Chapter 1

Type-free λ-calculus

The *λ-calculus* is a model of computation. It was introduced a few years before another such model, *Turing machines*. With the latter, computation is expressed by reading from and writing to a tape, and performing actions depending on the contents of the tape. Turing machines resemble programs in imperative programming languages, like Java or C.

In contrast, in λ-calculus one is concerned with functions, and these may both take other functions as arguments, and return functions as results. In programming terms, λ-calculus is an extremely simple higher-order, functional programming language.

In this chapter we only cover the *type-free* or *untyped* λ-calculus. Later we introduce several variants where λ-terms are categorized into various *types*.

1.1. A gentle introduction

Computation in λ-calculus is expressed with *λ-terms*. These are similar to the nameless function notation $n \mapsto n^2$ used in mathematics. However, a mathematician employs the latter notation to denote functions as mathematical objects (defined as sets of pairs). In contrast, λ-terms are formal expressions (strings) which, intuitively, express functions and applications of functions in a pure form. Thus, a λ-term is one of the following:

- a *variable*;
- an *abstraction* $\lambda x M$, where x is a variable and M is a λ-term;
- an *application* MN (of M to N), where M and N are λ-terms.

In an abstraction $\lambda x M$, the variable x represents the function argument (or *formal parameter*), and it may occur in the *body* M of the function, but it does not have to. In an application MN there is no restriction on the shape of the *operator* M or the *argument* N; both can be arbitrary λ-terms.

For instance, the λ-term $\mathbf{I} = \lambda x\, x$ intuitively denotes a function that maps any argument to itself, i.e. the identity function. As another example, $\mathbf{K} = \lambda x \lambda y x$ represents a function mapping any argument x to the constant function that always returns x. Finally, $\mathbf{IK}$ expresses application of the function $\mathbf{I}$ to the argument $\mathbf{K}$.

In mathematics we usually write the application of a function, say f, to an argument, say 4, with the argument in parentheses: $f(4)$. In the λ-calculus we would rather write this as $f4$. The use of parentheses cannot be entirely eliminated though. For instance, the notation λxxy would be ambiguous, and we should instead write either $(\lambda xx)y$ if we mean an application of $\mathbf{I}$ to y, or $\lambda x(xy)$ to denote an abstraction on x with the body xy. In the latter case, it is customary to use *dot notation*, i.e. to write $\lambda x.xy$ instead. Similarly we may need parentheses to disambiguate applications; for instance, $\mathbf{I}(\mathbf{KK})$ expresses application of $\mathbf{I}$ to $\mathbf{KK}$

If λxM denotes a function, and N denotes an argument, the "value" of the application $(\lambda xM)N$ can be calculated by substituting N for x in M. The result of such a substitution is denoted by $M[x := N]$, and we formalize the calculation by the β-reduction rule: $(\lambda xM)N \to_\beta M[x := N]$. For instance,

$$(\mathbf{IK})z = ((\lambda xx)\mathbf{K})z \to_\beta x[x := \mathbf{K}]z = \mathbf{K}z = (\lambda y \lambda x y)z \to_\beta \lambda x z.$$

This process of calculating the value of an expression is similar to common practice in mathematics; if $f(n) = n^2$, then $f(4) = 4^2$, and we get from the application $f(4)$ to the result 4^2 by substituting 4 for n in the body of the definition of f. A programming language analogue is the *call-by-name* parameter passing mechanism, where the formal parameter of a procedure is replaced throughout by the actual parameter expression.

The variable x in a λ-abstraction λxM is *bound* (or *local*) within M in much the same way a formal parameter of a procedure is considered local within that procedure. In contrast, a variable y without a corresponding abstraction is called *free* (or *global*) and is similar to a global variable in most programming languages. Thus, x is bound and y is free in $\lambda x.xy$.

Some confusion may arise when we use the same name for bound and free variables. For example, in $x(\lambda x.xy)$, there are obviously two different x's: the free (global) x, and the bound (local) x, which is "shadowing" the free one in the body. If we instead consider the λ-term $x(\lambda z.zy)$, there is no confusion. As another example of confusion, $(\lambda x.xy)[y := x]$ should replace y in $(\lambda x.xy)$ by a free variable x, but $\lambda x.xx$ is not the desired result. In the latter term we have lost the distinction between the formal parameter x and the free variable x (the free variable has been *captured* by the lambda). If we use a bound variable z, the confusion disappears: $(\lambda z.zy)[y := x] = \lambda z.zx$.

A local variable of a procedure can always be renamed without affecting the meaning of the program. Similarly, in λ-calculus we do not care about the names of bound variables; the λ-terms $\lambda x\, x$ and $\lambda y\, y$ both denote the identity

function. Because of this, it is usually assumed that *terms that differ only in their bound variables are identified.* This gives the freedom to choose bound variables so as to avoid any confusion, e.g. variable capture.

1.2. Pre-terms and λ-terms

We now define the notion of a *pre-term* and introduce λ-terms as equivalence classes of pre-terms. The section is rather dull, but necessary to make our formalism precise. However, to understand most of the book, the informal understanding of λ-terms of the preceding section suffices.

1.2.1. Definition. Let Υ denote a countably infinite set of symbols, henceforth called *variables* (also *object variables* or *λ-variables* when other kinds of variables may cause ambiguity). We define the notion of a *pre-term* by induction as follows:

- Every variable is a pre-term.
- If M, N are pre-terms, then (MN) is a pre-term.
- If x is a variable and M is a pre-term, then (λxM) is a pre-term.

The set of all pre-terms is denoted by Λ^-.

Remark. The definition can be summarized by the following *grammar*:

$$M ::= x \mid (MM) \mid (\lambda xM).$$

In the remainder of the book, we will often use this short style of definition.

Pre-terms, as defined above, are fully parenthesized. As the pre-term $(\lambda f((\lambda u(f(uu)))(\lambda v(f(vv)))))$ demonstrates, the heavy use of parentheses is rather cumbersome. We therefore introduce some notational conventions, which are used informally whenever no ambiguity or confusion can arise.

1.2.2. Convention.

(i) The outermost parentheses in a term are omitted.

(ii) Application associates to the left: $((PQ)R)$ is abbreviated (PQR).

(iii) Abstraction associates to the right: $(\lambda x(\lambda yP))$ is abbreviated $(\lambda x\lambda yP)$.

(iv) A sequence of abstractions $(\lambda x_1(\lambda x_2 \ldots (\lambda x_n P)\ldots))$, can be written as $(\lambda x_1 x_2 \ldots x_n.P)$, in which case the outermost parentheses in P (if any) are usually omitted.[1]

[1] The dot represents a left parenthesis whose scope extends as far to the right as possible.

EXAMPLE.

- $(\lambda v(vv))$ may be abbreviated $\lambda v(vv)$ by (i).
- $(((\lambda xx)(\lambda yy))(\lambda zz))$ may be abbreviated $(\lambda xx)(\lambda yy)(\lambda zz)$ by (i), (ii).
- $(\lambda x(\lambda y(xy)))$ is written $\lambda x\lambda y(xy)$ by (i), (iii) or as $\lambda xy.xy$ by (i), (iv).
- $(\lambda f((\lambda u(f(uu)))(\lambda v(f(vv)))))$ is written $\lambda f.(\lambda u.f(uu))(\lambda v.f(vv))$.

1.2.3. DEFINITION. Define the set $\mathrm{FV}(M)$ of *free variables of* M as follows.

$$\begin{array}{lcl} \mathrm{FV}(x) & = & \{x\}; \\ \mathrm{FV}(\lambda xP) & = & \mathrm{FV}(P) - \{x\}; \\ \mathrm{FV}(PQ) & = & \mathrm{FV}(P) \cup \mathrm{FV}(Q). \end{array}$$

EXAMPLE. Let x, y, z be distinct variables; then $\mathrm{FV}((\lambda xx)(\lambda y.xyz)) = \{x, z\}$. There are two *occurrences* of x: one under λx and one under λy. An occurrence of x in M is called *bound* if it is in a part of M with shape λxL, and *free* otherwise. Then $x \in \mathrm{FV}(M)$ iff there is a free occurrence of x in M.

We now define *substitution* of pre-terms. It will only be defined when no variable is captured as a result of the substitution.

1.2.4. DEFINITION. The *substitution of N for x in M,* written $M[x := N]$, is defined iff no free occurrence of x in M is in a part of M with form λyL, where $y \in \mathrm{FV}(N)$. In such cases $M[x := N]$ is given by:[2]

$$\begin{array}{rcl} x[x := N] & = & N; \\ y[x := N] & = & y, \text{ if } x \neq y; \\ (PQ)[x := N] & = & P[x := N]\, Q[x := N]; \\ (\lambda xP)[x := N] & = & \lambda xP; \\ (\lambda yP)[x := N] & = & \lambda yP[x := N], \text{ if } x \neq y. \end{array}$$

REMARK. In the last clause, $y \notin \mathrm{FV}(N)$ or $x \notin \mathrm{FV}(P)$.

1.2.5. LEMMA.

(i) *If $x \notin \mathrm{FV}(M)$ then $M[x := N]$ is defined, and $M[x := N] = M$.*

(ii) *If $M[x := N]$ is defined then $y \in \mathrm{FV}(M[x := N])$ iff either $y \in \mathrm{FV}(M)$ and $x \neq y$ or both $y \in \mathrm{FV}(N)$ and $x \in \mathrm{FV}(M)$.*

(iii) *The substitution $M[x := x]$ is defined and $M[x := x] = M$.*

(iv) *If $M[x := y]$ is defined, then $M[x := y]$ is of the same length as M.*

[2] In our meta-notation, substitution binds stronger than anything else, so in the third clause the rightmost substitution applies to Q, not to $P[x := N]\, Q$.

