Lambda Calculus and Combinatory Logic

Lecture notes
by
Thomas Piecha

These notes are almost completely based on *Peter Schroeder-Heister's* course "Lambda-Kalkül und Kombinatorische Logik" (summer semester 1997) and the typeset notes produced by *Michael Arndt* as well as my notes taken in that course. Some corrections and additions were made.

Winter semester 2017/18

University of Tübingen
Department of Computer Science
and Department of Philosophy

Vorwort 1997

Dies ist ein Skriptum zu einer Vorlesung, die ich zuletzt im Sommersemester 1997 gehalten habe. In den ersten beiden Teilen orientiert es sich im wesentlichen am klassischen Lehrbuch von Hindley und Seldin, in den letzten beiden Teilen an Barendregts Kapitel über den getypten λ-Kalkül im *Handbook of Logic in Computer Science* (Band II). Das Skriptum soll zur Orientierung über das technische Gerüst des Themas dienen. Dementsprechend ist es nicht bis in alle Einzelheiten ausgearbeitet. So wurde auf Stilfragen wenig Rücksicht genommen. Auch wurden elementare, aber langwierige Beweise häufig weggelassen. Erläuternde Passagen zu Sinn und Zweck des λ-Kalküls sowie einzelner Begriffsbildungen sind ebenfalls nicht aufgezeichnet. Hierzu seien Leser auf die genannten Texte verwiesen.

Ich danke Michael Arndt für die Erstellung des Skriptums. Frau Natali Alt und Herrn Reinhard Kahle danke ich für eine kritische Durchsicht des Textes. Alle verbleibenden inhaltlichen Fehler gehen natürlich zu meinen Lasten.

Peter Schroeder-Heister

Vorwort 2016

Bei dieser aktualisierten Fassung des Skripts konnte ich auch auf die zweite Auflage des Lehrbuchs von Hindley und Seldin zurückgreifen:

> J. Roger Hindley und Jonathan P. Seldin: *Lambda-Calculus and Combinators, an Introduction*, Cambridge University Press, 2008 (reprinted 2010).

Es wurde wieder darauf verzichtet, die Verwendung dieser und anderer Quellen (siehe Literaturverzeichnis) im Einzelnen immer kenntlich zu machen.

Thomas Piecha

Preface 2017/18

This is a translation of the 2016 version, including some further corrections and additions.

Thomas Piecha

Contents

1 The untyped λ-calculus

The λ-calculus is a formal system in which the formation and application of functions can be expressed. We first consider the untyped version of λ-calculus, where functions are not restricted to data types.

Motivation of the syntax

The syntax of λ-calculus will allow us to systematically form an expression for a function of a variable x from an expression containing x, and likewise for any other variables. Consider the expression "$x - y$", which can be taken as a definition of either a function f of x or of a function g of y, usually written as follows:

$$f(x) = x - y \qquad\qquad g(y) = x - y$$

or

$$f : x \mapsto x - y \qquad\qquad g : y \mapsto x - y$$

Using λ-notation, one can write instead:

$$f = \lambda x.x - y \qquad\qquad g = \lambda y.x - y$$

Equations like

$$f(0) = 0 - y \qquad \text{and} \qquad f(1) = 1 - y$$

are then written as

$$(\lambda x.x - y)(0) = 0 - y \qquad \text{and} \qquad (\lambda x.x - y)(1) = 1 - y$$

respectively. For functions of two variables like e.g.

$$h(x, y) = x - y \qquad\qquad k(y, x) = x - y$$

one can write

$$h = \lambda xy.x - y \qquad\qquad k = \lambda yx.x - y$$

One can avoid such expressions for functions of several variables by allowing for functions of functions. For example, instead of the two-place function h, one can then use the following one-place function h':

$$h' = \lambda x.(\lambda y.x - y)$$

For each number a we have

$$h'(a) = \lambda y.a - y$$

and thus for each pair of numbers a, b:

$$(h'(a))(b) = (\lambda y.a - y)(b) = a - b = h(a, b)$$

It is therefore sufficient to consider only one-place functions. (The transition from many-place to one-place functions is called "currying" (after H. B. Curry) or "schönfinkeling" (after M. Schönfinkel).)

