CHAPTER 1

LAMBDA CALCULUS

1.1 Introduction to Lambda Calculus

In this section, we will study the syntax and the sematics of Lambda Calculus, along with some of its properties. Lambda Calculus (denoted λ-calculus) is a mathematical theory of functions, including:

- a formal syntax for defining functions;
- a formal semantics of how functions can be evaluated; and
- a formal theory of function equivalence.

We now define the syntax of λ-calculus. We denote by the set Λ the set of λ-terms. It is the smallest set satisfying the following properties:

- If x is a variable, then $x \in \Lambda$ (there are infinitely many variables);
- If $M \in \Lambda$, then $(\lambda x M) \in \Lambda$ (called *abstraction*);
- If $M, N \in \Lambda$, then $(MN) \in \Lambda$ (called *application*).

This is an inductive definition for Λ. An abstraction represents a function, e.g. $(\lambda x x)$ can be thought of as the Haskell function `\x -> x`. We can define λ-calculus using BNF as well:

$$\begin{aligned} M ::= & \ x \\ & \mid (\lambda x M) \\ & \mid (MM) \end{aligned}$$

Further, we can use inference rules to define it:

$$\frac{\text{if } x \text{ is a variable}^*}{x \in \Lambda} \qquad \frac{M \in \Lambda \text{ and } x \text{ is a variable}^*}{(\lambda x M) \in \Lambda} \qquad \frac{M, N \in \Lambda}{(MN) \in \Lambda}.$$

We typically avoid using brackets for λ-terms (or terms). Moreover, an abstraction $(\lambda x M)$ is denoted $\lambda x.M$. If we have multiple abstractions, we write

$$\lambda x_1 \dots x_k.M \equiv \lambda \vec{x}.M \equiv (\lambda x_1 (\dots (\lambda x_k M) \dots))$$

So, abstraction is right-associative. Now, if we have multiple applications, we write

$$MN_1 \dots N_k \equiv M\vec{N} \equiv (((MN_1) \dots) N_k)$$

So, application is left-associative. The equivalence symbol $\equiv$ denotes syntactic equality.

Free and Bound Variables

The symbol λ binds variables, i.e. it gives rise to local variables. As such, different terms (e.g. $\lambda x.x$ and $\lambda y.y$) are equivalent. Moreover, this gives rise to free variables (not bound by λ) and bound variables (those bound by λ).

We will now define the set of bound variables for a term. It is a function $BV\colon \Lambda \to \mathcal{P}(Var)$, where Var is the set of variables. It is defined inductively:

$$\begin{aligned} BV\ x &= \varnothing \\ BV\ \lambda x.M &= (BV\ M) \cup \{x\} \\ BV\ MN &= (BV\ M) \cup (BV\ N) \end{aligned}$$

We illustrate this with an example. Consider the term $(\lambda y.(\lambda x.zx))y$. Then, we compute its bound variables as follows:

$$\begin{aligned} BV\ (\lambda y.(\lambda x.zx))y &= (BV\ \lambda y.(\lambda x.zx)) \cup (BV\ y) \\ &= ((BV\ \lambda x.zx) \cup \{y\}) \cup \varnothing \\ &= (((BV\ zx) \cup \{x\}) \cup \{y\}) \\ &= ((BV\ z) \cup (BV\ x)) \cup \{x, y\} \\ &= \varnothing \cup \varnothing \cup \{x, y\} \\ &= \{x, y\}. \end{aligned}$$

Next, we define free variables. It too is a function $FV\colon \Lambda \to \mathcal{P}(Var)$, given by:

$$\begin{aligned} FV\ x &= \{x\} \\ FV\ \lambda x.M &= (FV\ M) \setminus \{x\} \\ FV\ MN &= (FV\ M) \cup (FV\ N) \end{aligned}$$

We illustrate this with an example:

$$\begin{aligned} FV\ (\lambda y.(\lambda x.zx))y &= (FV\ \lambda y.(\lambda x.zx)) \cup (FV\ y) \\ &= ((FV\ \lambda x.xz) \setminus \{y\}) \cup \{y\} \\ &= ((FV\ xz \setminus \{x\}) \setminus \{y\}) \cup \{y\} \\ &= (((FV\ x) \cup (FV\ z) \setminus \{x, y\})) \cup \{y\} \\ &= (\{x, z\} \setminus \{x, y\}) \cup \{y\} \\ &= \{y, z\}. \end{aligned}$$

