PART I

SIMPLE TYPES $\boldsymbol{\lambda}^{\mathbb{A}}_{\rightarrow}$

The systems of *simple types* considered in Part I are built up from atomic types $\mathbb{A}$ using, as the only operator, the constructor $\rightarrow$ for forming function spaces. For example, from the atoms $\mathbb{A} = \{\alpha, \beta\}$ one can form types $\alpha{\rightarrow}\beta$, $(\alpha{\rightarrow}\beta){\rightarrow}\alpha$, $\alpha{\rightarrow}(\alpha{\rightarrow}\beta)$ and so on. Two choices of the set of atoms that will be made most often are $\mathbb{A} = \{\alpha_0, \alpha_1, \alpha_2, \dots\}$, an infinite set of type variables giving $\boldsymbol{\lambda}^{\infty}_{\rightarrow}$, and $\mathbb{A} = \{0\}$, consisting of only one atomic type giving $\boldsymbol{\lambda}^{0}_{\rightarrow}$. Particular atomic types that occur in applications are e.g. Bool, Nat, Real. Even for these simple type systems, the ordering effect is quite powerful.

Requiring terms to have simple types implies that they are strongly normalizing. For an untyped lambda term one can find the collection of its possible types. Similarly, given a simple type, one can find the collection of its possible inhabitants (in normal form). Equality of terms of a certain type can be reduced to equality of terms in a fixed type. Insights coming from this reducibility provide five canonical term models of $\boldsymbol{\lambda}^{0}_{\rightarrow}$. See pages 2 and 3 for types and terms involved in this analysis.

The problem of unification

$$\exists X{:}A.MX =_{\boldsymbol{\beta\eta}} NX$$

is for complex enough A undecidable. That of pattern matching

$$\exists X{:}A.MX =_{\boldsymbol{\beta\eta}} N$$

will be shown to be decidable for A up to 'rank 3'. The recent proof by Stirling of general decidability of matching is not included. The terms of finite type are extended by δ-functions, functionals for primitive recursion (Gödel) and bar recursion (Spector). Applications of the theory in computing, proof-checking and semantics of natural languages will be presented.

Other expositions of the simply typed lambda calculus are Church (1941), Lambek and Scott (1981), Girard et al. (1989), Hindley (1997), and Nerode et al. (In preparation). Part of the history of the topic, including the untyped lambda calculus, can be found in Crossley (1975), Rosser (1984), Kamareddine et al. (2004) and Cardone and Hindley (2009).

Sneak preview of $\boldsymbol{\lambda_{\to}}$ (Chapters 1, 2, 3)

Terms

Term *variables* $V \triangleq \{c, c', c'', \dots\}$

Terms Λ

$$\begin{cases} x \in V & \Rightarrow & x \in \Lambda \\ M, N \in \Lambda & \Rightarrow & (MN) \in \Lambda \\ M \in \Lambda, x \in V & \Rightarrow & (\lambda x M) \in \Lambda \end{cases}$$

Notations for terms
$x, y, z, \dots, F, G, \dots, \Phi, \Psi, \dots$ range over V
$M, N, L, \dots$ range over Λ
Abbreviations

$$\begin{array}{rcl} N_1 \dots N_n & \triangleq & (\cdot\cdot(MN_1)\dots N_n) \\ \lambda x_1 \dots x_n.M & \triangleq & (\lambda x_1(\dots(\lambda x_n.M)\cdot\cdot)) \end{array}$$

Standard terms: combinators

$$\begin{array}{rcl} \mathsf{I} & \triangleq & \lambda x.x \\ \mathsf{K} & \triangleq & \lambda xy.x \\ \mathsf{S} & \triangleq & \lambda xyz.xz(yz) \end{array}$$

Types

Type *atoms* $\mathbb{A}_\infty \triangleq \{\mathsf{c}, \mathsf{c}', \mathsf{c}'', \dots\}$