For the functions h and h' we have here assumed that the variables x and y range over numbers, i.e., x and y refer to objects of the type "number". To completely specify a function one would also have to declare the type of objects the function can return. We will consider types later, when we study typed λ-calculus. In untyped λ-calculus, a function like e.g. $(\lambda x.x)$ can be seen as a generic identity function; applied to an arbitrary object it simply returns the same object.

On semantics

The semantics of λ-calculus is operational. If M can be interpreted as a function, then (MN) is the result of an application of M to the argument N, in case the result exists. A term $(\lambda x.M)$ is interpreted as a function, whose value for an argument N is calculated by substituting N for x in M.

Example. The term $(\lambda x.x(xy))$ is interpreted as the operation of applying a function twice to an object y. Thus, for any term N, $(\lambda x.x(xy))N = N(Ny)$ in the sense that the left and the right term have the same interpretation.

1.1 Syntax and operational semantics

Let an infinite series of variables be given (in a fixed order).

Metalinguistic symbols for variables are: $u, v, w, x, y, z, x_1, x_2, x_3, \ldots$. (Different symbols denote different variables, unless stated otherwise.)

There are two variants of untyped λ-calculus:

- pure λ-calculus: no constants given.
- applied λ-calculus: finite or infinite set of constants given.

(The first two chapters deal only with untyped λ-calculus. We therefore omit the word "untyped".)

Definition 1.1 *λ-terms* (short: *terms*) are defined as follows: *λ-terms*

1. All variables and constants are λ-terms (*atoms*). *atoms*
2. If M and N are λ-terms, then (MN) is a λ-term (*application*), having M and N as immediate subterms. *application*
3. If M is a λ-term, then $(\lambda x.M)$ is a λ-term (*abstraction*), having x and M as immediate subterms. *abstraction*

Remark: Thus λ and e.g. λx are *not* λ-terms.

Further notions:

- The *length* of a term M is the number of occurrences of atoms in M. *length*
- *Subterms* of a term are the term itself, as well as the subterms of its immediate subterms. All subterms of a term except itself are its *proper* subterms. *subterm*
- We write $M[P]$, if P occurs as subterm at a certain position in M.
- In the context of $M[P]$ the expression $M[Q]$ means that one has replaced the occurrence of P by Q.
- The occurrence of a variable x in a term M is *bound*, if it belongs to a subterm $\lambda x.P$ of M, otherwise it is *free*. *bound* *free*
- If x has a free occurrence in M, then x is called a *free variable* of M. The set of free variables of a term M is called $\mathrm{FV}(M)$.
- M is called *closed*, if $\mathrm{FV}(M) = \emptyset$, otherwise *open*. *closed/open*
- A closed term without constants is called *combinator*. *combinator*
- *Metalinguistic variables:* $M, N, P, Q, R, S, T, \ldots$ for λ-terms; $a, b, c, \ldots$ for atoms.
- λ-terms can be abbreviated by omitting parentheses as follows, as long as no ambiguities can arise: *convention on parentheses*
  - Outermost parentheses can be omitted.
  - We use association to the left, i.e. $MNPQ$ stands for $((MN)P)Q$.
  - $\lambda x.MN$ stands for $(\lambda x.(MN))$.
  - $\lambda x_1 x_2 \ldots x_n.M$ stands for $(\lambda x_1.(\lambda x_2.(\ldots(\lambda x_n.M))))$.
- $M \simeq N$ denotes the *syntactic identity* of M and N.

  $M :\simeq N$ says that M and N are syntactically identical by definition; M is the definiendum, N is the definiens.

Remark. The terms of pure λ-calculus can be characterised by the following context-free grammar, if variables have the form $v_{0\ldots0}$: *grammar for terms*

- Terminals: $\{\lambda,\ .,\ (,\ ),\ v_0,\ {}_0\}$
- Non-terminals: $\{T, V\}$
- Start symbol: T
- Productions: $$\begin{aligned} T &\longrightarrow V \mid (TT) \mid (\lambda V.T) \\ V &\longrightarrow v_0 \mid V_0 \end{aligned}$$

Examples.

