# Chapter 1

# The systems $\lambda_{\rightarrow}$

## 1.1. The $\lambda_{\rightarrow}$ systems *à la* Curry

Types in this part are syntactic objects built from atomic types using the operator $\rightarrow$. In order to classify untyped lambda terms, such types will be assigned to a subset of these terms. The main idea is that if $M$ gets type $A{\rightarrow}B$ and $N$ gets type $A$, then the application $MN$ is 'legal' (as $M$ is considered as a function from terms of type $A$ to those of type $B$) and gets type $B$. In this way types help determining what terms fit together.

1.1.1. Definition. (i) Let $\mathbb{A}$ be a non-empty set of 'atomic types'. The set of *simple types over* $\mathbb{A}$, notation $\mathbb{T} = \mathbb{T}_{\mathbb{A}}$, is inductively defined as follows.

$$\begin{array}{rcll} \alpha \in \mathbb{A} & \Rightarrow & \alpha \in \mathbb{T} & \text{type atoms;} \\ A, B \in \mathbb{T} & \Rightarrow & (A{\rightarrow}B) \in \mathbb{T} & \text{function space types.} \end{array}$$

Such definitions will be used often and for these it is convenient to use the so called *abstract syntax*, see Waite and Goos [1984]. As an example we give the abstract syntax for $\mathbb{T} = \mathbb{T}_{\mathbb{A}}$.

$$\boxed{\mathbb{T} \quad = \quad \mathbb{A} \,|\, \mathbb{T} \rightarrow \mathbb{T}}$$

Figure 1.1: Simple types

(ii) Let $\mathbb{A}_o = \{o\}$. Then we write $\mathbb{T}_o = \mathbb{T}_{\mathbb{A}_o}$.
(iii) Let $\mathbb{A}_\infty = \{\alpha_0, \alpha_1, \alpha_2, \ldots\}$. Then we write $\mathbb{T}_\infty = \mathbb{T}_{\mathbb{A}_\infty}$

We consider that $o = \alpha_0$, hence $\mathbb{T}_o \subseteq \mathbb{T}_\infty$. If we write simply $\mathbb{T}$, then this refers to $\mathbb{T}_{\mathbb{A}}$ for an unspecified $\mathbb{A}$.

1.1.2. Notation. (i) If $A_1, \ldots, A_n \in \mathbb{T}$, then

$$A_1{\rightarrow} \ldots {\rightarrow}A_n \equiv (A_1{\rightarrow}(A_2{\rightarrow} \ldots {\rightarrow}(A_{n-1}{\rightarrow}A_n)..)).$$

That is, we use association to the right (here $\equiv$ denotes syntactic equality).

(ii) $\alpha, \boldsymbol{\beta}, \gamma, \ldots$ denote arbitrary elements of $\mathbb{A}$.
(iii) $A, B, C, \ldots$ denote arbitrary elements of $\mathbb{T}$.

Remember the untyped lambda calculus denoted by $\lambda$, see e.g. B[1984][1]. It consists of a set of terms $\Lambda$ defined by the following abstract syntax.

$$\boxed{\begin{array}{lcl} \mathsf{V} & = & x \,|\, \mathsf{V}' \\ \Lambda & = & \mathsf{V} \,|\, \lambda \mathsf{V}\, \Lambda \,|\, \Lambda\, \Lambda \end{array}}$$

Figure 1.2: Untyped lambda terms

This makes $\mathsf{V} = \{x, x', x'', \ldots\} = \{x_0, x_1, x_2, \ldots\}$.

1.1.3. Notation. (i) $x, y, z, \ldots$ denote arbitrary term variables.
(ii) $M, N, L, \ldots$ denote arbitrary lambda terms.
(iii) $MN_1 \ldots N_k \equiv (..(MN_1) \ldots N_k)$.
(iv) $\lambda x_1 \ldots x_n.M \equiv (\lambda x_1(..(\lambda x_n(M))..))$.