Now, we define subterms of a term. It is a function $Sub\colon \Lambda \to \mathcal{P}(\Lambda)$ defined by:

$$\begin{aligned} Sub\ x &= \{x\} \\ Sub\ \lambda x.M &= (Sub\ M) \cup \{\lambda x.M\} \\ Sub\ MN &= (Sub\ M) \cup (Sub\ N) \cup \{MN\} \end{aligned}$$

We illustrate this with an example:

$$\begin{aligned}
Sub\ (\lambda y.(\lambda x.zx))y &= (Sub\ \lambda y.(\lambda x.zx)) \cup (Sub\ y) \cup \{(\lambda y.(\lambda x.zx))y\} \\
&= (Sub\ \lambda x.zx \cup \{\lambda y.(\lambda x.zx)\}) \cup \{y, (\lambda y.(\lambda x.zx))y\} \\
&= ((Sub\ zx) \cup \{\lambda x.zx\}) \cup \{\lambda y.(\lambda x.zx), y, (\lambda y.(\lambda x.zx))y\} \\
&= ((Sub\ z) \cup (Sub\ x) \cup \{zx\}) \cup \\
&\quad \{\lambda x.zx, \lambda y.(\lambda x.zx), y, (\lambda y.(\lambda x.zx))y\} \\
&= \{z\} \cup \{x\} \cup \{zx, \lambda x.zx, \lambda y.(\lambda x.zx), y, (\lambda y.(\lambda x.zx))y\} \\
&= \{z, x, zx, \lambda x.zx, \lambda y.(\lambda x.zx), y, (\lambda y.(\lambda x.zx))y\}.
\end{aligned}$$

Structural Induction

We will now look at proofs involving structural induction for λ-calculus.

Proposition 1.1.1. *A term in Λ has balanced parentheses.*

Proof. We show that using structural induction on a term:

- In the base case, the term is x, for some variable x. Since there are no parentheses here, the result follows trivially.
- Now, assume that the terms M and N have balanced parentheses. Then, the term (MN) must also have balanced parentheses.
- Finally, assume that the term M has balanced parentheses. Then, the term (λxM) must also have balanced parentheses.

So, the result follows by structural induction. □

Note that we have 2 inductive cases here- this is because there are 3 production rules. We now look at a more complicated example.

Proposition 1.1.2. *Let M be a term. Then, $FV\ M \subseteq Sub\ M$.*

Proof. We prove this by structural induction on M.

- In the base case, we have $M = x$, for some variable x. Then,
$$FV\ M = \{x\} \subseteq \{x\} = Sub\ M.$$
- Now, assume that $M = N_1N_2$, for terms N_1 and N_2 that satisfy $FV\ N_1 \subseteq Sub\ N_1$ and $FV\ N_2 \subseteq Sub\ N_2$. In that case,
$$\begin{aligned}
FV\ M &= (FV\ N_1) \cup (FV\ N_2) \\
&\subseteq (Sub\ N_1) \cup (Sub\ N_2) \\
&\subseteq (Sub\ N_1) \cup (Sub\ N_2) \cup \{M\} = Sub\ M.
\end{aligned}$$
- Finally, assume that $M = \lambda x.N$, for some term N satisfying $FV\ N \subseteq Sub\ N$. Then,
$$\begin{aligned}
FV\ M &= (FV\ N) \setminus \{x\} \\
&\subseteq FV\ N \\
&\subseteq Sub\ N \\
&\subseteq (Sub\ N) \cup \{M\} = Sub\ M.
\end{aligned}$$

So, the result follows from structural induction. □

Contexts

A context is a term containing a *hole*, which is represented by []. This hole can be filled by a term, to make the context a term. It is defined as follows:

$$\begin{aligned} C[] ::= & \ x \\ & \mid [] \\ & \mid (\lambda x C[]) \\ & \mid (C[]C[]) \end{aligned}$$

The variable has no hole; the empty context is []. The context $C[]C'[]$ has two holes. We will now illustrate how to fill a context with a term. So, consider the context $C[] = ((\lambda x.[]x)M)$. Then, $C[\lambda y.y] = ((\lambda x.(\lambda y.y)x)M)$.

We can fill contexts with a hole given one term.

Proposition 1.1.3. *Let $C[]$ be a context with one hole and M a term. Then, $C[M]$ is a term.*

Proof. We show this using structural induction on the context $C[]$.