- $(\lambda v_0.(v_0 v_{00}))$ is a λ-term of length 3.

  Immediate subterms: v_0 and $(v_0 v_{00})$

  The term is open, since v_{00} occurs free. The variable v_0 occurs bound twice.

  Abbreviated: $\lambda v_0.v_0 v_{00}$
- In $\lambda xy.xy$ the term xy occurs once. It is $\lambda xy.xy \doteq (\lambda x.(\lambda y.(xy)))$.
- In $x(uv)(\lambda u.v(uv))uv$ the term uv occurs twice.

  It is $x(uv)(\lambda u.v(uv))uv \doteq ((((x(uv))(\lambda u.(v(uv))))u)v)$.
- In $\lambda u.uv$ the term $\lambda u.u$ does *not* occur.

  It is $\lambda u.u \doteq (\lambda u.u)$ and $\lambda u.uv \doteq (\lambda u.(uv))$.

Examples. Let x, y, z be any distinct variables. Then the following are λ-terms:

- $(\lambda x.(xy))$ (cp. last example above)
- $((\lambda y.y)(\lambda x.(xy)))$
- $(x(\lambda x.(\lambda x.x)))$

  This term has the form (MN), where N has two occurrences of λx. (This is permitted, but not recommended.)
- $(\lambda x.(yz))$

  This term has the form $(\lambda x.M)$, where x does not occur in M. This is called *vacuous abstraction*. Such terms stand for constant functions (i.e., for functions having for all inputs the same output).

Examples. On parentheses:

- $xyz(yx) \doteq (((xy)z)(yx))$
- $\lambda x.uxy \doteq (\lambda x.((ux)y))$
- $\lambda u.u(\lambda x.y) \doteq (\lambda u.(u(\lambda x.y)))$
- $(\lambda u.vuu)zy \doteq (((\lambda u.((vu)u))z)y)$
- $ux(yz)(\lambda v.vy) \doteq (((ux)(yz))(\lambda v.(vy)))$
- $(\lambda xyz.xz(yz))uvw \doteq ((((\lambda x.(\lambda y.(\lambda z.((xz)(yz)))))u)v)w)$

Examples. The following closed λ-terms (combinators) get a name:

- $\mathbf{I} :\doteq \lambda x.x$
- $\mathbf{K} :\doteq \lambda xy.x$
- $\mathbf{S} :\doteq \lambda xyz.xz(yz)$

Definition 1.2 For any M, N, x, we define the *substitution* $M[N/x]$ by induction on M to be the result of replacing every free occurrence of x in M by N, and changing bound variables to avoid clashes: *substitution*

1. $x[N/x] :\simeq N$,
2. $a[N/x] :\simeq a$, if $x \not\equiv a$,
3. $(PQ)[N/x] :\simeq (P[N/x]Q[N/x])$,
4. $(\lambda x.P)[N/x] :\simeq \lambda x.P$,
5. $(\lambda y.P)[N/x] :\simeq \lambda y.P[N/x]$, if $x \not\equiv y$ and not: $y \in \mathrm{FV}(N)$ and $x \in \mathrm{FV}(P)$,
6. $(\lambda y.P)[N/x] :\simeq \lambda z.P[z/y][N/x]$, if $x \not\equiv y$ and $y \in \mathrm{FV}(N)$ and $x \in \mathrm{FV}(P)$, where z is the first variable (in the enumeration of all variables) with $z \notin \mathrm{FV}(NP)$.

Examples.