1.1.4. Definition. On $\Lambda$ the following equational theory $\boldsymbol{\lambda\beta\eta}$ is defined by the usual equality axiom and rules (reflexivity, symmetry, transitivity, congruence), inluding congruence with respect to abstraction:

$$M = N \;\Rightarrow\; \lambda x.M = \lambda x.N,$$

and the following special axiom(schemes)

$$\boxed{\begin{array}{rcllr} (\lambda x.M)N & = & M[x := N] & & (\boldsymbol{\beta}\text{-rule}) \\ \lambda x.Mx & = & M, & \text{if } x \notin \mathrm{FV}(M) & (\eta\text{-rule}) \end{array}}$$

Figure 1.3: The theory $\boldsymbol{\lambda\beta\eta}$

As is know this theory can be analyzed by a notion of reduction.

1.1.5. Definition. On $\Lambda$ we define the following notions of reduction

$$\boxed{\begin{array}{rcllr} (\lambda x.M)N & \to & M[x := N] & & (\boldsymbol{\beta}) \\ \lambda x.Mx & \to & M, & \text{if } x \notin \mathrm{FV}(M) & (\boldsymbol{\eta}) \end{array}}$$

Figure 1.4: $\boldsymbol{\beta\eta}$-contraction rules

As usual, see B[1984], these notions of reduction generate the corresponding reduction relations $\to_{\boldsymbol{\beta}}, \twoheadrightarrow_{\boldsymbol{\beta}}, \to_{\boldsymbol{\eta}}, \twoheadrightarrow_{\boldsymbol{\eta}}, \to_{\boldsymbol{\beta\eta}}$ and $\twoheadrightarrow_{\boldsymbol{\beta\eta}}$. Also there are the corresponding conversion relations $=_{\boldsymbol{\beta}}, =_{\boldsymbol{\eta}}$ and $=_{\boldsymbol{\beta\eta}}$. Terms in $\Lambda$ will often be considered modulo $=_{\boldsymbol{\beta}}$ or $=_{\boldsymbol{\beta\eta}}$. If we write $M = N$, then we mean $M =_{\boldsymbol{\beta\eta}} N$ by default. (In B[1984] the default was $=_{\boldsymbol{\beta}}$.)

1.1.6. Proposition. *For all $M, N \in \Lambda$ one has*

$$\vdash_{\boldsymbol{\lambda\beta\eta}} M = N \iff M =_{\boldsymbol{\beta\eta}} N.$$

[1]This is an abbreviation fo the reference Barendregt [1984].

PROOF. See B[1984], Proposition 3.3.2. ∎

One reason why the analysis in terms of the notion of reduction $\boldsymbol{\beta\eta}$ is useful is that the following holds.

1.1.7. THEOREM (Church-Rosser Theorem for $\lambda\boldsymbol{\beta\eta}$). *For the notions of reduction $\twoheadrightarrow_{\boldsymbol{\beta}}$ and $\twoheadrightarrow_{\boldsymbol{\beta\eta}}$ one has the following.*

(i) *Let $M, N \in \Lambda$. Then*

$$M =_{\boldsymbol{\beta}(\eta)} N \;\Rightarrow\; \exists Z \in \Lambda . M \twoheadrightarrow_{\boldsymbol{\beta}(\eta)} Z \;\&\; N \twoheadrightarrow_{\boldsymbol{\beta}(\eta)} Z.$$

(ii) *Let $M, N_1, N_2 \in \Lambda$. Then*

$$M \twoheadrightarrow_{\boldsymbol{\beta}(\eta)} N_1 \;\&\; M \twoheadrightarrow_{\boldsymbol{\beta}(\eta)} N_2 \;\Rightarrow\; \exists Z \in \Lambda . N_1 \twoheadrightarrow_{\boldsymbol{\beta}(\eta)} Z \;\&\; N_2 \twoheadrightarrow_{\boldsymbol{\beta}(\eta)} Z.$$

PROOF. (i) See Theorems 3.2.8 and 3.3.9 in B[1984].
(ii) By (i). ∎

1.1.8. DEFINITION ($\lambda_\to^{\mathrm{Cu}}$). (i) A *(type assignment) statement* is of the form

$$M : A,$$

with $M \in \Lambda$ and $A \in \mathbb{T}$. This statement is pronounced as '$M$ in $A$'. The type $A$ is the *predicate* and the term $M$ is the *subject* of the statement.