- First, let $C[] = []$. Since M is a term, we find that $C[M] = M$ is a term.
- Now, let $C[] = \lambda x.C'[]$, where $C'[]$ is a context where $C'[M]$ is a term. Then, $C[M] = \lambda x.C'[M]$ must be a term.

So, the result follows from structural induction. □

In general, we can fill a context with n holes given n terms- this follows from the result above and mathematical induction.

1.2 Reduction

In this section, we will consider how we can evaluate λ-terms, using the β-rule in a cumbersome manner, and later using β-rule with α-equivalence to make it compact and efficient.

β-reduction

The key definition for expressing computation in λ-calculus is the β rule. It states:

$$(\lambda x.M)N \to_\beta M[x := N]$$

The notation $M[x := N]$ is substitution- every occurrence of x in M is replaced by N. For example,

$$(\lambda x.x + 1)2 \to_\beta 2 + 1.$$

We could further reduce this to 3 by extending mathematical operations, like the language Expr. The β rule expresses how we would like to pass parameters into a function.

Substitution is the main aspect of β-reduction. However, naive substitution is not necessarily what we want in general. For instance, if we have the term $(\lambda x.\lambda y.yx)y$, then we can β-reduce it to $\lambda y.yy$. However, this is not what we would like- the first y is meant to be bound by the local y, but the second one is not- it is the value we have just substituted. This would not have happened if we have the term $(\lambda x.\lambda y.yx)z$, in which case we get $\lambda y.yz$. In particular, this happens in the term $(\lambda x.\lambda y.yx)y$ because y is both a free and a bound variable in the term.

Hence, we need to define substitution in a more careful manner. This is done as follows:

1. $x[x := N] \equiv N$;
2. $y[x := N] \equiv y$ if x and y are distinct;
3. $(\lambda x.M)[x := N] \equiv \lambda x.M$;
4. $(\lambda y.M)[x := N] \equiv \lambda y.M[x := N]$ if x is not a free variable in M or y is not a free variable in N;
5. $(\lambda y.M)[x := N] \equiv (\lambda z.M[y := z])[x := N]$ if x is a free variable in M and y a free variable in N;
6. $(M_1M_2)[x := N] \equiv (M_1[x := N])(M_2[x := N])$.

Now, consider the term $(\lambda x.\lambda y.yx)y$. This β-reduces to

$$(\lambda y.yx)[x := y].$$

The variable x is free in $\lambda y.yx$ and the variable y is free in y. Hence, we apply rule 5 to get

$$(\lambda y.yx)[x := y] \equiv (\lambda z.yx[y := z])[x := y].$$

We have $yx[y := z] \equiv zx$ by rules 6 and 1, so

$$(\lambda z.yx[y := z])[x := y] \equiv (\lambda z.zx)[x := y].$$

Now, although x is free in $\lambda z.zx$, z is not free in y, so we finally get

$$\lambda z.zy$$

In programming language terms, this is similar to renaming a variable in a local scope so that it does not mask a variable in an outer scope.

α-equivalence

We will now give another, simpler, definition for substitution. This works by assuming that bound variables have been renamed so that they are different from any free variables. This ensures that variables cannot be captured. This is called Barendregt or variable convention. To define substitution, we define α-equivalence.

Definition 1.2.1. Let M and M' be terms. We say that M' is produced by M by a *change of bound variables* if $M \equiv C[\lambda x.N]$ and $M' \equiv C[\lambda y.N[x := y]]$, where y does not occur in N, and $C[]$ is a context with one hole.

Note that filling a hole in a context is different to substitution- we do not care about variable capture when doing so, unlike in substitution. For instance, if $C[] \equiv \lambda x.x[]$, then $C[x] \equiv \lambda x.xx$, even though substitution would have resulted in a variable change.

Definition 1.2.2. Let M and N be terms. We say that M is *α-equivalent* to N if N is produced from M by a series of changes of bound variable.

For instance, $\lambda x.xy \equiv_\alpha \lambda z.zy$ since we can let $C[] = []y$, and then $C[\lambda x.x] \equiv \lambda x.xy$ and $C[\lambda z.x[x := z]] \equiv \lambda z.zy$. However, $\lambda x.xy$ is not α-equivalent to $\lambda x.xx$. We consider α-equivalent terms to be the same as each other.