– $(\lambda x.zy)[(uv)/x] \simeq \lambda x.zy$ (Def. 1.2, 4)

– $(\lambda y.x)[y/x] \simeq \lambda z.y$ (Def. 1.2, 6, for z being the first variable distinct from x and y)

– $(\lambda y.x(\lambda x.x))[(\lambda y.xy)/x] \simeq \lambda y.(\lambda y.xy)(\lambda x.x)$, since

$$\begin{aligned}
(\lambda y.x(\lambda x.x))[(\lambda y.xy)/x] &\simeq (\lambda y.(x(\lambda x.x)))[(\lambda y.xy)/x] && \text{(parentheses)}\\
&\simeq (\lambda y.(x(\lambda x.x))[(\lambda y.xy)/x]) && \text{(Def. 1.2, 5)}\\
&\simeq (\lambda y.(x[(\lambda y.xy)/x](\lambda x.x)[(\lambda y.xy)/x])) && \text{(Def. 1.2, 3)}\\
&\simeq (\lambda y.((\lambda y.xy)(\lambda x.x)[(\lambda y.xy)/x])) && \text{(Def. 1.2, 1)}\\
&\simeq (\lambda y.((\lambda y.xy)(\lambda x.x))) && \text{(Def. 1.2, 4)}\\
&\simeq \lambda y.(\lambda y.xy)(\lambda x.x) && \text{(parentheses)}
\end{aligned}$$

Definition 1.3

– We define *renaming of bound variables* as follows: *renaming of bound variables*

If $y \notin \mathrm{FV}(M)$, then let $P\big[\lambda x.M\big] \equiv_{1\alpha} P\big[\lambda y.M[y/x]\big]$.

– Then α-*conversion* (or *congruence*) is defined as follows: *α-conversion* *congruence*

Let $P \equiv_\alpha Q$, if $P \simeq P_1 \equiv_{1\alpha} P_2 \equiv_{1\alpha} \cdots \equiv_{1\alpha} P_n \simeq Q$.

(If $P \equiv_\alpha Q$, we say "P is congruent to Q" or "P α-converts to Q".)

Example. It is $\lambda xy.x(xy) \equiv_\alpha \lambda uv.u(uv)$:

$$\begin{aligned}
\lambda xy.x(xy) \simeq \lambda x.(\lambda y.x(xy)) &\equiv_{1\alpha} \lambda x.(\lambda v.x(xv))\\
&\equiv_{1\alpha} \lambda u.(\lambda v.u(uv)) \simeq \lambda uv.u(uv)
\end{aligned}$$

Lemma 1.4 *For all λ-terms M, N and all variables x we have:*

1. $M[x/x] \simeq M$.

2. *If* $x \notin \mathrm{FV}(M)$*, then* $M[N/x] \simeq M$.

3. *If* $x \in \mathrm{FV}(M)$*, then* $\mathrm{FV}(M[N/x]) = \mathrm{FV}(N) \cup (\mathrm{FV}(M) \setminus \{x\})$.

Proof. Direct application of Definition 1.2. QED

Lemma 1.5 *Let no variable bound in the* λ*-term* M *be free in* λ*-terms* P, Q *and* z*. Then the following holds:*

1. *If* $z \notin \mathrm{FV}(M)$*, then* $M[z/x][P/z] \simeq M[P/x]$.
2. *If* $z \notin \mathrm{FV}(M)$*, then* $M[z/x][x/z] \simeq M$.
3. $M[Q/x][P/x] \simeq M[(Q[P/x])/x]$.
4. *If* $y \notin \mathrm{FV}(P)$*, then* $M[Q/y][P/x] \simeq M[P/x][(Q[P/x])/y]$.
5. *If* $y \notin \mathrm{FV}(P)$ *and* $x \notin \mathrm{FV}(Q)$*, then* $M[Q/y][P/x] \simeq M[P/x][Q/y]$.

Proof. The restriction on bound variables in M excludes substitutions according to Definition 1.2 (6).
Proofs of (1), (3) and (4) are by term induction on M.
(2) follows from (1) and Lemma 1.4 (1); (5) follows from (4) and Lemma 1.4 (2). QED

Remark. If in Lemma 1.5 the restriction on variables bound in M is lifted and $\simeq$ is replaced by $\equiv_\alpha$, then the resulting statements (1)-(5) hold as well.

Lemma 1.6

1. *If* $P \equiv_\alpha Q$*, then* $\mathrm{FV}(P) = \mathrm{FV}(Q)$.
2. *For each term* P *and all variables* $x_1, \ldots, x_n$ *there exists a term* P' *with* $P \equiv_\alpha P'$*, where no variable* $x_1, \ldots, x_n$ *occurs bound in* P'.
3. $\equiv_\alpha$ *is an equivalence relation. That is, we have:*

   Reflexivity: $P \equiv_\alpha P$.