Proof. Induction on M. As an example we show (i) in some detail. It is clear that $M[x := N]$ is defined. To show that $M[x := N] = M$ consider the following cases. If M is a variable y, then we must have $y \neq x$, and $y[x := N] = y$. If $M = PQ$ then $x \notin \mathrm{FV}(P)$ and $x \notin \mathrm{FV}(Q)$, so by the induction hypothesis $P[x := N] = P$ and $Q[x := N] = Q$. Then also $(PQ)[x := N] = P[x := N]\,Q[x := N] = PQ$. Finally, if M is an abstraction, we may have either $M = \lambda xP$ or $M = \lambda yP$, where $x \neq y$. In the former case, $(\lambda xP)[x := N] = \lambda xP$. In the latter case, we have $x \notin \mathrm{FV}(P)$, so by the induction hypothesis $(\lambda yP)[x := N] = \lambda yP[x := N] = \lambda yP$. □

1.2.6. Lemma. *Assume that $M[x := N]$ is defined, and both $N[y := L]$ and $M[x := N][y := L]$ are defined, where $x \neq y$. If $x \notin \mathrm{FV}(L)$ or $y \notin \mathrm{FV}(M)$ then $M[y := L]$ is defined, $M[y := L][x := N[y := L]]$ is defined, and*

$$M[x := N][y := L] = M[y := L][x := N[y := L]]. \qquad (*)$$

Proof. Induction on M. The main case is when $M = \lambda zQ$, for $z \notin \{x, y\}$. By the assumptions

(i) $x \notin \mathrm{FV}(L)$ or $y \notin \mathrm{FV}(Q)$;

(ii) $z \notin \mathrm{FV}(N)$ or $x \notin \mathrm{FV}(Q)$;

(iii) $z \notin \mathrm{FV}(L)$ or $y \notin \mathrm{FV}(Q[x := N])$.

For the "defined" part, it remains, by the induction hypothesis, to show:

- $z \notin \mathrm{FV}(L)$ or $y \notin \mathrm{FV}(Q)$;
- $z \notin \mathrm{FV}(N[y := L])$ or $x \notin \mathrm{FV}(Q[y := L])$.

For the first property, if $z \in \mathrm{FV}(L)$, then $y \notin \mathrm{FV}(Q[x := N])$, so $y \notin \mathrm{FV}(Q)$. For the second property, assume $x \in \mathrm{FV}(Q[y := L])$. From (i) we have $x \in \mathrm{FV}(Q)$, thus $z \notin \mathrm{FV}(N)$ by (ii). Therefore $z \in \mathrm{FV}(N[y := L])$ could only happen when $y \in \mathrm{FV}(N)$ and $z \in \mathrm{FV}(L)$. Together with $x \in \mathrm{FV}(Q)$ this contradicts (iii). Now $(*)$ follows from the induction hypothesis. □

A special case of the lemma is $M[x := y][y := L] = M[x := L]$, if the substitutions are defined and $y \notin \mathrm{FV}(M)$.

1.2.7. Lemma. *If $M[x{:=}y]$ is defined and $y \notin \mathrm{FV}(M)$ then $M[x{:=}y][y{:=}x]$ is defined, and $M[x{:=}y][y{:=}x] = M$.*

Proof. By induction with respect to M one shows that the substitution is defined. The equation follows from Lemmas 1.2.5(iii) and 1.2.6. □

The next definition formalizes the idea of identifying expressions that "differ only in their bound variables."

1.2.8. DEFINITION. The relation $=_\alpha$ (α-*conversion*) is the least (i.e. smallest) transitive and reflexive relation on Λ^- satisfying the following.

- If $y \notin \mathrm{FV}(M)$ and $M[x := y]$ is defined then $\lambda x M =_\alpha \lambda y.M[x := y]$.
- If $M =_\alpha N$ then $\lambda x M =_\alpha \lambda x N$, for all variables x.
- If $M =_\alpha N$ then $MZ =_\alpha NZ$.
- If $M =_\alpha N$ then $ZM =_\alpha ZN$.

EXAMPLE. Let x, y be different variables. Then $\lambda xy.xy =_\alpha \lambda yx.yx$, but $\lambda x.xy \neq_\alpha \lambda y.yx$.

By Lemma 1.2.7 the relation $=_\alpha$ is symmetric, so we easily obtain:

1.2.9. LEMMA. *The relation of α-conversion is an equivalence relation.*

Strictly speaking, the omitted proof of Lemma 1.2.9 should go by induction with respect to the definition of $=_\alpha$. We prove the next lemma in more detail, to demonstrate this approach.

1.2.10. LEMMA. *If $M =_\alpha N$ then $\mathrm{FV}(M) = \mathrm{FV}(N)$.*

PROOF. Induction on the definition of $M =_\alpha N$. If $M =_\alpha N$ follows from transitivity, i.e. $M =_\alpha L$ and $L =_\alpha N$, for some L, then by the induction hypothesis $\mathrm{FV}(M) = \mathrm{FV}(L)$ and $\mathrm{FV}(L) = \mathrm{FV}(N)$. If $M = N$ (i.e. $M =_\alpha N$ by reflexivity) then $\mathrm{FV}(N) = \mathrm{FV}(M)$. If $M = \lambda x P$ and $N = \lambda y.P[x := y]$, where $y \notin \mathrm{FV}(P)$ and $P[x := y]$ is defined, then by Lemma 1.2.5 we have $\mathrm{FV}(M) = \mathrm{FV}(P) - \{x\} = \mathrm{FV}(P[x := y]) - \{y\} = \mathrm{FV}(N)$. If $M = \lambda x P$ and $N = \lambda x Q$, where $P =_\alpha Q$, then by the induction hypothesis $\mathrm{FV}(P) = \mathrm{FV}(Q)$, so $\mathrm{FV}(M) = \mathrm{FV}(N)$. If $M = PZ$ and $N = QZ$, or $M = ZP$ and $N = ZQ$, where $P =_\alpha Q$, then we use the induction hypothesis. □

1.2.11. LEMMA. *If $M =_\alpha M'$ and $N =_\alpha N'$ then $M[x := N] =_\alpha M'[x := N']$, provided both sides are defined.*

PROOF. By induction on the definition of $M =_\alpha M'$. If $M = M'$ then proceed by induction on M. The only other interesting case is when we have $M = \lambda z P$ and $M' = \lambda y.P[z := y]$, where $y \notin \mathrm{FV}(P)$, and $P[z := y]$ is defined. If $x = z$, then $x \notin \mathrm{FV}(M) = \mathrm{FV}(M')$ by Lemma 1.2.10. Hence $M[x := N] = M =_\alpha M' = M'[x := N']$ by Lemma 1.2.5. The case $x = y$ is similar. So assume $x \notin \{y, z\}$. Since $M[x := N]$ is defined, $x \notin \mathrm{FV}(P)$ or $z \notin \mathrm{FV}(N)$. In the former case $M[x := N] = \lambda z P$ and $x \notin \mathrm{FV}(P[z := y])$, so $M'[x := N'] = \lambda y.P[z := y] =_\alpha \lambda z.P$.

It remains to consider the case when $x \in \mathrm{FV}(P)$ and $z \notin \mathrm{FV}(N)$. Since $M'[x := N'] = (\lambda y.P[z := y])[x := N']$ is defined, we have $y \notin \mathrm{FV}(N')$,

and thus also $y \notin \mathrm{FV}(P[x := N'])$. By Lemma 1.2.6 it then follows that $M'[x{:=}N'] = \lambda y.P[z{:=}y][x{:=}N'] = \lambda y.P[x{:=}N'][z{:=}y] =_\alpha \lambda z.P[x{:=}N']$. By the induction hypothesis $\lambda z.P[x{:=}N'] =_\alpha \lambda z.P[x{:=}N] = M[x{:=}N]$. □

1.2.12. Lemma.

(i) *For all M, N and x, there exists an M' such that $M =_\alpha M'$ and the substitution $M'[x := N]$ is defined.*

(ii) *For all pre-terms M, N, P, and all variables x, y, there exists M', N' such that $M' =_\alpha M$, $N' =_\alpha N$, and the substitutions $M'[x := N']$ and $M'[x := N'][y := P]$ are defined.*

Proof. Induction on M. The only interesting case in part (i) is $M = \lambda y P$ and $x \neq y$. Let z be a variable different from x, not free in N, and not occurring in P at all. Then $P[y := z]$ is defined. By the induction hypothesis applied to $P[y := z]$, there is a P' with $P' =_\alpha P[y := z]$ and such that the substitution $P'[x := N]$ is defined. Take $M' = \lambda z P'$. Then $M' =_\alpha M$ and $M'[x{:=}N] = \lambda z.P'[x{:=}N]$ is defined. The proof of part (ii) is similar. □

1.2.13. Lemma.

(i) *If $MN =_\alpha R$ then $R = M'N'$, where $M =_\alpha M'$ and $N =_\alpha N'$.*

(ii) *If $\lambda x P =_\alpha R$, then $R = \lambda y Q$, for some Q, and there is a term P' with $P =_\alpha P'$ such that $P'[x := y] =_\alpha Q$.*

Proof. Part (i) is by induction with respect to the definition of $MN =_\alpha R$. Part (ii) follows in a similar style. The main case in the proof is transitivity. Assume $\lambda x P =_\alpha R$ follows from $\lambda x P =_\alpha M$ and $M =_\alpha R$. By the induction hypothesis, we have $M = \lambda z N$ and $R = \lambda y Q$, and there are $P' =_\alpha P$ and $N' =_\alpha N$ such that $P'[x := z] =_\alpha N$ and $N'[z := y] =_\alpha Q$. By Lemma 1.2.12(ii) there is $P'' =_\alpha P$ with $P''[x := z]$ and $P''[x := z][z := y]$ defined. Then by Lemma 1.2.11, we have $P''[x := z] =_\alpha N =_\alpha N'$ and thus also $P''[x := z][z := y] =_\alpha Q$. And $P''[x := z][z := y] = P''[x := y]$ by Lemmas 1.2.5(iii) and 1.2.6. (Note that $z \notin \mathrm{FV}(P'')$ or $x = z$.) □

We are ready to define the true objects of interest: λ-terms.