From now, we will assume that the terms obey variable convention, i.e. all bound variables are different from each other and from all free variables. If this is not the case, we can achieve this using α-equivalence. With this convention, the definition of substitution simplifies to the following:

1. $x[x := N] \equiv N$;
2. $y[x := N] \equiv y$ if y and x are distinct;
3. $(\lambda y.M)[x := N] \equiv \lambda y.(M[x := N])$;
4. $(M_1 M_2)[x := N] \equiv M_1[x := N]M_2[x := N]$.

We now consider how substitution interacts with two variables.

Lemma 1.2.3 (Substitution Lemma)**.** *Let M and N be terms, and x, y distinct variables with x not free in L. Then,*

$$M[x := N][y := L] \equiv M[y := L][x := N[y := L]].$$

Proof. We prove this by structural induction on M:

- Let $M = z$, where z is a variable distinct to both x and y. In that case,

$$\begin{aligned} M[x := N][y := L] &\equiv z[x := N][y := L] \\ &\equiv z[y := L] \equiv z, \end{aligned}$$

and

$$\begin{aligned} M[y := L][x := N[y := L]] &\equiv z[y := L][x := N[y := L]] \\ &\equiv z[x := N[y := L]] \equiv z. \end{aligned}$$

So, the result holds in this case.

- Now, let $M = x$. Then,

$$\begin{aligned} M[x := N][y := L] &\equiv x[x := N][y := L] \\ &\equiv N[y := L], \end{aligned}$$

and, since x is not free in L,

$$\begin{aligned} M[y := L][x := N[y := L]] &\equiv x[y := L][x := N[y := L]] \\ &\equiv x[x := N[y := L]] \equiv N[y := L]. \end{aligned}$$

So, the result holds in this case.

- Next, let $M = y$. Then,

$$M[x := N][y := L] \equiv y[x := N][y := L] \equiv y[y := L] \equiv L,$$

and, since x is not free in L,

$$M[y := L][x := N[y := L]] \equiv L[x := N[y := L]] \equiv L.$$

So, the result holds in this case.

- Next, let $M = \lambda z.M'$, where z is distinct from x and y and

$$M'[x := N][y := L] \equiv M'[y := L][x := N[y := L]].$$

Then,

$$\begin{aligned} M[x := N][y := L] &\equiv (\lambda z.M')[x := N][y := L] \\ &\equiv (\lambda z.M'[x := N])[y := L] \\ &\equiv (\lambda z.M'[x := N][y := L]) \\ &\equiv (\lambda z.M'[y := L][x := N[y := L]]) \\ &\equiv (\lambda z.M'[y := L])[x := N[y := L]] \\ &\equiv (\lambda z.M')[y := L][x := N[y := L]] \\ &\equiv M[y := L][x := N[y := L]]. \end{aligned}$$

So, the result holds in this case.

- Finally, let $M = M_1 M_2$, where

$$M_1[x := N][y := L] \equiv M_1[y := L][x := N[y := L]]$$
$$M_2[x := N][y := L] \equiv M_2[y := L][x := N[y := L]].$$

Then,

$$\begin{aligned} M[x := N][y := L] &\equiv (M_1 M_2)[x := N][y := L] \\ &\equiv (M_1[x := N][y := L])(M_2[x := N][y := L]) \\ &\equiv (M_1[y := L][x := N[y := L]]) \\ &\quad (M_2[y := L][x := N[y := L]]) \\ &\equiv (M_1 M_2)[y := L][x := N[y := L]] \\ &\equiv M[y := L][x := N[y := L]]. \end{aligned}$$

So, the result holds in this case.

Hence, the result follows from induction. □

We can now complete the definition on evaluating λ-terms. First, we define one-step β-reduction $\to_\beta$, given by the following inference rules:

$$\frac{}{(\lambda x.M)N \to_\beta M[x := N]} \qquad \frac{M \to_\beta N}{MZ \to_\beta NZ}$$
$$\frac{M \to_\beta N}{ZM \to_\beta ZN} \qquad \frac{M \to_\beta N}{\lambda x.M \to_\beta \lambda x.N}$$

We have seen the first definition before. The other three allow us to apply the β-rule in larger terms.

We can define the β-reduction, denoted by $\twoheadrightarrow_\beta$ that is the reflexive and transitive closure of one-step β-reduction. This is defined by the following inference rules:

$$\frac{}{M \twoheadrightarrow_\beta M} \qquad \frac{M \to_\beta N}{M \twoheadrightarrow_\beta N} \qquad \frac{M \twoheadrightarrow_\beta N \quad N \twoheadrightarrow_\beta L}{M \twoheadrightarrow_\beta L}.$$

It turns out that this definition also extends to contexts.