   Symmetry: If $P \equiv_\alpha Q$*, then* $Q \equiv_\alpha P$.

   Transitivity: If $P \equiv_\alpha Q$ *and* $Q \equiv_\alpha M$*, then* $P \equiv_\alpha M$.

Proof. Exercise. QED

Lemma 1.7 (Congruence of $\equiv_\alpha$**)**
If $M \equiv_\alpha M'$ *and* $N \equiv_\alpha N'$*, then* $M[N/x] \equiv_\alpha M'[N'/x]$.

Proof. See Hindley & Seldin (2008), appendix A1. QED

Remarks.

1. Substitution is well-behaved with respect to α-conversion. The result of a substitution where N has been α-converted to a term N' differs only by congruence from the result for a substitution with N.

2. By using α-conversion one can always separate variables in a term first, to avoid more complicated substitutions later.

3. Any two congruent terms will have identical interpretations.

Remark. In the following, we write $P \overset{\rhd_{1\beta}}{\underset{\equiv_{1\alpha}}{}} Q$ for "$P \rhd_{1\beta} Q$ or $P \equiv_{1\alpha} Q$" and $P \begin{smallmatrix}\equiv_{1\alpha}\\ \rhd_{1\beta}\\ \lhd_{1\beta}\end{smallmatrix} Q$ for "$P \equiv_{1\alpha} Q$ or $P \rhd_{1\beta} Q$ or $Q \rhd_{1\beta} P$", etc. (Note that $P \lhd_{1\beta} Q$ just means $Q \rhd_{1\beta} P$.)

Definition 1.8

– A term of the form $(\lambda x.M)N$ is called *β-redex* (short: *redex*, from *reducible expression*), and the corresponding term $M[N/x]$ is its *contractum*. *β-redex* *contractum*

– The operation of *β-contraction* is defined by: *β-contraction*

$$P\big[\underbrace{(\lambda x.M)N}_{\beta\text{-redex}}\big] \rhd_{1\beta} P\big[\underbrace{M[N/x]}_{\text{contractum}}\big].$$

– If for a term P there exists a (possibly empty) finite series of β-contractions and renamings of bound variables ending in a term Q, i.e. if

$$P \simeq P_1 \begin{smallmatrix}\rhd_{1\beta}\\ \equiv_{1\alpha}\end{smallmatrix} P_2 \begin{smallmatrix}\rhd_{1\beta}\\ \equiv_{1\alpha}\end{smallmatrix} \cdots \begin{smallmatrix}\rhd_{1\beta}\\ \equiv_{1\alpha}\end{smallmatrix} P_n \simeq Q$$

then we say that P *β-reduces* to Q. Notation: $P \rhd_\beta Q$. *β-reduction*

– Two terms P and Q are called *β-equal* (or *β-convertible*), if the following holds: *β-equality* *β-conversion*

$$P \simeq P_1 \begin{smallmatrix}\equiv_{1\alpha}\\ \rhd_{1\beta}\\ \lhd_{1\beta}\end{smallmatrix} P_2 \begin{smallmatrix}\equiv_{1\alpha}\\ \rhd_{1\beta}\\ \lhd_{1\beta}\end{smallmatrix} \cdots \begin{smallmatrix}\equiv_{1\alpha}\\ \rhd_{1\beta}\\ \lhd_{1\beta}\end{smallmatrix} P_n \simeq Q$$

Notation: $P =_\beta Q$.

– For a (possibly empty) finite or infinite series of β-contractions

$$P \simeq P_1 \rhd_{1\beta} P_2 \rhd_{1\beta} P_3 \rhd_{1\beta} \cdots$$

we call $(P_1, P_2, P_3, \ldots)$, or the given series itself, a *β-reduction series* of P. *β-reduction series*

Remark. By the two operations of α-conversion (resp. renaming of bound variables) and β-contraction an *operational semantics* for λ-terms is given. The interpretation of λ-terms is defined by how they behave under these two operations. *operational semantics*

Definition 1.9

– P is in *β-normal form* (short: *β-nf*), if no β-redex occurs in P. *β-normal form*

– If $P \rhd_\beta Q$ holds, and Q is in β-nf, then Q is called *β-normal form of P*.

