Exercise 5. (a) Manually evaluate the lambda terms **add** $\overline{2}\,\overline{3}$ and **mult** $\overline{2}\,\overline{3}$.

(b) Prove that **add** $\overline{n}\,\overline{m} \twoheadrightarrow_\beta \overline{n+m}$, for all natural numbers n, m.

(c) Prove that **mult** $\overline{n}\,\overline{m} \twoheadrightarrow_\beta \overline{n \cdot m}$, for all natural numbers n, m.

Definition. Suppose $f : \mathbb{N}^k \to \mathbb{N}$ is a k-ary function on the natural numbers, and that M is a lambda term. We say that M *(numeralwise) represents* f if for all $n_1, \ldots, n_k \in \mathbb{N}$,

$$M\,\overline{n_1} \ldots \overline{n_k} \twoheadrightarrow_\beta \overline{f(n_1, \ldots, n_k)}\,.$$

This definition makes explicit what it means to be an "encoding". We can say, for instance, that the term **add** $= \lambda nmfx.nf(mfx)$ represents the addition function. The definition generalizes easily to boolean functions, or functions of other datatypes.

Often handy is the function **iszero** from natural numbers to booleans, which is defined by

$$\begin{array}{rcl} \textbf{iszero}\,(0) & = & \text{true} \\ \textbf{iszero}\,(n) & = & \text{false}, \quad \text{if } n \neq 0. \end{array}$$

Convince yourself that the following term is a representation of this function:

$$\textbf{iszero} = \lambda nxy.n(\lambda z.y)x.$$

Exercise 6. Find lambda terms that represent each of the following functions:

(a) $f(n) = (n+3)^2$,

(b) $f(n) = \begin{cases} \text{true} & \text{if } n \text{ is even,} \\ \text{false} & \text{if } n \text{ is odd,} \end{cases}$

(c) **exp** $(n, m) = n^m$,

(d) **pred** $(n) = n - 1$.

Note: part (d) is not easy. In fact, Church believed for a while that it was impossible, until his student Kleene found a solution. (In fact, Kleene said he found the solution while having his wisdom teeth pulled, so his trick for defining the predecessor function is sometimes referred to as the "wisdom teeth trick".)

We have seen how to encode some simple boolean and arithmetic functions. However, we do not yet have a systematic method of constructing such functions. What

we need is a mechanism for defining more complicated functions from simple ones. Consider for example the factorial function, defined by:

$$\begin{array}{rcl} 0! & = & 1 \\ n! & = & n \cdot (n-1)!, \quad \text{if } n \neq 0. \end{array}$$

The encoding of such functions in the lambda calculus is the subject of the next section. It is related to the concept of a fixpoint.

3.3 Fixpoints and recursive functions

Suppose f is a function. We say that x is a *fixpoint* of f if $f(x) = x$. In arithmetic and calculus, some functions have fixpoints, while others don't. For instance, $f(x) = x^2$ has two fixpoints 0 and 1, whereas $f(x) = x + 1$ has no fixpoints. Some functions have infinitely many fixpoints, notably $f(x) = x$.

We apply the notion of fixpoints to the lambda calculus. If F and N are lambda terms, we say that N is a fixpoint of F if $FN =_\beta N$. The lambda calculus contrasts with arithmetic in that *every* lambda term has a fixpoint. This is perhaps the first surprising fact about the lambda calculus we learn in this course.

Theorem 3.1. *In the untyped lambda calculus, every term F has a fixpoint.*

Proof. Let $A = \lambda xy.y(xxy)$, and define $\mathbf{\Theta} = AA$. Now suppose F is any lambda term, and let $N = \mathbf{\Theta} F$. We claim that N is a fixpoint of F. This is shown by the following calculation:

$$\begin{array}{rcl} N & = & \mathbf{\Theta} F \\ & = & AAF \\ & = & (\lambda xy.y(xxy))AF \\ & \twoheadrightarrow_\beta & F(AAF) \\ & = & F(\mathbf{\Theta} F) \\ & = & FN. \end{array}$$

□

The term $\mathbf{\Theta}$ used in the proof is called *Turing's fixpoint combinator*.