Proposition 1.2.4. *Let $C[]$ be a context with one hole. If $M \to_\beta N$, then $C[M] \to_\beta C[N]$.*

Proof. We prove this by structural induction on the context $C[]$:

- First, let $C[] = []$. In that case, if $M \to_\beta N$, then $C[M] = M \to_\beta N = C[N]$.
- Now, let $C[] = \lambda x.C'[]$, where $C'[]$ is another context such that $C'[M] \to_\beta C'[N]$. In that case, applying rule 4 of $\to_\beta$, we find that $C[M] = \lambda x.C'[M] \to \lambda x.C'[N] = C[N]$.

So, the result follows from induction. □

Proposition 1.2.5. *Let $C[]$ be a context with one hole. If $M \twoheadrightarrow_\beta N$, then $C[M] \twoheadrightarrow_\beta C[N]$.*

Proof. We prove this by structural induction on $\twoheadrightarrow_\beta$:

- If $M \equiv N$, then $C[M] \equiv C[N]$. Hence, $C[M] \twoheadrightarrow_\beta C[N]$.
- Instead, if $M \to_\beta N$, then by the result above, we have $C[M] \to_\beta C[N]$. Hence, $C[M] \twoheadrightarrow_\beta C[N]$.
- Otherwise, we have $M \twoheadrightarrow_\beta L$ and $L \twoheadrightarrow_\beta N$, with $C[M] \twoheadrightarrow_\beta C[L]$ and $C[L] \twoheadrightarrow_\beta C[N]$. Hence, we can apply the third inference rule of $\twoheadrightarrow_\beta$ to conclude that $C[M] \twoheadrightarrow_\beta C[N]$.

So, the result follows from structural induction. □

1.3 Theory of Equality

In this section, we will define the theory of equality between λ-terms. β-reduction is part of the equality, e.g. we say

$$(\lambda x.x + 1)1 = 2.$$

We assume that mathematical computations are part of the λ-calculus. Unlike β-reduction, we also have

$$(\lambda x.x + 1)2 = (\lambda x.x + 2)1$$

We denote the λ formal theory $\lambda \vdash M = N$. This is a collection of axioms and inference rules. The theory of equality must satisfy the following:

- an application term must be equal to the result obtained by applying the function part of the term to the argument.
- equality must be an equivalence relation.
- equal terms should be equal in any context.

We will now go through the rules of λ-theory. The first axiom is β-reduction:

$$\frac{}{(\lambda x.M)N = M[x := N]}.$$

Now, we have reflexivity axiom, and the symmetry and the transitivity inference rules to make equality an equivalence relation.

$$\frac{}{M = M} \qquad \frac{M = N}{N = M} \qquad \frac{M = N \qquad N = L}{M = L}.$$

The remaining rules cover reduction in applications and abstractions.

$$\frac{M = N}{MZ = NZ} \qquad \frac{M = N}{ZM = ZN} \qquad \frac{M = N}{\lambda x.M = \lambda x.N}.$$

If $M = N$ can be derived from the axioms and the inference rules of λ-theory, then we write $\lambda \vdash M = N$. The *true statements* derived from applying axioms and inference rules of λ, are called *theorems*. In this case, $M = N$ is a *theorem*, and we say that M and N are *convertible*. Note that if $M \equiv N$, then $M = N$, but not the other way around, e.g. $(\lambda x.x)y = y$, but they are not syntactically equivalent.

We will now show that equal terms are equal in any context.

Theorem 1.3.1. *Let $C[]$ be a context and M, N be terms. If $\lambda \vdash M = N$, then $\lambda \vdash C[M] = C[N]$.*

Proof. We show this by structural induction:

- If $C[] = x$, then
$$x[M] \equiv x \equiv x[N].$$
- If $C[] = []$, then
$$C[M] \equiv M = N \equiv C[N].$$

- Now, let $C[] = \lambda x.C'[]$, where $C'[]$ is a context satisfying $C'[M] = C'[N]$. Hence,
$$C[M] \equiv \lambda x.C'[M] = \lambda x.C'[N] \equiv C[N],$$
applying the rule
$$\frac{C'[M] = C'[N]}{\lambda x.C'[M] = \lambda x.C'[N]}$$