1.2.14. Definition. Define the set Λ of *λ-terms* as the quotient set of $=_\alpha$:

$$\Lambda \quad = \quad \{[M]_\alpha \mid M \in \Lambda^-\},$$

where[3] $[M]_\alpha = \{N \in \Lambda^- \mid M =_\alpha N\}$.

[3] For simplicity we write $[M]_\alpha$ instead of $[M]_{=_\alpha}$.

EXAMPLE. Thus, for every variable x, the string λxx is a pre-term, while $\mathbf{I} = [\lambda xx]_\alpha = \{\lambda xx, \lambda yy, \ldots\}$, where $x, y, \ldots$ are all the variables, is a λ-term.

WARNING. The notion of a pre-term and the explicit distinction between pre-terms and λ-terms are not standard in the literature. Rather, it is customary to call our pre-terms λ-terms and remark that "α-equivalent terms are identified" (cf. the preceding section).

We can now "lift" the notions of free variables and substitution.

1.2.15. DEFINITION. The free variables $\mathrm{FV}(M)$ of a λ-term M are defined as follows. Let $M = [M']_\alpha$. Then

$$\mathrm{FV}(M) = \mathrm{FV}(M'). \qquad (*)$$

If $\mathrm{FV}(M) = \varnothing$ then we say that M is *closed* or that it is a *combinator*.

Lemma 1.2.10 ensures that any choice of M' yields the same result.

1.2.16. DEFINITION. For λ-terms M and N, we define $M[x := N]$ as follows. Let $M = [M']_\alpha$ and $N = [N']_\alpha$, where $M'[x := N']$ is defined. Then

$$M[x := N] = [M'[x := N']]_\alpha.$$

Here Lemma 1.2.11 ensures that any choice of M' and N' yields the same result, and Lemma 1.2.12 guarantees that suitable M' and N' can be chosen.

The term formation operations can themselves be lifted.

1.2.17. NOTATION. Let P and Q be λ-terms, and let x be a variable. Then PQ, λxP, and x denote the following unique λ-terms:

$$\begin{array}{lcll} PQ & = & [P'Q']_\alpha\,, & \text{where } [P']_\alpha = P \text{ and } [Q']_\alpha = Q; \\ \lambda xP & = & [\lambda xP']_\alpha\,, & \text{where } [P']_\alpha = P; \\ x & = & [x]_\alpha. & \end{array}$$

Using this notation, we can show that the equations defining free variables and substitution for pre-terms also hold for λ-terms. We omit the easy proofs.

1.2.18. LEMMA. *The following equations are valid:*

$$\begin{array}{lcl} \mathrm{FV}(x) & = & \{x\}; \\ \mathrm{FV}(\lambda xP) & = & \mathrm{FV}(P) - \{x\}; \\ \mathrm{FV}(PQ) & = & \mathrm{FV}(P) \cup \mathrm{FV}(Q). \end{array}$$

1.2.19. LEMMA. *The following equations on λ-terms are valid:*

$$\begin{array}{r} x[x := N] = N; \\ y[x := N] = y, \text{ if } x \neq y; \\ (PQ)[x := N] = P[x := N]\, Q[x := N]; \\ (\lambda yP)[x := N] = \lambda y.P[x := N], \end{array}$$

where $x \neq y$ and $y \notin \mathrm{FV}(N)$ in the last clause.

The next lemma "lifts" a few properties of pre-terms to the level of terms.

1.2.20. Lemma. *Let P, Q, R, L be λ-terms. Then*

(i) $\lambda x P = \lambda y.P[x := y]$, *if* $y \notin \mathrm{FV}(P)$.

(ii) $P[x := Q][y := R] = P[y := R][x := Q[y := R]]$, *if* $y \neq x \notin \mathrm{FV}(R)$.

(iii) $P[x := y][y := Q] = P[x := Q]$, *if* $y \notin \mathrm{FV}(P)$.

(iv) *If* $PQ = RL$ *then* $P = R$ *and* $Q = L$.

(v) *If* $\lambda y P = \lambda z Q$, *then* $P[y := z] = Q$ *and* $Q[z := y] = P$.

Proof. An easy consequence of Lemmas 1.2.6 and 1.2.11–1.2.13. □

From now on, expressions involving abstractions, applications, and variables are always understood as λ-terms, as defined in Notation 1.2.17. In particular, with the present machinery at hand, we can formulate definitions by induction on the structure of λ-terms, rather than first introducing the relevant notions for pre-terms and then lifting. The following definition is the first example of this; its correctness is established in Exercise 1.3.

1.2.21. Definition. For $M, \vec{N} \in \Lambda$ and distinct variables $\vec{x} \in \Upsilon$, the *simultaneous substitution of $\vec{N}$ for $\vec{x}$ in M* is the term $M[\vec{x} := \vec{N}]$, such that

$$\begin{aligned} x_i[\vec{x} := \vec{N}] &= N_i; \\ y\,[\vec{x} := \vec{N}] &= y, \quad \text{if } y \neq x_i, \text{ for all } i; \\ (PQ)[\vec{x} := \vec{N}] &= P[\vec{x} := \vec{N}]Q[\vec{x} := \vec{N}]; \\ (\lambda y P)[\vec{x} := \vec{N}] &= \lambda y.P[\vec{x} := \vec{N}], \end{aligned}$$

where, in the last clause, $y \neq x_i$ and $y \notin \mathrm{FV}(N_i)$, for all i.

Other syntax. In this book we define many different languages (logics and λ-calculi) with various binding operators (e.g. quantifiers). Expressions (terms, formulas, types etc.) that differ only in their bound variables are always identified as we just did it in the untyped λ-calculus. However, in order not to exhaust the reader, we generally present the syntax in a slightly informal manner, thus avoiding the explicit introduction of "pre-expressions."

In all such cases, however, we actually have to deal with equivalence classes of some α-conversion relation, and a precise definition of the syntax must take this into account. We believe that the reader is able in each case to reconstruct all missing details of such a definition.

1.3. Reduction

1.3.1. DEFINITION. A relation $\succ$ on Λ is *compatible* iff it satisfies the following conditions for all $M, N, Z \in \Lambda$.

(i) If $M \succ N$ then $\lambda x M \succ \lambda x N$, for all variables x.

(ii) If $M \succ N$ then $MZ \succ NZ$.

(iii) If $M \succ N$ then $ZM \succ ZM$.

1.3.2. DEFINITION. The least compatible relation $\to_\beta$ on Λ satisfying

$$(\lambda x.P)Q \to_\beta P[x := Q],$$

is called *β-reduction*. A term of form $(\lambda x P)Q$ is called a *β-redex,* and the term $P[x := Q]$ is said to arise by *contracting* the redex. A λ-term M is in *β-normal form* (notation $M \in \mathrm{NF}_\beta$) iff there is no N such that $M \to_\beta N$, i.e. M does not contain a β-redex.

1.3.3. DEFINITION.

(i) The relation $\twoheadrightarrow_\beta$ (*multi-step β-reduction*) is the transitive and reflexive closure of $\to_\beta$. The transitive closure of $\to_\beta$ is denoted by $\twoheadrightarrow_\beta^+$.

(ii) The relation $=_\beta$ (called *β-equality* or *β-conversion*) is the least equivalence relation containing $\to_\beta$.

(iii) A *β-reduction sequence* is a finite or infinite sequence

$$M_0 \to_\beta M_1 \to_\beta M_2 \to_\beta \cdots$$

1.3.4. REMARK. The above notation applies in general: for any relation $\to_\circ$, the symbol $\twoheadrightarrow_\circ^+$ (respectively $\twoheadrightarrow_\circ$) denotes the transitive (respectively transitive and reflexive) closure of $\to_\circ$, and the symbol $=_\circ$ is used for the corresponding equivalence. We often omit β (or in general $\circ$) from such notation when no confusion arises, with one exception: the symbol $=$ without any qualification always denotes ordinary equality. That is, we write $A = B$ when A and B denote the same object.

WARNING. In the literature, and contrary to the use in this book, the symbol $=$ is often used for β-equality.

1.3.5. EXAMPLE.

(i) $(\lambda x.xx)(\lambda z z) \to_\beta (xx)[x := \lambda z z] = (\lambda z z)(\lambda y y)$.

(ii) $(\lambda z z)(\lambda y y) \to_\beta z[z := \lambda y y] = \lambda y y$.

(iii) $(\lambda x.xx)(\lambda z z) \twoheadrightarrow_\beta \lambda y y$.

(iv) $(\lambda x x)yz =_\beta y((\lambda x x)z)$.