(ii) A *declaration* is a statement with as subject a term variable.

(iii) A *basis* is a set of declarations with distinct variables as subjects.

(iv) A statement $M{:}A$ is *derivable from* a basis $\Gamma$, notation

$$\Gamma \vdash_{\lambda_\to}^{\mathrm{Cu}} M{:}A$$

(or $\Gamma \vdash_{\lambda_\to} M : A$, $\Gamma \vdash^{\mathrm{Cu}} M : A$ or even $\Gamma \vdash M{:}A$ if there is little danger of confusion) if $\Gamma \vdash M{:}A$ can be produced by the following rules.

$$\begin{aligned}
(x{:}A) \in \Gamma \quad &\Rightarrow \quad \Gamma \vdash x : A;\\
\Gamma \vdash M : (A \to B),\;\; \Gamma \vdash N : A \quad &\Rightarrow \quad \Gamma \vdash (MN) : B;\\
\Gamma,\; x{:}A \vdash M : B \quad &\Rightarrow \quad \Gamma \vdash (\lambda x.M) : (A \to B).
\end{aligned}$$

These rules are usually written as follows.

| | | |
|---|---|---|
| (axiom) | $\Gamma \vdash x : A,$ | if $(x{:}A) \in \Gamma;$ |
| ($\to$-elimination) | $\dfrac{\Gamma \vdash M : (A \to B) \quad \Gamma \vdash N : A}{\Gamma \vdash (MN) : B};$ | |
| ($\to$-introduction) | $\dfrac{\Gamma, x{:}A \vdash M : B}{\Gamma \vdash (\lambda x.M) : (A \to B)}.$ | |

Figure 1.5: The system $\lambda_\to^{\mathrm{Cu}}$ of type assignment *á la* Curry

This is the modification to the lambda calculus of the system in Curry [1934], as developed in Curry et al. [1958].

NOTATION. Another way of writing these rules is sometimes found in the literature.

| | |
|---|---|
| Introduction rule | $\dfrac{\begin{array}{c}\cancel{x : A}\\ \vdots \\ M : B\end{array}}{\lambda x.M : (A{\to}B)}$ |
| Elimination rule | $\dfrac{M : (A \to B) \quad N : A}{MN : B}$ |

$\lambda_\to^{\mathrm{Cu}}$ alternative version

In this version the axiom is considered as implicit and is not notated. The notation

$$\begin{array}{c} x : A \\ \vdots \\ M : B \end{array}$$

denotes that $M : B$ can be derived from $x{:}A$. Striking through $x{:}A$ means that for the conclusion $\lambda x.M : A{\to}B$ the assumption $x{:}A$ is no longer needed; it is *discharged*.

1.1.9. DEFINITION. Let $\Gamma = \{x_1{:}A_1, \dots, x_n{:}A_n\}$. Then

(i) $\mathrm{dom}(\Gamma) = \{x_1, \dots, x_n\}$.

(ii) $x_1{:}A_1, \dots, x_n{:}A_n \vdash M : A$ denotes $\Gamma \vdash M : A$.

(iii) In particular $\vdash M : A$ stands for $\emptyset \vdash M : A$.

(iv) $x_1, \dots, x_n{:}A \vdash M : B$ stands for $x_1{:}A, \dots, x_n{:}A \vdash M : B$.