Types $\mathbb{T}$

$$\begin{cases} \alpha \in \mathbb{A} & \Rightarrow & \alpha \in \mathbb{T} \\ A, B \in \mathbb{T} & \Rightarrow & (A \to B) \in \mathbb{T} \end{cases}$$

Notations for types
$\alpha, \beta, \gamma, \dots$ range over $\mathbb{A}_\infty$
$A, B, C, \dots$ range over $\mathbb{T}$
Abbreviation
$A_1 \to A_2 \to \cdots \to A_n \triangleq (A_1 \to (A_2 \to \dots (A_{n-1} \to A_n)\cdot\cdot))$

Standard types: each $n \in \omega$ is interpreted as type $n \in \mathbb{T}$

$$\begin{array}{rcl} 0 & \triangleq & \mathsf{c} \\ n+1 & \triangleq & n \to 0 \\ (n+1)_2 & \triangleq & n \to n \to 0 \end{array}$$

Assignment of types to terms $\vdash M : A$ $(M \in \Lambda, A \in \mathbb{T})$

Basis: a set $\Gamma = \{x_1{:}A_1, \dots, x_n{:}A_n\}$, with $x_i \in V$ distinct
Type *assignment* (relative to a basis Γ) axiomatized by

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

Notations for assignment
'$x{:}A \vdash M : B$' stands for '$\{x{:}A\} \vdash M : B$'
'$\Gamma, x{:}A$' for '$\Gamma \cup \{x{:}A\}$' and '$\vdash M : A$' for '$\emptyset \vdash M : A$'

Standard assignments: for all $A, B, C \in \mathbb{T}$ one has

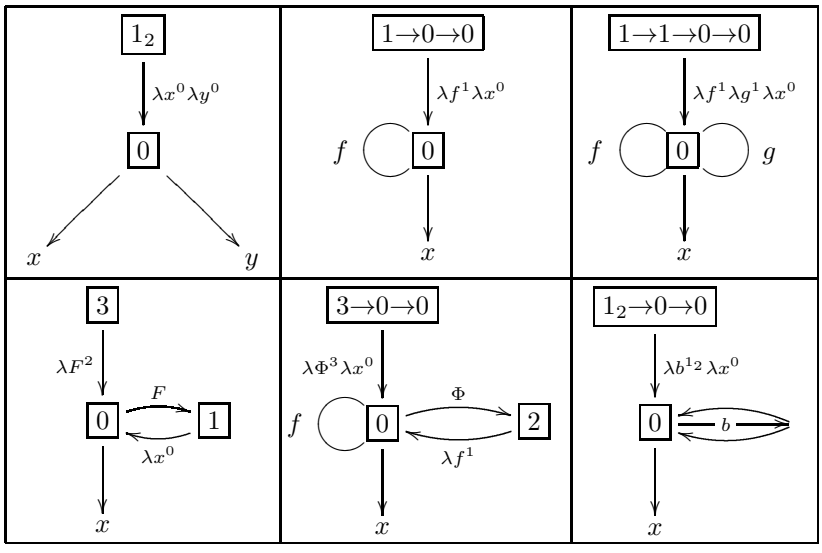
$$\begin{array}{lll} \vdash \mathsf{I} : A{\to}A & & \text{as } x{:}A \vdash x : A \\ \vdash \mathsf{K} : A{\to}B{\to}A & & \text{as } x{:}A, y{:}B \vdash x : A \\ \vdash \mathsf{S} : (A{\to}B{\to}C){\to}(A{\to}B){\to}A{\to}C & & \text{similarly} \end{array}$$

Canonical term-models built up from constants

The following types A play an important role in Sections 3.4, 3.5. Their normal inhabitants (i.e. terms M in normal form such that $\vdash M : A$) can be enumerated by the following schemes.