1.3.6. Example (Some common λ-terms).

(i) Let $\mathbf{I} = \lambda x.x$, $\mathbf{K} = \lambda xy.x$, and $\mathbf{S} = \lambda xyz.xz(yz)$. Then $\mathbf{SKK} \twoheadrightarrow_\beta \mathbf{I}$.

(ii) Let $\omega = \lambda x.xx$ and $\mathbf{\Omega} = \omega\omega$. Then $\mathbf{\Omega} \to_\beta \mathbf{\Omega} \to_\beta \mathbf{\Omega} \to_\beta \cdots$.

(iii) Let $\mathbf{Y} = \lambda f.(\lambda x.f(xx))(\lambda x.f(xx))$. Then $\mathbf{Y} \to_\beta \mathbf{Y}' \to_\beta \mathbf{Y}'' \to_\beta \cdots$, where $\mathbf{Y}, \mathbf{Y}', \mathbf{Y}'', \ldots$ are all different.

1.3.7. Remark. A term is in normal form iff it is an abstraction $\lambda x M$, where M is in normal form, or it is $x M_1 \ldots M_n$, where $n \geq 0$ and all M_i are in normal form ("if" is obvious and "only if" is by induction on the length of M). Even more compact: a normal form is $\lambda y_1 \ldots y_m.x M_1 \ldots M_n$, where $m, n \geq 0$ and all M_i are normal forms.

The following little properties are constantly used.

1.3.8. Lemma.

(i) *If $N \to_\beta N'$ then $M[x := N] \twoheadrightarrow_\beta M[x := N']$.*

(ii) *If $M \to_\beta M'$ then $M[x := N] \to_\beta M'[x := N]$.*

Proof. By induction on M and $M \to_\beta M'$ using Lemma 1.2.20. □

In addition to β-reduction, other notions of reduction are considered in the λ-calculus. In particular, we have *η-reduction*.

1.3.9. Definition. Let $\to_\eta$ denote the least compatible relation satisfying

$$\lambda x.Mx \to_\eta M, \quad \text{if } x \notin \mathrm{FV}(M).$$

The symbol $\to_{\beta\eta}$ denotes the union of $\to_\beta$ and $\to_\eta$.

Remark. In general, when we have compatible relations $\to_R$ and $\to_Q$, the union is written $\to_{RQ}$. Similar notation is used for more than two relations.

The motivation for this notion of reduction (and the associated notion of equality) is, informally speaking, that two functions should be considered equal if they yield equal results whenever applied to equal arguments. Indeed:

1.3.10. Proposition. *Let $=_{ext}$ be the least equivalence relation such that:*

- *If $M =_\beta N$, then $M =_{ext} N$;*
- *If $Mx =_{ext} Nx$ and $x \notin FV(M) \cup FV(N)$, then $M =_{ext} N$;*
- *If $P =_{ext} Q$, then $PZ =_{ext} QZ$ and $ZP =_{ext} ZQ$.*

Then $M =_{ext} N$ iff $M =_{\beta\eta} N$.

PROOF. The implication from left to right is by induction on the definition of $M =_{ext} N$, and from right to left is by induction with respect to the definition of $M =_{\beta\eta} N$. Note that the definition of $=_{ext}$ does not include the rule "If $P =_{ext} Q$ then $\lambda x P =_{ext} \lambda x Q$." This is no mistake: this property (known as *rule* ξ) follows from the others. □

We do not take $\to_{\beta\eta}$ as our standard notion of reduction. We want to be able to distinguish between two algorithms, even if their input-output behaviour is the same. Nevertheless, many properties of $\beta\eta$-reduction are similar to those of β-reduction. For instance, we have the following.

1.3.11. LEMMA. *If there is an infinite $\beta\eta$-reduction sequence starting with a term M then there is an infinite β-reduction sequence from M.*

PROOF. First observe that in an infinite $\beta\eta$-reduction sequence there must be infinitely many β-reduction steps (cf. Exercise 1.6). These β-reduction steps can be "permuted forward", yielding an infinite β-reduction. Indeed, by induction with respect to $M \to_\eta N$, one can show that $M \to_\eta N \to_\beta L$ implies $M \to_\beta P \twoheadrightarrow_{\beta\eta} L$, for some P. □

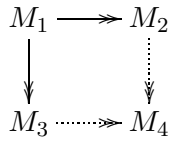
OTHER SYNTAX. In the numerous lambda-calculi occurring in the later chapters, many notions introduced here will be used in an analogous way. For instance, the notion of a compatible relation (Definition 1.3.1) generalizes naturally to other syntax. A compatible relation "respects" the syntactic constructions. Imagine that, in addition to application and abstraction, we have in our syntax an operation of "acclamation," written as $M!!$, i.e. whenever M is a term, $M!!$ is also a term. Then we should add the clause

- If $M \succ N$ then $M!! \succ N!!$.

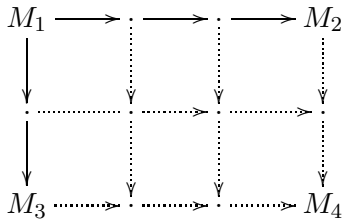
to our definition of compatibility. Various additional reduction relations will also be considered later, and we usually define these by stating one or more reduction axioms, similar to the β-rule of Definition 1.3.2 and the η-rule of Definition 1.3.9. In such cases, we usually assume that the reduction relation is the least compatible relation satisfying the given reduction axiom.

1.4. The Church-Rosser theorem

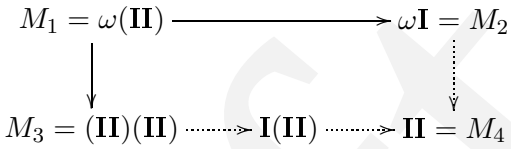
Since a λ-term M may contain several β-redexes, there may be several N with $M \to_\beta N$. For instance, $\mathbf{K}(\mathbf{II}) \to_\beta \lambda x.\mathbf{II}$ and $\mathbf{K}(\mathbf{II}) \to_\beta \mathbf{KI}$. However, as shown below, there must be some term to which all such N reduce in one or more steps. In fact, even if we make several reduction steps, we can still converge to a common term (possibly using several steps):



This property is known as *confluence* or the *Church-Rosser property.* If the above diagram was correct in a stronger version with $\to$ in place of $\twoheadrightarrow$, then we could prove the theorem by a *diagram chase:*



Unfortunately, our prerequisite fails. For instance, in the diagram



two reductions are needed to get from M_3 to M_4. The problem is that the redex contracted in the reduction from M_1 to M_2 is duplicated in the reduction to M_3. We can solve the problem by working with *parallel reduction,* i.e. an extension of $\to_\beta$ allowing such duplications to be contracted in one step.

1.4.1. Definition. Let $\Rightarrow_\beta$ be the least relation on Λ such that:

- $x \Rightarrow_\beta x$ for all variables x.
- If $P \Rightarrow_\beta Q$ then $\lambda x.P \Rightarrow_\beta \lambda x.Q$.
- If $P_1 \Rightarrow_\beta Q_1$ and $P_2 \Rightarrow_\beta Q_2$ then $P_1P_2 \Rightarrow_\beta Q_1Q_2$.
- If $P_1 \Rightarrow_\beta Q_1$ and $P_2 \Rightarrow_\beta Q_2$ then $(\lambda x.P_1)P_2 \Rightarrow_\beta Q_1[x := Q_2]$.

1.4.2. Lemma.

(i) *If* $M \to_\beta N$ *then* $M \Rightarrow_\beta N$.

(ii) *If* $M \Rightarrow_\beta N$ *then* $M \twoheadrightarrow_\beta N$.

(iii) *If* $M \Rightarrow_\beta M'$ *and* $N \Rightarrow_\beta N'$ *then* $M[x{:=}N] \Rightarrow_\beta M'[x{:=}N']$.

Proof. (i) is by induction on the definition of $M \to_\beta N$ (note that $P \Rightarrow_\beta P$ for all P), and (ii), (iii) are by induction on the definition of $M \Rightarrow_\beta M'$. □

1.4.3. Definition. Let M^* (*the complete development of* M) be defined by:

$$\begin{array}{lcl} x^* & = & x; \\ (\lambda x M)^* & = & \lambda x M^*; \\ (MN)^* & = & M^*N^*, \text{ if } M \text{ is not an abstraction;} \\ ((\lambda x M)N)^* & = & M^*[x := N^*]. \end{array}$$

Note that $M \Rightarrow_\beta N$ if N arises by reducing *some* of the redexes present in M, and that M^* arises by reducing *all* redexes present in M.