The importance of fixpoints lies in the fact that they allow us to solve *equations*. After all, finding a fixpoint for f is the same thing as solving the equation $x = f(x)$. This covers equations with an arbitrary right-hand side, whose left-hand side is x. From the above theorem, we know that we can always solve such equations in the lambda calculus.

To see how to apply this idea, consider the question from the last section, namely, how to define the factorial function. The most natural definition of the factorial function is recursive, and we can write it in the lambda calculus as follows:

$$\textbf{fact}\ n \quad = \quad \textbf{if\_then\_else}\,(\textbf{iszero}\ n)(\overline{1})(\textbf{mult}\ n(\textbf{fact}\,(\textbf{pred}\ n)))$$

Here we have used various abbreviations for lambda terms that were introduced in the previous section. The evident problem with a recursive definition such as this one is that the term to be defined, **fact**, appears both on the left- and the right-hand side. In other words, to find **fact** requires solving an equation!

We now apply our newfound knowledge of how to solve fixpoint equations in the lambda calculus. We start by rewriting the problem slightly:

$$\begin{array}{lcl} \textbf{fact} & = & \lambda n.\,\textbf{if\_then\_else}\,(\textbf{iszero}\ n)(\overline{1})(\textbf{mult}\ n(\textbf{fact}\,(\textbf{pred}\ n))) \\ \textbf{fact} & = & (\lambda f.\lambda n.\,\textbf{if\_then\_else}\,(\textbf{iszero}\ n)(\overline{1})(\textbf{mult}\ n(f(\textbf{pred}\ n))))\,\textbf{fact} \end{array}$$

Let us temporarily write F for the term

$$\lambda f.\lambda n.\,\textbf{if\_then\_else}\,(\textbf{iszero}\ n)(\overline{1})(\textbf{mult}\ n(f(\textbf{pred}\ n))).$$

Then the last equation becomes $\textbf{fact} = F\,\textbf{fact}$, which is a fixpoint equation. We can solve it up to β-equivalence, by letting

$$\begin{array}{lcl} \textbf{fact} & = & \boldsymbol{\Theta} F \\ & = & \boldsymbol{\Theta}(\lambda f.\lambda n.\,\textbf{if\_then\_else}\,(\textbf{iszero}\ n)(\overline{1})(\textbf{mult}\ n(f(\textbf{pred}\ n)))) \end{array}$$

Note that **fact** has disappeared from the right-hand side. The right-hand side is a closed lambda term that represents the factorial function. (A lambda term is called *closed* if it contains no free variables).

To see how this definition works in practice, let us evaluate $\textbf{fact}\ \overline{2}$. Recall from the proof of Theorem 3.1 that $\boldsymbol{\Theta} F \twoheadrightarrow_\beta F(\boldsymbol{\Theta} F)$, therefore $\textbf{fact} \twoheadrightarrow_\beta F\,\textbf{fact}$.