Type	Inhabitants (all possible $\boldsymbol{\beta\eta}^{-1}$-normal forms are listed)
1_2	$\boldsymbol{\lambda xy.x}, \boldsymbol{\lambda xy.y}$.
$1{\to}0{\to}0$	$\lambda fx.x, \lambda fx.fx, \lambda fx.f(fx), \lambda fx.f^3x, \ldots$; general pattern: $\boldsymbol{\lambda fx.f^n x}$.
3	$\lambda F.F(\lambda x.x), \lambda F.F(\lambda x.F(\lambda y.x)), \ldots; \boldsymbol{\lambda F.F(\lambda x_1.F(\lambda x_2.\ldots F(\lambda x_n.x_i)\cdots))}$.
$1{\to}1{\to}0{\to}0$	$\lambda fgx.x, \lambda fgx.fx, \lambda fgx.gx,$ $\lambda fgx.f(gx), \lambda fgx.g(fx), \lambda fgx.f^2x, \lambda fgx.g^2x,$ $\lambda fgx.f(g^2x), \lambda fgx.f^2(gx), \lambda fgx.g(f^2x), \lambda fgx.g^2(fx), \lambda fgx.f(g(fx)), \ldots;$ $\boldsymbol{\lambda fgx.w_{\{f,g\}}x}$, where $w_{\{f,g\}}$ is a 'word over $\Sigma = \{f, g\}$' which is 'applied' to x by interpreting juxtaposition 'fg' as function composition '$f \circ g = \lambda x.f(gx)$'.
$3{\to}0{\to}0$	$\lambda\Phi x.x, \lambda\Phi x.\Phi(\lambda f.x), \lambda\Phi x.\Phi(\lambda f.fx), \lambda\Phi x.\Phi(\lambda f.f(\Phi(\lambda g.g(fx)))), \ldots$ $\lambda\Phi x.\Phi(\lambda f_1.w_{\{f_1\}}x), \lambda\Phi x.\Phi(\lambda f_1.w_{\{f_1\}}\Phi(\lambda f_2.w_{\{f_1,f_2\}}x)), \ldots;$ $\boldsymbol{\lambda\Phi x.\Phi(\lambda f_1.w_{\{f_1\}}\Phi(\lambda f_2.w_{\{f_1,f_2\}}\cdots\Phi(\lambda f_n.w_{\{f_1,\ldots,f_n\}}x)\cdots))}$.
$1_2{\to}0{\to}0$	$\lambda bx.x, \lambda bx.bxx, \lambda bx.bx(bxx), \lambda bx.b(bxx)x, \lambda bx.b(bxx)(bxx), \ldots;$ $\boldsymbol{\lambda bx.t}$, where t is an element of the context-free language generated by the grammar `tree ::= x | (b tree tree)`.

This follows by considering the inhabitation machine, see Section 1.3, for each mentioned type.



We have juxtaposed the machines for types $1{\to}0{\to}0$ and $1{\to}1{\to}0{\to}0$, as they are similar, and also those for 3 and $3{\to}0{\to}0$. According to the type reducibility theory of Section 3.4 the types $1{\to}0{\to}0$ and 3 are equivalent and therefore they are presented together in the statement.

From the types 1_2, $1{\to}0{\to}0$, $1{\to}1{\to}0{\to}0$, $3{\to}0{\to}0$, and $1_2{\to}0{\to}0$ five canonical λ-theories and term-models will be constructed, that are strictly increasing (decreasing). The smallest theory is the good old simply typed $\lambda\boldsymbol{\beta\eta}$-calculus, and the largest theory corresponds to the minimal model, Definition 3.5.47, of the simply typed λ-calculus.

1

The Simply Typed Lambda Calculus

1.1 The systems $\boldsymbol{\lambda}^{\mathbb{A}}_{\rightarrow}$

Untyped lambda calculus

Recall the untyped lambda calculus denoted by $\boldsymbol{\lambda}$, see e.g. B[1984][1].

1.1.1 Definition The set of untyped λ-terms Λ is defined by the so-called '*simplified syntax*' in Fig. 1.1. This basically means that parentheses are left implicit.

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

Figure 1.1 Untyped lambda terms

This makes $\mathsf{V} = \{x, x', x'', \dots\}$.