1.1.10. EXAMPLE. (i) $\vdash (\lambda xy.x) : (A \to B \to A)$ for all $A, B \in \mathbb{T}$.
We will use the notation of version 1 of $\lambda_\to$ for a derivation of this statement.

$$\dfrac{\dfrac{x{:}A, y{:}B \vdash x : A}{x{:}A \vdash (\lambda y.x) : B{\to}A}}{\vdash (\lambda x \lambda y.x) : A{\to}B{\to}A}$$

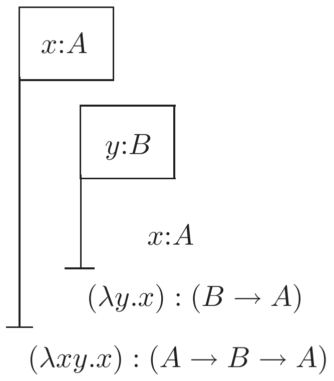
Note that $\lambda xy.x \equiv \lambda x\lambda y.x$ by definition.

(ii) A *natural deduction* derivation (for the alternative version of the system) of the same type assignment is the following.

$$\dfrac{\dfrac{\dfrac{\cancel{x{:}A}\ 2 \qquad \cancel{y{:}B}\ 1}{x{:}A}}{(\lambda y.x)\ :\ (B \to A)}\ 1}{(\lambda xy.x)\ :\ (A \to B \to A)}\ 2$$

The indices 1 and 2 are bookkeeping devices that indicate at which application of a rule a particular assumption is being discharged.

(iii) A more explicit way of dealing with cancellations of statements is the 'flag-notation' used by Fitch (1952) and in the languages AUTOMATH of de Bruijn (1980). In this notation the above derivation becomes as follows.


$x{:}A$
$y{:}B$
$x{:}A$
$(\lambda y.x) : (B \to A)$
$(\lambda xy.x) : (A \to B \to A)$

As one sees, the bookkeeping of cancellations is very explicit; on the other hand it is less obvious how a statement is derived from previous statements in case applications are used.

(iv) Similarly one can show for all $A \in \mathbb{T}$

$$\vdash (\lambda x.x) : (A \to A).$$

(v) An example with a non-empty basis is $y{:}A \vdash (\lambda x.x)y : A$.

In the rest of this chapter and in fact in the rest of this book we usually will introduce systems of typed lambda calculi in the style of the first variant of $\lambda_\to$.

1.1.11. Definition. Let $\Gamma$ be a basis and $A \in \mathbb{T}$. Then

(i) $\Lambda^\Gamma_\to(A) = \{M \in \Lambda \mid \Gamma \vdash_{\lambda_\to} M : A\}$.

(ii) $\Lambda^\Gamma_\to = \bigcup_{A \in \mathbb{T}} \Lambda^\Gamma_\to(A)$.

(iii) $\Lambda_\to(A) = \Lambda^\emptyset_\to(A)$.

(iv) $\Lambda_\to = \Lambda^\emptyset_\to$

1.1.12. Definition. Let $\Gamma$ be a basis, $A \in \mathbb{T}$ and $M \in \Lambda$. Then

(i) If $M \in \Lambda_\to(A)$, then we say that

*$M$ has type $A$* or *$A$ is inhabited by $M$.*

(ii) If $M \in \Lambda_\to$, then $M$ is called *typable.*

(iii) If $M \in \Lambda^\Gamma_\to(A)$, then $M$ *has type $A$ relative to* $\Gamma$.

(iv) If $M \in \Lambda^\Gamma_\to$, then $M$ is called *typeable relative to* $\Gamma$.