- Finally, let $C[] = C_1[]C_2[]$, where $C_1[]$ and $C_2[]$ are contexts satisfying $C_1[M] = C_1[N]$ and $C_2[M] = C_2[N]$. Then,
$$C[M] \equiv C_1[M]C_2[M] = C_1[N]C_2[M] = C_1[N]C_2[N] \equiv C[N].$$
We apply the following rules:
$$\frac{M = N}{C_1[M]C_2[M] = C_1[N]C_2[M]} \qquad \frac{M = N}{C_1[N]C_2[M] = C_1[N]C_2[N]}$$
and
$$\frac{C_1[M]C_2[M] = C_1[N]C_2[M] \qquad C_1[N]C_2[M] = C_1[N]C_2[N]}{C_1[M]C_2[M] = C_1[N]C_2[N]}.$$

□

This property is called *referential transparency*. This is an advantage of functional languages that states that the value of the expression is the only thing that matters; not the way it is computed. This property does not hold if expressions have side effects (e.g. changing state or performing input/output).

We now establish the relationship between equality and substitution.

Theorem 1.3.2. *Let M, M', N, N' be terms.*

- *If $M = M'$ then $M[x := N] = M'[x := N]$.*
- *If $N = N'$, then $M[x := N] = M[x := N']$.*
- *If $M = M'$ and $N = N'$, then $M[x := N] = M'[x := N']$.*

Proof.

- By definition, we have
$$M[x := N] \equiv (\lambda x.M)N.$$
Since $M = M'$, we have $\lambda x.M = \lambda x.M'$. Moreover, since $\lambda x.M = \lambda x.M'$, we find that $(\lambda x.M)N = (\lambda x.M')N$. Hence,
$$M[x := N] \equiv (\lambda x.M)N = (\lambda x.M')N \equiv M'[x := N].$$

- We show this by structural induction on substitution.
  - First, let $M = x$. Then,
$$x[x := N] \equiv N = N' \equiv x[x := N'].$$

- Now, let $M = y$, where y is distinct from x. Then,

$$y[x := N] \equiv y \equiv y[x := N'].$$

- Next, let $M = (\lambda x.M')$, where $M'[x := N] = M'[x := N']$. Then, we know that

$$\begin{aligned} M[x := N] &\equiv (\lambda y.M')[x := N] \\ &\equiv \lambda y.(M'[x := N]) \\ &= \lambda y.(M'[x := N']) \\ &\equiv (\lambda y.M')[x := N'] \equiv M[x := N']. \end{aligned}$$

- Finally, let $M = M_1M_2$, where $M_1[x := N] = M_1[x := N']$ and $M_2[x := N] = M_2[x := N']$. Then, we know that

$$\begin{aligned} M[x := N] &\equiv (M_1M_2)[x := N] \\ &\equiv (M_1[x := N])(M_2[x := N]) \\ &= (M_1[x := N'])(M_2[x := N]) \\ &= (M_1[x := N'])(M_2[x := N']) \\ &\equiv (M_1M_2)[x := N'] \equiv M[x := N']. \end{aligned}$$

So, the result follows from induction.

- Since $M[x := N] = M'[x := N]$ and $M'[x := N] = M'[x := N']$, this follows from transitivity.

□

Theorem 1.3.3 (Fixed Point Theorem)**.** *Let $F \in \Lambda$ be a term. Then, there exists a term $X \in \Lambda$ such that $FX = X$.*

Proof. Let $W \equiv \lambda x.F(xx)$ and $X \equiv WW$. Then,

$$X \equiv WW \equiv (\lambda x.F(xx))W = F(WW) \equiv FX.$$

□

We say that X is a *fixed point* of X. The identity function $I \equiv \lambda x.x$ fixes every term, i.e. $IM \equiv (\lambda x.x)M = M$. We will now use the theorem to find a fixed point for the term $F = \lambda xy.xy$. We define

$$W \equiv \lambda x.F(xx) \equiv \lambda x.((\lambda xy.xy)(xx)) = \lambda x.\lambda y.(xx)y \equiv \lambda xy.(xx)y$$

So, the fixed point is

$$X \equiv WW \equiv (\lambda xy.(xx)y)(\lambda xy.(xx)y)$$

We can directly verify if X is a fixed point of F:

$$\begin{aligned} FX &\equiv (\lambda xy.xy)((\lambda xy.(xx)y)(\lambda xy.(xx)y)) \\ &= \lambda y.((\lambda xy.(xx)y)(\lambda xy.(xx)y))y \\ &= (\lambda xy.(xx)y)(\lambda xy.(xx)y) \equiv X. \end{aligned}$$

The fixed point for the identity function is $(\lambda x.xx)(\lambda x.xx)$. We will see this term later. In the fixed point theorem, we have self-application, i.e. subterms of the form xx. This is not typically seen in programming languages, but has huge consequences in λ-calculus.