1.1.2 Notation

(i) $x, y, z, \dots, x_0, y_0, z_0, \dots, x_1, y_1, z_1, \dots$ denote arbitrary variables.
(ii) $M, N, L, \dots$ denote arbitrary lambda terms.
(iii) $MN_1 \cdots N_k \equiv (..(MN_1) \cdots N_k)$, *association to the left*.
(iv) $\lambda x_1 \cdots x_n.M \equiv (\lambda x_1(..(\lambda x_n(M))..))$, *association to the right*.

1.1.3 Definition Let $M \in \Lambda$.

(i) The set of *free variables* of M, written $\mathrm{FV}(M)$, is defined as in Fig. 1.2. The variables in M that are not free are called *bound*.
(ii) If $\mathrm{FV}(M) = \emptyset$, then we say that M is *closed* or that it is a *combinator*:

$$\Lambda^{\emptyset} = \{M \in \Lambda \mid M \text{ is closed}\}.$$

[1] This is an abbreviation for the reference Barendregt (1984).

M	$\mathrm{FV}(M)$
x	$\{x\}$
PQ	$\mathrm{FV}(P) \cup \mathrm{FV}(Q)$
$\lambda x.P$	$\mathrm{FV}(P) - \{x\}$

Figure 1.2

Well-known combinators are $\mathsf{I} \equiv \lambda x.x, \mathsf{K} \equiv \lambda xy.y, \mathsf{S} \equiv \lambda xyz.xz(yz)$, $\mathbf{\Omega} \equiv (\lambda x.xx)(\lambda x.xx)$, and $\mathsf{Y} \equiv \lambda f.(\lambda x.f(xx))(\lambda x.f(xx))$. Officially one has $\mathsf{S} \equiv (\lambda x(\lambda x'(\lambda x''((xx'')(x'x'')))))$, according to Definition 1.1.1, so we see that the effort of learning the Notation 1.1.2 pays off.

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

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

and the special axiom (schemes) in Fig. 1.3.

$(\lambda x.M)N$	$=$	$M[x := N]$		($\boldsymbol{\beta}$-rule)
$\lambda x.Mx$	$=$	M,	if $x \notin \mathrm{FV}(M)$	($\boldsymbol{\eta}$-rule)

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

This theory can be analyzed by an appropriate notion of reduction.

1.1.5 Definition On Λ we define in Figure 1.4 the notions of $\boldsymbol{\beta}$*-reduction* and $\boldsymbol{\eta}$*-reduction*.

$(\lambda x.M)N$	$\to$	$M[x := N]$		($\boldsymbol{\beta}$)
$\lambda x.Mx$	$\to$	M,	if $x \notin \mathrm{FV}(M)$	($\boldsymbol{\eta}$)

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 Λ will often be considered modulo $=_{\boldsymbol{\beta}}$ or $=_{\boldsymbol{\beta\eta}}$.

1.1.6 Notation If we write $M = N$, then by default we mean $M =_{\boldsymbol{\beta\eta}} N$. This is the extensional version of the theory. See B[1984], where the default was $=_{\boldsymbol{\beta}}$.]

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

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

Proof See B[1984], Proposition 3.3.2. ■

1.1.8 Remark As in B[1984], Convention 2.1.12, we will not be concerned with α-conversion, renaming bound variables in order to avoid confusion between free and bound occurrences of variables. So we write $\lambda x.x \equiv \lambda y.y$. We do this by officially working on the α-equivalence classes; when dealing with a concrete term as representative of such a class, the bound variables will be chosen maximally fresh: different from the free variables and from each other. See, however, Section 7.4, in which we introduce λ-conversion on recursive types and show how it can be avoided in a way that is more effective than for terms.

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

1.1.9 Proposition (Church–Rosser theorem for $\boldsymbol{\lambda\beta\eta}$) *For the notions of reduction* $\twoheadrightarrow_{\boldsymbol{\beta}}$, $\twoheadrightarrow_{\boldsymbol{\eta}}$, *and* $\twoheadrightarrow_{\boldsymbol{\beta\eta}}$ *one has the following.*

(i) *Let* $M, N_1, N_2 \in \Lambda$. *Then (the* diamond property*)*

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

One also says that the reduction relations $\twoheadrightarrow_R$, *for* $R \in \{\boldsymbol{\beta}, \boldsymbol{\eta}, \boldsymbol{\beta\eta}\}$ *are* confluent.