(v) If $\Lambda^{(\Gamma)}_\to(A) \neq \emptyset$, then $A$ is *inhabited (relative to* $\Gamma$*).*

1.1.13. Example. We have

$$\begin{array}{rcl} \mathsf{K} & \in & \Lambda^{\emptyset}_\to(A \to B \to A); \\ \mathsf{K}x & \in & \Lambda^{\{x:A\}}_\to(B \to A). \end{array}$$

1.1.14. Definition. Let $A \in \mathbb{T}(\lambda_\to)$.

(i) The *depth* of $A$, notation $\mathrm{dpt}(A)$, is defined as follows.

$$\begin{array}{rcl} \mathrm{dpt}(\alpha) & = & 0 \\ \mathrm{dpt}(A \to B) & = & \max\{\mathrm{dpt}(A), \mathrm{dpt}(B)\} + 1 \end{array}$$

(ii) The *rank* of $A$, notation $\mathrm{rk}(A)$, is defined as follows.

$$\begin{array}{rcl} \mathrm{rk}(\alpha) & = & 0 \\ \mathrm{rk}(A \to B) & = & \max\{\mathrm{rk}(A) + 1, \mathrm{rk}(B)\} \end{array}$$

(iii) The *order* of $A$, notation $\mathrm{ord}(A)$, is defined as follows.

$$\begin{array}{rcl} \mathrm{ord}(\alpha) & = & 1 \\ \mathrm{ord}(A \to B) & = & \max\{\mathrm{ord}(A) + 1, \mathrm{ord}(B)\} \end{array}$$

(iv) The depth (rank or order) of a basis $\Gamma$ is

$$\max_i\{\mathrm{dpt}(A_i) \mid (x_i{:}A_i) \in \Gamma\},$$

(similarly for the rank and order, respectively). Note that $\mathrm{ord}(A) = \mathrm{rk}(A) + 1$.

1.1.15. Definition. For $A \in \mathbb{T}$ we define $A^k \to B$ by recursion on $k$:

$$\begin{array}{rcl} A^0 \to B & = & B; \\ A^{k+1} \to B & = & A \to A^k \to B. \end{array}$$

Note that $\mathrm{rk}(A^k \to B) = \mathrm{rk}(A \to B)$, for all $k > 0$.

Several properties can be proved by induction on the depth of a type. This holds for example for Lemma 1.1.18(i).

The asymmetry in the definition of rank is intended because e.g. a type like $(o{\rightarrow}o){\rightarrow}o$ is more complex than $o{\rightarrow}o{\rightarrow}o$, as can be seen by looking to the inhabitants of these types: functionals with functions as arguments versus binary function. Sometimes one uses instead of 'rank' the name *type level*. This notion will turn out to be used most of the times.

In logically motivated papers one finds the notion $\mathrm{ord}(A)$. The reason is that in first-order logic one deals with domains and their elements. In second order logic one deals with functions between first-order objects. In this terminology 0-th order logic can be identified with propositional logic.

### The minimal and maximal systems $\lambda^o_{\rightarrow}$ and $\lambda^\infty_{\rightarrow}$

The collection $\mathbb{A}$ of type variables serves as set of base types from which other types are constructed. We have $\mathbb{T}_o = \{o\}$ with just one type atom and $\mathbb{T}_\infty = \{\alpha_0, \alpha_1, \alpha_2, \ldots\}$ with infinitely many of them. These two sets of atoms and their resulting type systems play a major role in this Part I of the book.

1.1.16. Definition. We define the following systems of type assignment.

(i) $\lambda^o_{\rightarrow} = \lambda^{\mathbb{T}_o}_{\rightarrow}$. This system is also called $\lambda^\tau$ in the literature.

(ii) $\lambda^\infty_{\rightarrow} = \lambda^{\mathbb{T}_\infty}_{\rightarrow}$.

If it becomes necessary to distinguish the set of atomic types, will use notations like $\Lambda_o(A) = \Lambda_{\mathbb{T}_o}(A)$ and $\Lambda_\infty(A) = \Lambda_{\mathbb{T}_\infty}(A)$.

Many of the interesting features of the 'larger' $\lambda_{\rightarrow}$ are already present in the minimal version $\lambda^o_{\rightarrow}$. The complexity of $\lambda_{\rightarrow}$ is already present in $\lambda^o_{\rightarrow}$.