$$\begin{array}{lll} \textbf{fact}\ \overline{2} & \twoheadrightarrow_\beta & F\,\textbf{fact}\ \overline{2} \\ & \twoheadrightarrow_\beta & \textbf{if\_then\_else}\,(\textbf{iszero}\ \overline{2})(\overline{1})(\textbf{mult}\ \overline{2}(\textbf{fact}\,(\textbf{pred}\ \overline{2}))) \\ & \twoheadrightarrow_\beta & \textbf{if\_then\_else}\,(\textbf{F})(\overline{1})(\textbf{mult}\ \overline{2}(\textbf{fact}\,(\textbf{pred}\ \overline{2}))) \\ & \twoheadrightarrow_\beta & \textbf{mult}\ \overline{2}(\textbf{fact}\,(\textbf{pred}\ \overline{2})) \\ & \twoheadrightarrow_\beta & \textbf{mult}\ \overline{2}(\textbf{fact}\ \overline{1}) \\ & \twoheadrightarrow_\beta & \textbf{mult}\ \overline{2}(F\,\textbf{fact}\ \overline{1}) \\ & \twoheadrightarrow_\beta & \ldots \\ & \twoheadrightarrow_\beta & \textbf{mult}\ \overline{2}(\textbf{mult}\ \overline{1}(\textbf{fact}\ \overline{0})) \\ & \twoheadrightarrow_\beta & \textbf{mult}\ \overline{2}(\textbf{mult}\ \overline{1}(F\,\textbf{fact}\ \overline{0})) \\ & \twoheadrightarrow_\beta & \textbf{mult}\ \overline{2}(\textbf{mult}\ \overline{1}(\textbf{if\_then\_else}\,(\textbf{iszero}\ \overline{0})(\overline{1})(\textbf{mult}\ \overline{2}(\textbf{fact}\,(\textbf{pred}\ \overline{2}))))) \\ & \twoheadrightarrow_\beta & \textbf{mult}\ \overline{2}(\textbf{mult}\ \overline{1}(\textbf{if\_then\_else}\,(\textbf{T})(\overline{1})(\textbf{mult}\ \overline{2}(\textbf{fact}\,(\textbf{pred}\ \overline{2}))))) \end{array}$$

$$\begin{array}{ll} \twoheadrightarrow_\beta & \textbf{mult}\ \overline{2}(\textbf{mult}\ \overline{1}\,\overline{1}) \\ \twoheadrightarrow_\beta & \overline{2} \end{array}$$

Note that this calculation, while messy, is completely mechanical. You can easily convince yourself that $\textbf{fact}\ \overline{3}$ reduces to $\textbf{mult}\ \overline{3}(\textbf{fact}\ \overline{2})$, and therefore, by the above calculation, to $\textbf{mult}\ \overline{3}\,\overline{2}$, and finally to $\overline{6}$. It is now a matter of a simple induction to prove that $\textbf{fact}\ \overline{n} \twoheadrightarrow_\beta \overline{n!}$, for any n.

Exercise 7. Write a lambda term that represents the Fibonacci function, defined by

$$f(0) = 1, \qquad f(1) = 1, \qquad f(n+2) = f(n+1) + f(n), \text{for } n \geqslant 2$$

Exercise 8. Write a lambda term that represents the characteristic function of the prime numbers, i.e., $f(n) =$ true if n is prime, and false otherwise.

Exercise 9. We have remarked at the beginning of this section that the number-theoretic function $f(x) = x + 1$ does not have a fixpoint. On the other hand, the lambda term $F = \lambda x.\,\textbf{succ}\ x$, which represents the same function, does have a fixpoint by Theorem 3.1. How can you reconcile the two statements?

Exercise 10. The first fixpoint combinator for the lambda calculus was discovered by Curry. Curry's fixpoint combinator, which is also called the *paradoxical fixpoint combinator*, is the term $\mathbf{Y} = \lambda f.(\lambda x.f(xx))(\lambda x.f(xx))$.

(a) Prove that this is indeed a fixpoint combinator, i.e., that $\mathbf{Y}F$ is a fixpoint of F, for any term F.

(b) Turing's fixpoint combinator not only satisfies $\boldsymbol{\Theta} F =_\beta F(\boldsymbol{\Theta} F)$, but also $\boldsymbol{\Theta} F \twoheadrightarrow_\beta F(\boldsymbol{\Theta} F)$. We used this fact in evaluating $\textbf{fact}\ \overline{2}$. Does an analogous property hold for $\mathbf{Y}$? Does this affect the outcome of the evaluation of $\textbf{fact}\ \overline{2}$?

(c) Can you find another fixpoint combinator, besides Curry's and Turing's?

3.4 Other datatypes: pairs, tuples, lists, trees, etc.

So far, we have discussed lambda terms that represented functions on booleans and natural numbers. However, it is easily possible to encode more general data structures in the untyped lambda calculus. Pairs and tuples are of interest to everybody. The examples of lists and trees are primarily interesting to people with