.x)\ nilhead\\
&\rightsquigarrow_\beta (\lambda x.(\lambda x.\lambda y.x)\ a\ x)\ nilhead\\
&\rightsquigarrow_\beta (\lambda x.\lambda y.x)\ a\ nilhead\\
&\rightsquigarrow_\beta^2 a
\end{aligned}$$

d) ISEMPTY, a function to check whether a list is empty.

**Solution:** it can be defined as follows:

$$\mathsf{ISEMPTY} \triangleq \lambda l.l\ (\lambda h.\lambda t.\lambda x.\lambda y.y)\ (\lambda x.\lambda y.x)$$

Given a list $l$, if it is the empty list, then it reduces to the second parameter, which we have provided as the Church encoding TRUE. Otherwise, it will use the second parameter which we have provided as an abstraction that ignores two arguments and becomes into FALSE.

Again, these are two examples to show both different cases:

ISEMPTY NIL

$$
\begin{aligned}
&\equiv(\lambda l.l\ (\lambda h.\lambda t.\lambda x.\lambda y.y)\ (\lambda x.\lambda y.x))\ (\lambda f.\lambda x.x)\\
&\rightsquigarrow_\beta(\lambda f.\lambda x.x)\ (\lambda h.\lambda t.\lambda x.\lambda y.y)\ (\lambda x.\lambda y.x)\\
&\rightsquigarrow_\beta(\lambda x.x)\ (\lambda x.\lambda y.x)\\
&\rightsquigarrow_\beta\lambda x.\lambda y.x\\
&\equiv\mathsf{TRUE}
\end{aligned}
$$

ISEMPTY $(\lambda f.\lambda x.f\ a\ x)$

$$
\begin{aligned}
&\equiv(\lambda l.l\ (\lambda h.\lambda t.\lambda x.\lambda y.y)\ (\lambda x.\lambda y.x))\ (\lambda f.\lambda x.f\ a\ x)\\
&\rightsquigarrow_\beta(\lambda f.\lambda x.f\ a\ x)\ (\lambda h.\lambda t.\lambda x.\lambda y.y)\ (\lambda x.\lambda y.x)\\
&\rightsquigarrow_\beta(\lambda x.(\lambda h.\lambda t.\lambda x.\lambda y.y)\ a\ x)\ (\lambda x.\lambda y.x)\\
&\rightsquigarrow_\beta(\lambda h.\lambda t.\lambda x.\lambda y.y)\ a\ (\lambda x.\lambda y.x)\\
&\rightsquigarrow^2_\beta\lambda x.\lambda y.y\\
&\equiv\mathsf{FALSE}
\end{aligned}
$$