1.4.4. LEMMA. *If $M \Rightarrow_\beta N$ then $N \Rightarrow_\beta M^*$. In particular, if $M_1 \Rightarrow_\beta M_2$ and $M_1 \Rightarrow_\beta M_3$ then $M_2 \Rightarrow_\beta M_1^*$ and $M_3 \Rightarrow_\beta M_1^*$.*

PROOF. By induction on the definition of $M \Rightarrow_\beta N$, using Lemma 1.4.2. □

1.4.5. THEOREM (Church and Rosser). *If $M_1 \twoheadrightarrow_\beta M_2$ and $M_1 \twoheadrightarrow_\beta M_3$, then there is $M_4 \in \Lambda$ with $M_2 \twoheadrightarrow_\beta M_4$ and $M_3 \twoheadrightarrow_\beta M_4$.*

PROOF. If $M_1 \to_\beta \cdots \to_\beta M_2$ and $M_1 \to_\beta \cdots \to_\beta M_3$, the same holds with $\Rightarrow_\beta$ in place of $\to_\beta$. By Lemma 1.4.4 and a diagram chase, $M_2 \Rightarrow_\beta \cdots \Rightarrow_\beta M_4$ and $M_3 \Rightarrow_\beta \cdots \Rightarrow_\beta M_4$ for some M_4. Then $M_2 \twoheadrightarrow_\beta M_4$ and $M_3 \twoheadrightarrow_\beta M_4$. □

1.4.6. COROLLARY.

(i) *If $M =_\beta N$, then $M \twoheadrightarrow_\beta L$ and $N \twoheadrightarrow_\beta L$ for some L.*

(ii) *If $M \twoheadrightarrow_\beta N_1$ and $M \twoheadrightarrow_\beta N_2$ for β-normal forms N_1, N_2, then $N_1 = N_2$.*

(iii) *If there are β-normal forms L_1 and L_2 such that $M \twoheadrightarrow_\beta L_1$, $N \twoheadrightarrow_\beta L_2$, and $L_1 \not\equiv L_2$, then $M \neq_\beta N$.*

PROOF. Left to the reader. □

REMARK. One can consider λ-calculus as an equational theory, i.e. a formal theory with formulas of the form $M =_\beta N$. Part (i) establishes *consistency* of this theory, in the following sense: there exists a formula that cannot be proved, e.g. $\lambda x x =_\beta \lambda x y.x$ cf. Exercise 2.5).

Part (ii) in the corollary is similar to the fact that when we calculate the value of an arithmetical expression, e.g. $(4+2)\cdot(3+7)\cdot 11$, the end result is independent of the order in which we do the calculations.

1.5. Leftmost reductions are normalizing

1.5.1. DEFINITION. A term M is *normalizing* (notation $M \in \mathrm{WN}_\beta$) iff there is a reduction sequence from M ending in a normal form N. We then say that M *has* the normal form N. A term M is *strongly normalizing* ($M \in \mathrm{SN}_\beta$ or just $M \in \mathrm{SN}$) if all reduction sequences starting with M are finite. We write $M \in \infty_\beta$ if $M \notin \mathrm{SN}_\beta$. Similar notation applies to other notions of reduction.

Any strongly normalizing term is also normalizing, but the converse is not true, as $\mathbf{KI\Omega}$ shows. But Theorem 1.5.8 states that a normal form, if it exists, can always be found by repeatedly reducing the leftmost redex, i.e. the redex whose λ is the furthest to the left. The following notation and definition are convenient for proving Theorem 1.5.8.

VECTOR NOTATION. Let $n \geq 0$. If $\vec{P} = P_1, \ldots, P_n$, then we write $M\vec{P}$ for $MP_1 \ldots P_n$. In particular, if $n = 0$, i.e. $\vec{P}$ is empty, then $M\vec{P}$ is just M. Similarly, if $\vec{z} = z_1, \ldots, z_n$, then we write $\lambda\vec{z}.M$ for $\lambda z_1 \ldots z_n.M$. Again, $\lambda\vec{z}.M$ is just M, if $n = 0$, i.e. $\vec{z}$ is empty.

REMARK. Any term has exactly one of the following two forms: $\lambda\vec{z}.x\vec{R}$ or $\lambda\vec{z}.(\lambda xP)Q\vec{R}$, in which case $(\lambda xP)Q$ is called *head* redex (in the former case, there is no head redex). Any redex that is not a head redex is called *internal*. A head redex is always the leftmost redex, but the leftmost redex in a term is not necessarily a head redex—it may be internal.

1.5.2. DEFINITION. For a term M not in normal form, we write

- $M \xrightarrow{l}_\beta N$ if N arises from M by contracting the leftmost redex.
- $M \xrightarrow{h}_\beta N$ if N arises from M by contracting a head redex.
- $M \xrightarrow{i}_\beta N$ if N arises from M by contracting an internal redex.

1.5.3. LEMMA.

(i) *If* $M \xrightarrow{h}_\beta N$ *then* $\lambda xM \xrightarrow{h}_\beta \lambda xN$.

(ii) *If* $M \xrightarrow{h}_\beta N$ *and* M *is not an abstraction, then* $ML \xrightarrow{h}_\beta NL$.

(iii) *If* $M \xrightarrow{h}_\beta N$ *then* $M[x := L] \xrightarrow{h}_\beta N[x := L]$.

PROOF. Easy. □

The following technical notions are central in the proof of Theorem 1.5.8.

1.5.4. DEFINITION. We write $\vec{P} \Rightarrow_\beta \vec{Q}$ if $\vec{P} = P_1, \ldots, P_n$, $\vec{Q} = Q_1, \ldots, Q_n$, $n \geq 0$, and $P_j \Rightarrow_\beta Q_j$ for all $1 \leq j \leq n$. *Parallel internal reduction* $\xRightarrow{i}_\beta$ is the least relation on Λ satisfying the following rules.

- If $\vec{P} \Rightarrow_\beta \vec{Q}$ then $\lambda\vec{x}.y\vec{P} \xRightarrow{i}_\beta \lambda\vec{x}.y\vec{Q}$.
- If $\vec{P} \Rightarrow_\beta \vec{Q}$, $S \Rightarrow_\beta T$ and $R \Rightarrow_\beta U$, then $\lambda\vec{x}.(\lambda yS)R\vec{P} \xRightarrow{i}_\beta \lambda\vec{x}.(\lambda yT)U\vec{Q}$.

REMARK. If $M \xrightarrow{i}_\beta N$, then $M \xRightarrow{i}_\beta N$. Conversely, if $M \xRightarrow{i}_\beta N$, then $M \xrightarrow{i}{}\!\!\!\!\rightarrow_\beta N$. Also, if $M \xRightarrow{i}_\beta N$, then $M \Rightarrow_\beta N$.

1.5.5. DEFINITION. We write $M \Rrightarrow_\beta N$ if there are $M_0, M_1, \ldots, M_n$ with

$$M = M_0 \xrightarrow{h}_\beta M_1 \xrightarrow{h}_\beta \cdots \xrightarrow{h}_\beta M_n \xRightarrow{i}_\beta N$$

and $M_i \Rightarrow_\beta N$ for all $i \in \{0, \ldots, n\}$, where $n \geq 0$.

1.5.6. LEMMA.

(i) *If* $M \Rrightarrow_\beta M'$ *then* $\lambda x M \Rrightarrow_\beta \lambda x M'$.

(ii) *If* $M \Rrightarrow_\beta M'$ *and* $N \Rightarrow_\beta N'$, *then* $MN \Rrightarrow_\beta M'N'$.

(iii) *If* $M \Rrightarrow_\beta M'$ *and* $N \Rrightarrow_\beta N'$, *then* $M[x := N] \Rrightarrow_\beta M'[x := N']$.

PROOF. Part (i) is easy. For (ii), let

$$M = M_0 \overset{h}{\rightarrow}_\beta M_1 \overset{h}{\rightarrow}_\beta \cdots \overset{h}{\rightarrow}_\beta M_n \overset{i}{\Rightarrow}_\beta M'$$

where $M_i \Rightarrow_\beta M'$ for all $i \in \{0, \ldots, n\}$. Assume at least one M_i is an abstraction, and let k be the smallest number such that M_k is an abstraction. Then $M_k N \overset{i}{\Rightarrow}_\beta M'N'$. By Lemma 1.5.3:

$$MN = M_0 N \overset{h}{\rightarrow}_\beta M_1 N \overset{h}{\rightarrow}_\beta \cdots \overset{h}{\rightarrow}_\beta M_k N \overset{i}{\Rightarrow}_\beta M'N', \qquad (*)$$

where $M_i N \Rightarrow_\beta M'N'$. If there is no abstraction among M_i, $0 \leq i \leq n$, then $(*)$ still holds with $k = n$.

For (iii) first assume $M \overset{i}{\Rightarrow}_\beta M'$. We have either $M = \lambda\vec{z}.(\lambda y.P)Q\vec{R}$ or $M = \lambda\vec{z}.y\vec{R}$. In the former case, $M' = \lambda\vec{z}.(\lambda y.P')Q'\vec{R}'$, where $P \Rightarrow_\beta P'$, $Q \Rightarrow_\beta Q'$, and $\vec{R} \Rightarrow_\beta \vec{R}'$. By Lemma 1.4.2, $M[x := N] \overset{i}{\Rightarrow}_\beta M'[x := N']$. In the latter case, $M' = \lambda\vec{z}.y\vec{R}'$. We consider two possibilities. If $x \neq y$, we proceed as in the case just considered. If $x = y$, let $\vec{S}$ and $\vec{S}'$ arise from $\vec{R}$ (respectively $\vec{R}'$) by substituting N (respectively N') for x. By (i), (ii) and Lemma 1.4.2 we then have $M[x := N] = \lambda\vec{z}.N\vec{S} \Rrightarrow_\beta \lambda\vec{z}.N'\vec{S}' = M'[x := N']$.

Now consider the general case. By the above and Lemma 1.5.3, we have

$$M[x := N] = M_0[x := N] \overset{h}{\rightarrow}_\beta \cdots \overset{h}{\rightarrow}_\beta M_n[x := N] \Rrightarrow_\beta M'[x := N'].$$

Also, $M_i[x := N] \Rightarrow_\beta M'[x := N']$ holds by Lemma 1.4.2. □

1.5.7. LEMMA.

(i) *If* $M \Rightarrow_\beta N$ *then* $M \overset{h}{\twoheadrightarrow}_\beta L \overset{i}{\Rightarrow}_\beta N$ *for some* L.

(ii) *If* $M \overset{i}{\Rightarrow}_\beta N \overset{h}{\rightarrow}_\beta L$, *then* $M \overset{h\,+}{\twoheadrightarrow}_\beta O \overset{i}{\Rightarrow}_\beta L$ *for some* O.