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

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

Proof See Theorems 3.2.8 and 3.3.9 in B[1984]. ■

1.1.10 Definition

(i) Let T be a set of equations between λ-terms. Write

$$T \vdash_{\boldsymbol{\lambda\beta\eta}} M = N, \text{ or simply } T \vdash M = N$$

if $M = N$ is provable in $\boldsymbol{\lambda\beta\eta} + T$ with the equations in T added as axioms.

(ii) We say T is *inconsistent* if T proves every equation; otherwise consistent.

(iii) The equation $P = Q$, with $P, Q \in \Lambda$, is called *inconsistent*, written $P \# Q$, if $\{P = Q\}$ is inconsistent. Otherwise $P = Q$ is *consistent*.

The set $T = \emptyset$, i.e. the $\boldsymbol{\lambda\beta\eta}$-calculus itself, is consistent, as follows from the Church–Rosser theorem. Examples of inconsistent equations are $\mathsf{K}\#\mathsf{I}$ and $\mathsf{I}\#\mathsf{S}$. On the other hand $\boldsymbol{\Omega} = \mathsf{I}$ is consistent.

Simple types

Types in this part, also called *simple types*, are syntactic objects built from atomic types using the operator $\to$. 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 $A{\to}B$ 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 determine what terms fit together.

1.1.11 Definition

(i) Let $\mathbb{A}$ be a non-empty set. An element of $\mathbb{A}$ is called a *type atom*. The set of *simple types* over $\mathbb{A}$, written $\mathbb{T} = \mathbb{T}^{\mathbb{A}}$, is recursively defined as follows:

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

We assume that no relations like $\alpha{\to}\beta = \gamma$ hold between type atoms: $\mathbb{T}^{\mathbb{A}}$ is freely generated. Often one finds $\mathbb{T} = \mathbb{T}^{\mathbb{A}}$ given by a simplified syntax, see Fig. 1.5.

$$\boxed{\mathbb{T} \quad ::= \quad \mathbb{A} \mid \mathbb{T} \to \mathbb{T}}$$

Figure 1.5 Simple Types

(ii) Let $\mathbb{A}_0 = \{0\}$. Then we write $\mathbb{T}^0 = \mathbb{T}^{\mathbb{A}_0}$.
(iii) Let $\mathbb{A}_\infty = \{\mathsf{c}_0, \mathsf{c}_1, \mathsf{c}_2, \dots\}$. Then we write $\mathbb{T}^\infty = \mathbb{T}^{\mathbb{A}_\infty}$

We usually take $0 = \mathsf{c}_0$, then $\mathbb{T}^0 \subseteq \mathbb{T}^\infty$. If we write simply $\mathbb{T}$, then this refers to $\mathbb{T}^{\mathbb{A}}$ for an unspecified $\mathbb{A}$.

1.1.12 Notation

(i) If $A_1, \dots, A_n \in \mathbb{T}$, then

$$A_1{\to}\cdots{\to}A_n \triangleq (A_1{\to}(A_2{\to}\cdots{\to}(A_{n-1}{\to}A_n)..)).$$

That is, we use association to the right.
(ii) $\alpha, \beta, \gamma, \dots, \alpha_0, \beta_0, \gamma_0, \dots \alpha', \beta'\gamma', \dots$ denote arbitrary elements of $\mathbb{A}$.
(iii) $A, B, C, \dots$ denote arbitrary elements of $\mathbb{T}$.