Now, we will see an application of the fixed point theorem-defining recursive functions. The normal definition of the factorial function is the following:

```
fac(n) = if n == 0 then 1 else n*fac(n-1)
```

Since λ-terms do not have the concept of recursion, we will have to implement this algorithm using the fixed point theorem. To do so, define the term

$$F = \lambda f.\lambda n.\texttt{if}(\texttt{eq}\ n\ 0)1(\texttt{mul}\ n\ (f(n-1))).$$

In this term, assume that the λ-calculus has the terms `if`, `eq` and `mul` with their expected meaning. Then, let `fac` be a fixed point of F. We can evaluate factorial of 1 using this term:

$$\begin{aligned}
\texttt{fac}(1) &= F(\texttt{fac})(1) \\
&\equiv (\lambda f.\lambda n.\texttt{if}(\texttt{eq}\ n\ 0)1(\texttt{mul}\ n(f(n-1))))(\texttt{fac})(1) \\
&= \texttt{if}(\texttt{eq}\ 1\ 0)1(\texttt{mul}\ 1(\texttt{fac}(0))) \\
&= \texttt{fac}(0) \\
&= F(\texttt{fac})(0) \\
&\equiv (\lambda f.\lambda n.\texttt{if}(\texttt{eq}\ n\ 0)1(\texttt{mul}\ n(f(n-1))))(\texttt{fac})(0) \\
&= \texttt{if}(\texttt{eq}\ 0\ 0)1(\texttt{mul}\ 1(\texttt{fac}(-1))) \\
&= 1.
\end{aligned}$$

Clearly, this formula can be used to evaluate higher factorials as well.

Extensionality

The concept of equality we have defined is the convertibility relationship. It says that two terms are equal if they encode the same algorithm. However, there are some terms we would naturally consider to be equal, but we cannot prove that they are equal in λ-theory. For instance, $\lambda x.Mx = M$ cannot be shown in λ. In the extensionality relationship, two terms are considered equal if they give equal results to equal arguments.

We can extend the λ-theory to derive $\lambda x.Mx = M$. We can directly add it as a new axiom, called the η-axiom- it states that $\lambda x.Mx = M$ if x is not free in M. This is the $\lambda\eta$ theory. Another way of doing this is using the ext rule:

$$\frac{Mx = Nx}{M = N}$$

if x is not free in M and N. This is the $\lambda + ext$ theory. We will show that the two theories are equivalent.

Theorem 1.3.4. *$\lambda\eta$ and $\lambda + ext$ are equivalent.*

Proof. Since $\lambda\eta$ and $\lambda + ext$ both extend λ, it suffices to show that their extensions are equivalent.

We first show that $\lambda\eta \vdash Mx = Nx \implies M = N$ if x is not free in M and N. So, let M, N be terms and x a variable not free in M such that $Mx = Nx$. This implies that $\lambda x.Mx = \lambda x.Nx$. Hence, by $\lambda\eta$, we find that

$$M = \lambda x.Mx = \lambda x.Nx = N.$$

So, $\lambda\eta \vdash \lambda + ext$.

Now, we show that $\lambda + ext \vdash \lambda x.Mx = M$ if x is not free in M. So, let M be a term and let x be a variable not free in M. We know that $(\lambda x.Mx)x = Mx$. Hence, by ext, we find that $\lambda x.Mx = M$. So, $\lambda + ext \vdash \lambda\eta$. □

Consistency

For a theory to be useful, nothing every equation can hold- there must be theorems (satisfied equations) and non-theorems (equations not satisfied).

Definition 1.3.5. An *equation* is a formula of the form $M = N$, for terms M and N. It is *closed* if it has no free variables.

Definition 1.3.6. Let $\mathcal{T}$ be a theory with equations as formulae. We say that $\mathcal{T}$ is *consistent*, denoted by $\mathrm{Con}(\mathcal{T})$, if it does not prove every closed equation. If $\mathcal{T}$ is a set of equations, then $\lambda + \mathcal{T}$ is formed by adding the equations of $\mathcal{T}$ as axioms to λ, denoted bt $\mathrm{Con}(\lambda + \mathcal{T})$.