PROOF. For (i), show that $M \Rightarrow_\beta N$ implies $M \Rrightarrow_\beta N$. For (ii), note that $M = \lambda\vec{z}.(\lambda x.P)Q\vec{R}$, $N = \lambda\vec{z}.(\lambda x.P')Q'\vec{R}'$, where $P \Rightarrow_\beta P'$, $Q \Rightarrow_\beta Q'$, and $\vec{R} \Rightarrow_\beta \vec{R}'$. Then $L = \lambda\vec{z}.P'[x := Q']\vec{R}'$. Define $O = \lambda\vec{z}.P[x := Q]\vec{R}$, and we then have $M \overset{h}{\rightarrow}_\beta O \Rightarrow_\beta L$. Now use (i). □

1.5.8. Theorem. *If M has normal form N, then $M \overset{l}{\twoheadrightarrow}_\beta N$.*

Proof. Induction on the length of N. We have $M \Rightarrow_\beta \cdots \Rightarrow_\beta N$. By Lemma 1.5.7, the reduction from M to N consists of head reductions and parallel internal reductions, and the head reductions can be brought to the beginning. Thus $M \overset{h}{\twoheadrightarrow}_\beta L \overset{i}{\twoheadrightarrow}_\beta N$, where $L = \lambda\vec{z}.y\vec{P}$ and $N = \lambda\vec{z}.y\vec{P'}$, and where P_i has normal form P'_i. By the induction hypothesis, leftmost reduction of each P_i yields P'_i. Then $M \overset{l}{\twoheadrightarrow}_\beta N$. □

1.5.9. Definition. A *reduction strategy* F is a map from λ-terms to λ-terms such that $F(M) = M$ when M is in normal form, and $M \rightarrow_\beta F(M)$ otherwise. Such an F is *normalizing* if for all $M \in \mathrm{WN}_\beta$, there is an i such that $F^i(M)$ is in normal form.

1.5.10. Corollary. *Define $F_l(M) = M$ for each normal form M, and $F_l(M) = N$, where $M \overset{l}{\rightarrow}_\beta N$, otherwise. Then F_l is normalizing.*

1.5.11. Definition. A reduction sequence is called

- *quasi-leftmost* if it contains infinitely many leftmost reductions;
- *quasi-head* if it contains infinitely many head reductions.

1.5.12. Corollary. *Let M be normalizing. We then have the following.*

(i) *M has no infinite head-reduction sequence.*

(ii) *M has no quasi-head reduction sequence.*

(iii) *M has no quasi-leftmost reduction sequence.*

Proof. Part (i) follows directly from the theorem. For (ii), suppose M has a quasi-head reduction sequence. By Lemma 1.5.7(ii), we can postpone the internal reductions in the quasi-head reduction indefinitely to get an infinite head reduction, contradicting Theorem 1.5.8.

Part (iii) is by induction on the length of the normal form of M. Suppose M has a quasi-leftmost reduction sequence. By (ii) we may assume that, from some point on, the quasi-leftmost reduction contains no head reductions. One of the reductions after this point must be leftmost, so the sequence is

$$M \twoheadrightarrow_\beta L_1 \overset{i}{\rightarrow}_\beta L_2 \overset{i}{\rightarrow}_\beta L_3 \overset{i}{\rightarrow}_\beta \cdots \qquad (*)$$

where $L_1 = \lambda\vec{z}.y\vec{P}$. Infinitely many of the leftmost steps in $(*)$ must occur within the same P_i and these steps are leftmost relative to P_i, contradicting the induction hypothesis. □

1.6. Perpetual reductions and the conservation theorem

Theorem 1.5.8 provides a way to obtain the normal form of a term, when it exists. There is also a way to *avoid* the normal form, i.e. to find an infinite reduction, when one exists.

1.6.1. DEFINITION. Define $F_\infty : \Lambda \to \Lambda$ as follows. If $M \in \mathrm{NF}_\beta$ then $F_\infty(M) = M$; otherwise

$$\begin{array}{lcll} F_\infty(\lambda\vec{z}.x\vec{P}Q\vec{R}) & = & \lambda\vec{z}.x\vec{P}F_\infty(Q)\vec{R}, & \text{if } \vec{P} \in \mathrm{NF}_\beta \text{ and } Q \notin \mathrm{NF}_\beta; \\ F_\infty(\lambda\vec{z}.(\lambda x P)Q\vec{R}) & = & \lambda\vec{z}.P[x := Q]\vec{R}, & \text{if } x \in \mathrm{FV}(P) \text{ or } Q \in \mathrm{NF}_\beta; \\ F_\infty(\lambda\vec{z}.(\lambda x P)Q\vec{R}) & = & \lambda\vec{z}.(\lambda x P)F_\infty(Q)\vec{R}, & \text{if } x \notin \mathrm{FV}(P) \text{ and } Q \notin \mathrm{NF}_\beta. \end{array}$$

It is easy to see that $M \to F_\infty(M)$ when $M \notin \mathrm{NF}_\beta$.

1.6.2. LEMMA. *Assume $Q \in \mathrm{SN}_\beta$ or $x \in \mathrm{FV}(P)$. If $P[x := Q]\vec{R} \in \mathrm{SN}_\beta$, then $(\lambda x.P)Q\vec{R} \in \mathrm{SN}_\beta$.*

PROOF. Let $P[x := Q]\vec{R} \in \mathrm{SN}_\beta$. Then $P, \vec{R} \in \mathrm{SN}_\beta$. If $x \notin \mathrm{FV}(P)$, then $Q \in \mathrm{SN}_\beta$ by assumption. If $x \in \mathrm{FV}(P)$, then Q is part of $P[x := Q]\vec{R}$, so $Q \in \mathrm{SN}_\beta$. If $(\lambda x.P)Q\vec{R} \in \infty_\beta$, then any infinite reduction must have form

$$(\lambda x.P)Q\vec{R} \twoheadrightarrow_\beta (\lambda x.P')Q'\vec{R}' \to_\beta P'[x := Q']\vec{R}' \to_\beta \cdots$$

Then $P[x := Q]\vec{R} \twoheadrightarrow_\beta P'[x := Q']\vec{R}' \to_\beta \cdots$, a contradiction. □

1.6.3. THEOREM. *If $M \in \infty_\beta$ then $F_\infty(M) \in \infty_\beta$.*

PROOF. By induction on M. If $M = \lambda\vec{z}.x\vec{P}$, apply the induction hypothesis as necessary. We consider the remaining cases in more detail.
CASE 1: $M = \lambda\vec{z}.(\lambda x.P)Q\vec{R}$ where $x \in \mathrm{FV}(P)$ or $Q \in \mathrm{NF}_\beta$. Then we have $F_\infty(M) = \lambda\vec{z}.P[x := Q]\vec{R}$, and thus $F_\infty(M) \in \infty_\beta$, by Lemma 1.6.2.
CASE 2: $M = \lambda\vec{z}.(\lambda x.P)Q\vec{R}$ where $x \notin \mathrm{FV}(P)$ and $Q \in \infty_\beta$. Then we have $F_\infty(M) = \lambda\vec{z}.(\lambda x.P)F_\infty(Q)\vec{R}$. By the induction hypothesis $F_\infty(Q) \in \infty_\beta$, so $F_\infty(M) \in \infty_\beta$.
CASE 3: $M = \lambda\vec{z}.(\lambda x.P)Q\vec{R}$ where $x \notin \mathrm{FV}(P)$ and $Q \in \mathrm{SN}_\beta - \mathrm{NF}_\beta$. Then we have $F_\infty(M) = \lambda\vec{z}.(\lambda x.P)F_\infty(Q)\vec{R} \twoheadrightarrow_\beta P\vec{R}$. From Lemma 1.6.2, we obtain $P\vec{R} \in \infty_\beta$, but then also $F_\infty(M) \in \infty_\beta$. □

1.6.4. DEFINITION. A reduction strategy F is *perpetual* iff for all $M \in \infty_\beta$,

$$M \to_\beta F(M) \to_\beta F(F(M)) \to_\beta \cdots$$

is an infinite reduction sequence from M.

1.6.5. COROLLARY. *F_∞ is perpetual.*

PROOF. Immediate from the preceding theorem. □

1.6.6. DEFINITION. The set of *$\lambda\mathbf{I}$-terms* is defined as follows.

- Every variable is a $\lambda\mathbf{I}$-term.
- An application MN is a $\lambda\mathbf{I}$-term iff both M and N are $\lambda\mathbf{I}$-terms.
- An abstraction $\lambda x M$ is a $\lambda\mathbf{I}$-term iff M is a $\lambda\mathbf{I}$-term and $x \in \mathrm{FV}(M)$.

The following is known as the *conservation theorem* (for $\lambda\mathbf{I}$-terms).

1.6.7. COROLLARY.

(i) *For all $\lambda\mathbf{I}$-terms M, if $M \in \mathrm{WN}_\beta$ then $M \in \mathrm{SN}_\beta$.*

(ii) *For all $\lambda\mathbf{I}$-terms M, if $M \in \infty_\beta$ and $M \to_\beta N$ then $N \in \infty_\beta$.*

PROOF. For part (i), assume $M \in \mathrm{WN}_\beta$. Then $M \overset{l}{\twoheadrightarrow}_\beta N$ for some normal form N. Now note that for all $\lambda\mathbf{I}$-terms L not in normal form, $L \overset{l}{\to}_\beta F_\infty(L)$. Thus $N = F_\infty^k(M)$ for some k, so $M \in \mathrm{SN}_\beta$, by Corollary 1.6.5.