1.1.17. Definition. (i) The following types of $\mathbb{T}_o \subseteq \mathbb{T}_{\mathbb{A}}$ are often used.

$$0 = o,\ 1 = 0{\rightarrow}0,\ 2 = (0{\rightarrow}0){\rightarrow}0,\ \ldots.$$

In general

$$0 = o \text{ and } k+1 = k{\rightarrow}0.$$

Note that $\mathrm{rk}(n) = n$.

(ii) Define $n_k$ by cases on $n$.

$$\begin{array}{rcl} o_k & = & o; \\ (n+1)_k & = & n^k{\rightarrow}o. \end{array}$$

For example

$$\begin{array}{rcl} 1_2 & = & o{\rightarrow}o{\rightarrow}o; \\ 2_3 & = & 1{\rightarrow}1{\rightarrow}1{\rightarrow}o. \end{array}$$

Notice that $\mathrm{rk}(n_k) = \mathrm{rk}(n)$, for $k > 0$.

1.1.18. LEMMA. (i) *Every type $A$ of $\lambda_{\to}^{\infty}$ is of the form*

$$A = A_1 \to A_2 \to \ldots \to A_n \to \alpha.$$

(ii) *Every type $A$ of $\lambda_{\to}^{o}$ is of the form*

$$A = A_1 \to A_2 \to \ldots \to A_n \to o.$$

(iii) $\mathrm{rk}(A_1 \to A_2 \to \ldots \to A_n \to \alpha) = \max\{\mathrm{rk}(A_i) + 1 \mid 1 \leq i \leq n\}$.

PROOF. (i) By induction on the structure (depth) of $A$. If $A = \alpha$, then this holds for $n = 0$. If $A = B \to C$, then by the induction hypothesis one has
$C = C_1 \to \ldots \to C_n \to \gamma$. Hence $A = B \to C_1 \to \ldots \to C_n \to \gamma$.
(ii) By (i).
(iii) By induction on $n$. ■

1.1.19. NOTATION. Let $A \in \mathbb{T}_{\mathbb{A}}$ and suppose $A = A_1 \to A_2 \to \ldots \to A_n \to \alpha$. Then the $A_i$ are called the *components* of $A$. We write

$$\begin{array}{rcll} \mathtt{arity}(A) & = & n, & \\ A(i) & = & A_i, & \text{for } 1 \leq i \leq n; \\ \mathtt{target}(A) & = & \alpha. & \end{array}$$

Iterated components are denoted as follows

$$A(i,j) = A(i)(j).$$

## Different versions of $\lambda_{\to}^{\mathbb{A}}$

The system $\lambda_{\to}^{\mathbb{A}}$ that was introduced in Definition 1.1.8 assigns types to untyped lambda terms. These system will be referred to as the Curry system and be denoted by $\lambda_{\to\mathrm{Cu}}^{\mathbb{A}}$ or $\lambda_{\to}^{\mathrm{Cu}}$, as the set $\mathbb{A}$ often does not need to be specified. There will be introduced two variants of $\lambda_{\to}^{\mathbb{A}}$.

The first variant of $\lambda_{\to}^{\mathrm{Cu}}$ is the *Church* version of $\lambda_{\to}^{\mathbb{A}}$, denoted by $\lambda_{\to\mathrm{Ch}}^{\mathbb{A}}$ or $\lambda_{\to}^{\mathrm{Ch}}$. In this theory the types are assigned to embellished terms in which the variables (free and bound) come with types attached. For example the Curry style type assignments

$$\begin{array}{rll} & \vdash_{\lambda_{\to}}^{\mathrm{Cu}} (\lambda x.x) : A \to A & (1_{\mathrm{Cu}}) \\ y{:}A & \vdash_{\lambda_{\to}}^{\mathrm{Cu}} (\lambda x.xy) : (A \to B) \to A \to B & (2_{\mathrm{Cu}}) \end{array}$$

now becoming

$$\begin{array}{ll} (\lambda x^A.x^A) \in \Lambda_{\to}^{\mathrm{Ch}}(A \to A) & (1_{\mathrm{Ch}}) \\ (\lambda x^{A \to B}.x^{A \to B} y^A) : \Lambda_{\to}^{\mathrm{Ch}}((A \to B) \to A \to B) & (2_{\mathrm{Ch}}) \end{array}$$