The theories λ and $\lambda\eta$ are consistent. It is quite easily possible to lose consistency, e.g. by adding just a single equation. For instance, define

$$\begin{aligned} S &\equiv \lambda xyz.xz(yz) \\ K &\equiv \lambda xy.x \\ I &\equiv \lambda x.x \end{aligned}$$

If we add the equation $S = K$ to λ or $\lambda\eta$, we get an inconsistent theory. To show this, let D be an arbitrary term. We note that

$$\begin{aligned} SMNO &\equiv MO(NO) \\ KMN &\equiv M \\ IM &\equiv M \end{aligned}$$

for all terms M, N, O. So,

$$S = K \implies SABC = KABC \implies AC(BC) = AC$$

for all terms A, B, C. Now, if $A = C = I$, then

$$AC(BC) = AC \implies BI = I.$$

Finally, if $B = KD$, then

$$BI = I \implies KDI = I \implies D = I.$$

Hence, we have shown that any arbitrary term D is equal to the identity term I. So, the theory is inconsistent.

Definition 1.3.7. We say that the terms M and N are *incompatible*, denoted $M\#N$, if $\neg \mathrm{Con}(M = N)$, i.e. $\lambda + M = N$ is not consistent.

Note that this is different to $M = N$ not being derivable- this means that adding $M = N$ makes the theory inconsistent. From the example we considered above, S and K are incompatible.

We will now show that the terms xx and xy are incompatible. So, assume that $xx = xy$. In that case,

$$\begin{aligned} \lambda xy.xx &= \lambda xy.xy \\ (\lambda xy.xx)MN &= (\lambda xy.xx)MN \\ MM &= MN \end{aligned}$$

for any terms M and N. Now, if $M = I$, then we find that $N = I$, so this theory is also inconsistent.

Next, we show that $x(yz)$ and $(xy)z$ are inconsistent. To see this, assume $x(yz) = (xy)z$. Then,

$$\begin{aligned} \lambda xyz.x(yz) &= \lambda xyz.(xy)z \\ (\lambda xyz.x(yz))MNO &= (\lambda xyz.(xy)z)MNO \\ M(NO) &= (MN)O \end{aligned}$$

for any terms M, N and O. Now, let $M = \lambda xy.y$. Then,

$$P = M(NO)P = (MN)OP = OP.$$

So, if $P = N = I$ and O arbitrary, we find that $O = I$. So, this theory is inconsistent.

We will now look at normal forms.

Definition 1.3.8. Let M be a term.

- We say that M is a β-normal form, denoted β-nf. or nf, if M has no subterms of the form $(\lambda x.R)S$.
- We say that M has a β-normal form if there exists an $N = M$ such that N is β-nf.
- We say that M is a $\beta\eta$-normal form if M is a β-n.f. with no subterms of the form $(\lambda x.Rx)$ where x is free in R.
- We say that M has a $\beta\eta$-normal form if there exists an $N = M$ such that N is $\beta\eta$-nf.

The basic idea of λ-calculus as a programming language is that computation consists of applying the β-rule from left to right and converting $(\lambda x.M)N$ to $M[x := N]$ until we reach a β-nf, which is the result. We show this with some examples:

- The term $\lambda x.x$ is a normal form;
- The term $(\lambda xy.x)(\lambda x.x)$ has a normal form- $\lambda yx.x$;

- The term $(\lambda x.xx)(\lambda x.xx)$ does not have a normal form since it β-reduces to itself.

As we just saw, it might be that a term does not have a normal form.
We will now look at some properties for normal forms.

Proposition 1.3.9. *Let M be a term.*

- *M has a β-nf if and only if M has a $\beta\eta$-nf;*
- *If M and N are distinct β-nfs, then $M = N$ is not a theorem of λ (and similarly for $\lambda\eta$ with $\beta\eta$-nfs);*
- *If M and N are distinct $\beta\eta$-nfs, then $M\#N$.*

Note the final equation states that any two distinct $\beta\eta$-nfs cannot be set equal without making the theory inconsistent. Hence, $\beta\eta$ is the weakest consistent form of equality. This is the property of completeness, given below.

Proposition 1.3.10. *Let M and N be λ-terms. Then, either $\lambda\eta \vdash M = N$ or $\lambda\eta + (M = N)$ is inconsistent.*