For part (ii), suppose $M \to_\beta N$. If $M \in \infty_\beta$, then $M \notin \mathrm{WN}_\beta$, by (i). Hence $N \notin \mathrm{WN}_\beta$, in particular $N \in \infty_\beta$. □

1.7. Expressibility and undecidability

The untyped λ-calculus is so simple that it may be surprising how powerful it is. In this section we show that λ-calculus in fact can be seen as an alternative formulation of recursion theory.

We can use λ-terms to represent various constructions, e.g. truth values:

$$\mathrm{true} = \lambda xy.x;$$
$$\mathrm{false} = \lambda xy.y;$$
$$\textbf{if } P \textbf{ then } Q \textbf{ else } R = PQR.$$

It is easy to see that

$$\begin{aligned} \textbf{if true then P else Q} &\twoheadrightarrow_\beta P; \\ \textbf{if false then P else Q} &\twoheadrightarrow_\beta Q. \end{aligned}$$

Another useful construction is the ordered pair

$$\begin{aligned} \langle M, N\rangle &= \lambda x.xMN; \\ \pi_1 &= \lambda p.p(\lambda xy.x); \\ \pi_2 &= \lambda p.p(\lambda xy.y). \end{aligned}$$

As expected we have

$$\pi_i\langle M_1, M_2\rangle \twoheadrightarrow_\beta M_i.$$

1.7.1. DEFINITION. We represent the natural numbers in the λ-calculus as *Church numerals*:

$$\mathbf{c}_n = \lambda fx.f^n(x),$$

where $f^n(x)$ abbreviates $f(f(\cdots(x)\cdots))$ with n occurrences of f. Sometimes we write $\mathbf{n}$ for $\mathbf{c}_n$, so that for instance

$$\begin{array}{rcl} \mathbf{0} & = & \lambda fx.x; \\ \mathbf{1} & = & \lambda fx.fx; \\ \mathbf{2} & = & \lambda fx.f(fx). \end{array}$$

1.7.2. DEFINITION. A partial function $f : \mathbb{N}^k \to \mathbb{N}$ is λ-*definable* iff there is an $F \in \Lambda$ such that:

- If $f(n_1, \ldots, n_k) = m$ then $F\mathbf{c}_{n_1} \ldots \mathbf{c}_{n_k} =_\beta \mathbf{c}_m$.
- If $f(n_1, \ldots, n_k)$ is undefined then $F\mathbf{c}_{n_1} \ldots \mathbf{c}_{n_k}$ has no normal form.

We say that the term F *defines* the function f.

REMARK. If F defines f, then in fact $F\mathbf{c}_{n_1} \ldots \mathbf{c}_{n_k} \twoheadrightarrow_\beta \mathbf{c}_{f(n_1,\ldots,n_k)}$.

1.7.3. EXAMPLE. The following terms define a few commonly used functions.

- Successor: succ $= \lambda nfx.f(nfx)$.
- Addition: add $= \lambda mnfx.mf(nfx)$.
- Multiplication: mult $= \lambda mnfx.m(nf)x$.
- Exponentiation: exp $= \lambda mnfx.mnfx$.
- The constant zero function: zero $= \lambda m.\mathbf{0}$.
- The i-th projection of k-arguments: $\Pi^k_i = \lambda m_1 \ldots m_k.m_i$.

We show that all partial recursive functions are λ-definable.

1.7.4. PROPOSITION. *The primitive recursive functions are λ-definable.*

PROOF. It follows from Example 1.7.3 that the initial functions are definable. It should also be obvious that the class of λ-definable total functions is closed under composition. It remains to show that λ-definability is closed under primitive recursion. Assume that f is given by

$$\begin{array}{rcl} f(0, n_1, \ldots, n_m) & = & g(n_1, \ldots, n_m); \\ f(n+1, n_1, \ldots, n_m) & = & h(f(n, n_1, \ldots, n_m), n, n_1, \ldots, n_m), \end{array}$$

where g and h are λ-definable by G and H. Define auxiliary terms

$$\begin{array}{lcl} \text{Init} & = & \langle 0, Gx_1 \dots x_m \rangle. \\ \text{Step} & = & \lambda p. \langle \text{succ}(\pi_1 p), H(\pi_2 p)(\pi_1 p) x_1 \dots x_m \rangle; \end{array}$$

The function f is then λ-definable by

$$F = \lambda x x_1 \dots x_m . \pi_2 (x \, \text{Step} \, \text{Init}).$$

This expresses the following algorithm: Generate a sequence of pairs

$$(0, a_0), (1, a_1), \dots, (n, a_n),$$

where $a_0 = g(n_1, \dots, n_m)$ and $a_{i+1} = h(a_i, i, n_1, \dots, n_m)$, so at the end of the sequence, we have $a_n = f(n, n_1, \dots, n_m)$. □

1.7.5. THEOREM. *All partial recursive functions are λ-definable.*

PROOF. Let f be a partial recursive function. By Theorem A.3.8

$$f(n_1, \dots, n_m) = \ell(\mu y[g(y, n_1, \dots, n_m) = 0]),$$

where g and ℓ are primitive recursive. We show that f is λ-definable. For this, we first define a test for zero:

$$\text{zero?} = \lambda x . x (\lambda y . \text{false}) \text{true}.$$

By Proposition 1.7.4, the functions g and ℓ are definable by some terms G and L, respectively. Let

$$W = \lambda y . \textbf{if } \text{zero?}(Gyx_1 \dots x_m) \textbf{ then } \lambda w . Ly \textbf{ else } \lambda w . w (\text{succ}\, y) w.$$

Note that $x_1, \dots, x_m$ are free in W. The following term defines f:

$$F = \lambda x_1 . \dots \lambda x_m . W \mathbf{c}_0 W.$$

Indeed, take any $n_1, \dots, n_m$ and let $\vec{c} = \mathbf{c}_{n_1} \dots \mathbf{c}_{n_m}$. Then

$$F \vec{c} \twoheadrightarrow_\beta W' \mathbf{c}_0 W',$$

where $W' = W[\vec{x} := \vec{c}]$. Suppose that $g(n, n_1, \dots, n_m) = 0$, and n is the least number with this property. Then

$$W' \mathbf{c}_0 W' \twoheadrightarrow_\beta W' \mathbf{c}_1 W' \twoheadrightarrow_\beta \cdots \twoheadrightarrow_\beta W' \mathbf{c}_n W' \twoheadrightarrow_\beta L \mathbf{c}_n \twoheadrightarrow_\beta \mathbf{c}_{\ell(n)}.$$

It remains to see what happens when the minimum is not defined. Then we have the following infinite quasi-leftmost reduction sequence

$$W' \mathbf{c}_0 W' \twoheadrightarrow_\beta W' \mathbf{c}_1 W' \twoheadrightarrow_\beta W' \mathbf{c}_2 W' \twoheadrightarrow_\beta \cdots$$

so Corollary 1.5.12 implies that $F \vec{c}$ has no normal form. □

1.7.6. REMARK. A close inspection of the proof of Theorem 1.7.5 reveals that it shows more than stated in the theorem: For every partial recursive function $f : \mathbb{N}^m \to \mathbb{N}$, there is a defining term F such that every application $F\mathbf{c}_{n_1} \dots \mathbf{c}_{n_m}$ is *uniformly normalizing*, i.e. either strongly normalizing or without normal form. The details of this claim can be found in Exercise 1.21.

1.7.7. COROLLARY. *The following problems are undecidable:*

(i) *Given $M \in \Lambda$, does M have a normal form?*

(ii) *Given $M \in \Lambda$, is M strongly normalizing?*

PROOF. For (i), suppose we have an algorithm to decide whether any term has a normal form. Take any recursively enumerable set $A \subseteq \mathbb{N}$ that is not recursive, and let f be a partial recursive function with domain A. Clearly, f is λ-definable by some term F. Now, for a given $n \in \mathbb{N}$, we can effectively decide whether $n \in A$ by checking whether the term $F\mathbf{n}$ has a normal form. Part (ii) now follows from Remark 1.7.6. □

1.8. Notes

The λ-calculus and the related systems of *combinatory logic* were introduced around 1930 by Alonzo Church [73, 74] and Haskell B. Curry [101, 102, 104], respectively. From the beginning, the calculi were parts of systems intended to be a foundation for logic. Unfortunately, Church's students Kleene and Rosser [277] discovered in 1935 that the original systems were inconsistent, and Curry [106] simplified the result, which became known as *Curry's paradox*. Consequently, the subsystem dealing with λ-terms, reduction, and conversion, i.e. what we call λ-calculus, was studied independently.

The notions of λ-binding and α-convertible terms are intuitively very clear, but we have seen in Section 1.2 that various technical difficulties must be overcome in order to handle them properly. This issue becomes especially vital when one faces the problem of a practical implementation. A classical solution [63] is to use a nameless representation of variables (so called *de Bruijn indices*). For more on this and related subjects, see e.g. [403, 406, 412].

The Church-Rosser theorem, which can be seen as a consistency result for the λ-calculus, was proved in 1936 by Church and Rosser [80]. Many proofs appeared later. Barendregt [36] cites Tait and Martin-Löf for the technique using parallel reductions; our proof is from Takahashi [481]. Proofs of the Church-Rosser theorem and an extension of Theorem 1.5.8 for $\beta\eta$-reductions can also be found there.