The second variant of $\lambda_{\to}^{\mathrm{Cu}}$ is the *de Bruijn* version of $\lambda_{\to}^{\mathbb{A}}$, denoted by $\lambda_{\to\mathrm{dB}}^{\mathbb{A}}$ or $\lambda_{\to}^{\mathrm{dB}}$. Now only bound variables get ornamented with types, but only at the binding stage. The examples (1), (2) now become

$$\begin{array}{rlll} & \vdash_{\lambda_\to}^{\mathrm{dB}} & (\lambda x : A.x) : A{\to}A & (1_{\mathrm{dB}}) \\ y{:}A & \vdash_{\lambda_\to}^{\mathrm{dB}} & (\lambda x : (A{\to}B).xy) : (A{\to}B){\to}A{\to}B & (2_{\mathrm{dB}}) \end{array}$$

The reasons to have these variants will be explained later in Section 1.4. In the meantime we will work intuitively.

1.1.20. Notation. Terms like $(\lambda fx.f(fx)) \in \Lambda^{\emptyset}(1{\to}o{\to}o)$ will often be written

$$\lambda f^1 x^0.f(fx)$$

to indicate the types of the bound variables. We will come back to this notational issue in section 1.4.

## 1.2. Normal inhabitants

In this section we will give an algorithm that enumerates the set of closed terms in normal form of a given type $A \in \mathbb{T}$. Since we will prove in the next chapter that all typable terms do have a nf and that reduction preserves typing, we thus have an enumeration of essentially all closed terms of that given type. We do need to distinguish various kinds of nf's.

1.2.1. Definition. Let $A = A_1 {\to} \dots A_n {\to} \alpha$ and suppose $\Gamma \vdash M : A$.

(i) Then $M$ is in long-nf, notation lnf, if $M \equiv \lambda x_1^{A_1} \dots x_n^{A_n}.xM_1 \dots M_n$ and each $M_i$ is in lnf. By induction on the depth of the type of the closure of $M$ one sees that this definition is well-founded.

(ii) $M$ has a lnf if $M =_{\beta\eta} N$ and $N$ is a lnf.

In Exercise 1.5.16 it is proved that if $M$ has a $\boldsymbol{\beta}$-nf, which according to Theorem 2.2.4 is always the case, then it also has a unique lnf and will be its unique $\boldsymbol{\beta\eta}^{-1}$ nf. Here $\eta^{-1}$ is the notion of reduction that is the converse of $\eta$.

1.2.2. Examples. (i) Note that $\lambda f^1.f =_{\beta\eta} \lambda f^1 \lambda x^o.fx$ and that $\lambda f^1.f$ is a $\boldsymbol{\beta\eta}$-nf but not a lnf.

(ii) $\lambda f^1 \lambda x^o.fx$ is a lnf, but not a $\boldsymbol{\beta\eta}$-nf.

(iii) $\lambda x{:}o.x$ is both in $\boldsymbol{\beta\eta}$-nf and lnf.

(iv) The $\boldsymbol{\beta}$-nf $\lambda F{:}2_2 \lambda f{:}1.Ff(\lambda x{:}o.fx)$ is neither in $\boldsymbol{\beta\eta}$-nf nor lnf.

(v) A variable of atomic type $\alpha$ is a lnf, but of type $A{\to}B$ not.

(vi) A variable $f : 1{\to}1$ has as lnf $\lambda g^1 \lambda x^o.f(\lambda y^o.gy)x$.

1.2.3. Proposition. *Every $\boldsymbol{\beta}$-nf $M$ has a lnf $M^\ell$ such that $M^\ell \twoheadrightarrow_{\boldsymbol{\eta}} M$.*