1.1.13 Definition (Type substitution) Let $A, C \in \mathbb{T}^{\mathbb{A}}$ and $\alpha \in \mathbb{A}$. The result of substituting C for the occurrences of α in A, written $A[\alpha := C]$, is defined as follows:

$$\begin{aligned} \alpha[\alpha := C] &\triangleq C; \\ \beta[\alpha := C] &\triangleq \beta, && \text{if } \alpha \not\equiv \beta; \\ (A \to B)[\alpha := C] &\triangleq (A[\alpha := C]) \to (B[\alpha := C]). \end{aligned}$$

Assigning simple types

1.1.14 Definition ($\boldsymbol{\lambda}^{\mathrm{Cu}}_{\to}$)

(i) A (type assignment) *statement* is of the form

$$M : A,$$

with $M \in \Lambda$ and $A \in \mathbb{T}$. This statement is read 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* Γ, written

$$\Gamma \vdash^{\mathrm{Cu}}_{\boldsymbol{\lambda}_{\to}} M{:}A$$

(or $\Gamma \vdash_{\boldsymbol{\lambda}_{\to}} M : A$, 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:

$$(x{:}A) \in \Gamma \;\Rightarrow\; \Gamma \vdash x : A;$$

$$\Gamma \vdash M : (A \to B),\ \Gamma \vdash N : A \;\Rightarrow\; \Gamma \vdash (MN) : B;$$

$$\Gamma,\ x{:}A \vdash M : B \;\Rightarrow\; \Gamma \vdash (\lambda x.M) : (A \to B).$$

In the last rule $\Gamma, x{:}A$ is required to be a basis.
These rules are usually written as in Fig. 1.6.

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

1.1.15 Definition Let $\Gamma = \{x_1{:}A_1, \ldots, x_n{:}A_n\}$. Then

(i) $\mathrm{dom}(\Gamma) = \{x_1, \ldots, x_n\}$, the *domain* of Γ.

(ii) $x_1{:}A_1, \ldots, x_n{:}A_n \vdash_{\boldsymbol{\lambda}_{\to}} M : A$ denotes $\Gamma \vdash_{\boldsymbol{\lambda}_{\to}} M : A$.

(iii) In particular $\vdash_{\boldsymbol{\lambda}_{\to}} M : A$ stands for $\emptyset \vdash_{\boldsymbol{\lambda}_{\to}} M : A$.

(iv) $x_1, \ldots, x_n{:}A \vdash_{\boldsymbol{\lambda}_{\to}} M : B$ stands for $x_1{:}A, \ldots, x_n{:}A \vdash_{\boldsymbol{\lambda}_{\to}} M : B$.

(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 1.6 The system $\boldsymbol{\lambda}^{\mathrm{Cu}}_{\to}$ à la *Curry*

1.1.16 Example

(i) $\vdash_{\boldsymbol{\lambda}\to} \mathsf{I} : A{\to}A$;
$\vdash_{\boldsymbol{\lambda}\to} \mathsf{K} : A{\to}B{\to}A$;
$\vdash_{\boldsymbol{\lambda}\to} \mathsf{S} : (A{\to}B{\to}C){\to}(A{\to}B){\to}A{\to}C$.

(ii) Also one has

$$\begin{array}{rcll} x{:}A & \vdash_{\boldsymbol{\lambda}\to} & \mathsf{I}x & : A; \\ x{:}A, y{:}B & \vdash_{\boldsymbol{\lambda}\to} & \mathsf{K}xy & : A; \\ x{:}(A{\to}B{\to}C), y{:}(A{\to}B), z{:}A & \vdash_{\boldsymbol{\lambda}\to} & \mathsf{S}xyz & : C. \end{array}$$

(iii) The terms $\mathsf{Y}, \boldsymbol{\Omega}$ do not have a type. This is obvious after some effort. As we will see later, a systematic reason is that all typable terms have a normal form, but these two do not.

(iv) The term $\omega \equiv \lambda x.xx$ is in normal form but does not have a type either.

1.1.17 Notation Another way of writing these rules is sometimes found in the literature, see Fig. 1.7. In this version the basis is considered as implicit

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

Figure 1.7 $\boldsymbol{\lambda}^{\mathrm{Cu}}_{\to}$ alternative version