Church and Rosser [80] also proved the conservation theorem for $\lambda\mathbf{I}$-terms (which is sometimes called the *second Church-Rosser theorem*). Again, many different proofs have appeared. Our proof uses the effective perpetual strategy from [36], and the fact, also noted in [424], that the perpetual strategy always contracts the leftmost redex, when applied to a $\lambda\mathbf{I}$-term. More about perpetual strategies and their use in proving conservation theorems can be found in [418] and [371].

The λ-calculus turned out to be useful for formalizing the intuitive notion of effective computability. Kleene [273] showed that every partial recursive function

was λ-definable and vice versa. This led Church to conjecture that λ-definability is an appropriate formalization of the intuitive notion of effective computability [76], which became known as *Church's thesis*.

The problems of deciding membership of WN_β and SN_β can be seen as variants of the halting problem. Church [75, 76] inferred from the former his celebrated theorem stating that first-order arithmetic is undecidable, as well as the undecidability of the *Entscheidungsproblem* (the "decision problem" for first-order logic), results that were "in the air" in this period [167].

Curry and his co-workers continued the work on *illative* combinatory logic [110, 111], i.e. logical systems including formulas as well as combinators and types. The calculus of combinators was then studied as an independent subject, and a wealth of results was obtained. For instance, the theorem about leftmost reductions is from [110]. Like many other classical results in λ-calculus it has been proved in many different ways ever since; our proof is taken from [481].

With the invention of computing machinery came also programming languages. Already in the 1960's λ-calculus was recognized as a useful tool in the design, implementation, and theory of programming languages [302]. In particular, type-free λ-calculus constitutes a model of untyped *functional* programming languages, e.g. Scheme [3] and Lisp [204], while typed calculi correspond to functional languages like Haskell [489] and ML [398].

The classic texts on type-free λ-calculus are [244] and [36]. First-hand historical information may be obtained from Curry and Feys' book [110], which contains a wealth of historical information, and from Rosser and Kleene's eyewitness statements [276, 432]. Other interesting papers are [30, 241].

1.9. Exercises

1.1. For a pre-term M, the directed labeled graph $G(M)$ is defined by induction.

- If $M = x$ then $G(M)$ has a single root node labeled x and no edges.
- If $M = PQ$ then $G(M)$ is obtained from the union of $G(P)$ and $G(Q)$ by adding a new initial (root) node labeled @. This node has two outcoming edges: to the root nodes of $G(P)$ and $G(Q)$.
- If $M = \lambda x P$ then $G(M)$ is obtained from $G(P)$ by
  - Adding a new root node labeled λx, and an edge from there to the root node of $G(P)$;
  - Adding edges to the new root from all final nodes labeled x.

For a given graph G, let $\mathit{erase}(G)$ be a graph obtained from G by deleting all labels from the variable nodes that are not final and renaming every label λx, for some variable x, to λ. Prove that the conditions $\mathit{erase}(G(M)) = \mathit{erase}(G(N))$ and $M =_\alpha N$ are equivalent for all $M, N \in \Lambda^-$.

1.2. Modify Definition 1.2.4 so that the operation $M[x := N]$ is defined for all M, N and x. Then prove that $M[x := N] =_\alpha M'[x := N']$ holds for all $M =_\alpha M'$ and $N =_\alpha N'$ (cf. Lemma 1.2.11).

1.3. Let $\vec{x} \in \Upsilon$ and $\vec{N} \in \Lambda$ be fixed. Show that Definition 1.2.21 determines a total function from Λ to Λ. *Hint:* Rewrite the definition as a relation $r \subseteq \Lambda \times \Lambda$ and

show that for every $M \in \Lambda$ there is exactly one $L \in \Lambda$ such that $r(M, L)$. It may be beneficial to show uniqueness and existence separately.

1.4. Prove that if $(\lambda x P)Q = (\lambda y M)N$ then $P[x := Q] = M[y := N]$. In other words, the contraction of a given redex yields a unique result.

1.5. Show that M is in normal form if and only if M is either a variable or an abstraction $\lambda x M'$, where M' is normal, or $M = M'[x := yN]$, for some normal forms M' and N, and some x occurring free exactly once in M'.

1.6. Show that every term is strongly normalizing with respect to eta-reductions.

1.7. Which of the following are true?

(i) $\lambda x.Mx =_\beta M$, for any abstraction M with $x \notin \mathrm{FV}(M)$.

(ii) $\lambda x \mathbf{\Omega} =_\beta \mathbf{\Omega}$.

(iii) $(\lambda x.xx)(\lambda xy.y(xx)) =_\beta (\lambda x.x\mathbf{I}x)(\lambda zxy.y(xzx))$.

1.8. Prove the *weak Church-Rosser theorem:* For all $M_1, M_2, M_3 \in \Lambda$, if $M_1 \to_\beta M_2$ and $M_1 \to_\beta M_3$, then there is $M_4 \in \Lambda$ with $M_2 \twoheadrightarrow_\beta M_4$ and $M_3 \twoheadrightarrow_\beta M_4$. Do not use the Church-Rosser theorem.

1.9. Which of the following are true?

(i) $\mathbf{II}(\mathbf{II}) \Rightarrow_\beta \mathbf{II}$.

(ii) $\mathbf{II}(\mathbf{II}) \Rightarrow_\beta \mathbf{I}$.

(iii) $\mathbf{IIII} \Rightarrow_\beta \mathbf{III}$.

(iv) $\mathbf{IIII} \Rightarrow_\beta \mathbf{I}$.

1.10. Find terms M, N such that $M \Rightarrow_\beta N$ and $M \overset{i}{\twoheadrightarrow}_\beta N$, but not $M \overset{i}{\Rightarrow}_\beta N$.

1.11. Find terms M, N such that $M \overset{i}{\to}_\beta N$ and $M \overset{h}{\to}_\beta N$ both hold.

1.12. Let $M \to_\beta F_\infty(M) \to_\beta F_\infty(F_\infty(M)) \to_\beta \cdots \to_\beta F_\infty^n(M)$, where $F_\infty^n(M) \in \mathrm{NF}$. Show that there is no reduction from M with more than n reduction steps. *Hint:* Generalize Lemma 1.6.2 to show that

$$l_\beta((\lambda x.P)Q\vec{R}) = l_\beta(P[x := Q]\vec{R}) + \epsilon(P) \cdot l_\beta(Q) + 1,$$

where $l_\beta(M)$ denotes the length of the longest reduction sequence from M and $\epsilon(P)$ is 1 if $x \notin \mathrm{FV}(P)$ and 0 else. Show that $l_\beta(M) = 1 + l_\beta(F_\infty(M))$, if $M \notin \mathrm{NF}$.

1.13. Show that there is no total computable $l : \Lambda \to \mathbb{N}$ such that, for all $M \in \mathrm{SN}_\beta$,

$$l(M) \geq l_\beta(M),$$

where $l_\beta(M)$ is as in the Exercise 1.12. *Hint:* That would imply decidability of SN.

1.14. Consider the *fixed point combinator:*

$$\mathbf{Y} = \lambda f.(\lambda x.f(xx))(\lambda x.f(xx)).$$

Show that $F(\mathbf{Y}F) =_\beta \mathbf{Y}F$ holds for all F. (Thus in the untyped lambda-calculus every fixpoint equation $X = FX$ has a solution.)

1.15. Let M be any other fixed point combinator, i.e. assume that $F(MF) =_\beta MF$ holds for all F. Show that M has no normal form.

1.16. Define the predecessor function in the untyped lambda-calculus.

1.17. (B. Maurey, J.-L. Krivine.) Let $C = \lambda xy.(xF(\lambda z\mathbf{1}))(yF(\lambda z\mathbf{0}))$, where $F = \lambda fg.gf$. Show that C defines the function

$$c(m,n) = \begin{cases} 1, & \text{if } m \leq n; \\ 0, & \text{otherwise.} \end{cases}$$

How many steps are needed to reduce $C\mathbf{c}_m\mathbf{c}_n$ to normal form? Will the same hold if we define c using Proposition 1.7.4?

1.18. (From [78].) Find $\lambda\mathbf{I}$-terms P_1, P_2 (projections) such that $P_1\langle\mathbf{c}_m, \mathbf{c}_n\rangle =_\beta \mathbf{c}_m$ and $P_2\langle\mathbf{c}_m, \mathbf{c}_n\rangle =_\beta \mathbf{c}_n$, for all m, n.

1.19. Show that the following functions are λ-definable:

- For each n, the function $f(i, m_1, \ldots, m_n) = m_i$.
- Integer division, i.e. a function f such that $f(mn, m) = n$, for all $m, n \neq 0$.
- Integer square root, i.e. a function f such that $f(n^2) = n$, for all n.

1.20. Assume that $M \twoheadrightarrow_\beta z$. Show that if $M[z := \lambda xN] \twoheadrightarrow_\beta \lambda uQ$, where N is normal, then $\lambda uQ = \lambda xN$. Will this remain true if we replace $\twoheadrightarrow_\beta$ by $\twoheadrightarrow_{\beta\eta}$?

1.21. For $n \in \mathbb{N}$ put $d_n = \mathrm{succ}^n(\mathbf{c}_0)$. Assume the following:[4]

> Every primitive recursive function is definable by a term F such that all applications $Fd_{n_1} \ldots d_{n_m}(\lambda v.\,\mathrm{false})\mathrm{true}$ are strongly normalizing.

Prove the claim in Remark 1.7.6. *Hint:* Use Exercise 1.20.

1.22. Prove that β-equality is undecidable. *Hint:* See Corollary 1.7.7(i).

[4] The proof of this is beyond our reach at the moment. We return to it in Exercise 11.26.