(We then also say that P *has* β-normal form Q.)

– P is called *(weakly) normalisable*, if there exists a β-normal form of P. *normalisable*

– P is called *strongly normalisable*, if there is no infinite β-reduction series of P. *strongly normalisable*

Remark. Note the difference between *being* in β-nf and *having* a β-nf. To see whether a term *is* in β-nf, one just has to check whether it contains a β-redex or not. To show that a term *has* a β-nf, one has to show that the term reduces to a β-nf (i.e., one has to show that there exists a finite β-reduction series ending with a term in β-nf).

Examples.

- $(\lambda x.x)N \rhd_{1\beta} N$
- $(\lambda x.y)N \rhd_{1\beta} y$
- $(\lambda x.x(xy))N \rhd_{1\beta} N(Ny)$
- $(\lambda x.(\lambda y.yx)z)v \rhd_{1\beta} (\lambda x.zx)v \rhd_{1\beta} zv$, and zv is a β-normal form of $(\lambda x.(\lambda y.yx)z)v$.
- $\mathbf{\Omega} :\simeq (\lambda x.xx)(\lambda x.xx)$ does not have a β-normal form; $\mathbf{\Omega}$ is not in β-nf, and there is only an infinite β-reduction series: $(\lambda x.xx)(\lambda x.xx) \rhd_{1\beta} (\lambda x.xx)(\lambda x.xx) \rhd_{1\beta} \cdots$.

  However, λ-terms containing the $\mathbf{\Omega}$-combinator can have a β-normal form. For example, the term $(\lambda x.y)\mathbf{\Omega}$ has β-nf y, since $(\lambda x.y)\mathbf{\Omega} \rhd_{1\beta} y$.

  The term $(\lambda x.y)\mathbf{\Omega}$ is thus weakly normalisable; but it is not strongly normalisable, since there is an infinite β-reduction series: $(\lambda x.y)\mathbf{\Omega} \rhd_{1\beta} (\lambda x.y)\mathbf{\Omega} \rhd_{1\beta} \cdots$.
- $(\lambda x.xxy)(\lambda x.xxy) \rhd_{1\beta} (\lambda x.xxy)(\lambda x.xxy)y \rhd_{1\beta} (\lambda x.xxy)(\lambda x.xxy)yy \rhd_{1\beta} \cdots$.

  The term $(\lambda x.xxy)(\lambda x.xxy)$ has no β-normal form.

  The term $(\lambda u.v)((\lambda x.xxy)(\lambda x.xxy))$ is weakly but not strongly normalisable, having the β-normal form v.

Example. It is $(\lambda xyz.xzy)(\lambda xy.x) =_\beta (\lambda xy.x)(\lambda x.x)$, since

$$\begin{aligned}(\lambda xyz.xzy)(\lambda xy.x) &\equiv_\alpha (\lambda xyz.xzy)(\lambda uv.u) \\ &\rhd_{1\beta} \lambda yz.(\lambda uv.u)zy \\ &\rhd_{1\beta} \lambda yz.(\lambda v.z)y \\ &\rhd_{1\beta} \lambda yz.z\end{aligned} \qquad \text{and} \qquad \begin{aligned}(\lambda xy.x)(\lambda x.x) &\equiv_\alpha (\lambda xy.x)(\lambda u.u) \\ &\rhd_{1\beta} \lambda y.(\lambda u.u) \\ &\simeq \lambda yu.u \\ &\equiv_\alpha \lambda yz.z\end{aligned}$$

Lemma 1.10

1. *If* $P \rhd_\beta Q$, *then* $\mathrm{FV}(P) \supseteq \mathrm{FV}(Q)$.
2. *If* $P \equiv_\alpha P'$, $Q \equiv_\alpha Q'$ *and* $P \rhd_\beta Q$, *then* $P' \rhd_\beta Q'$.
3. *If* $P \equiv_\alpha P'$, $Q \equiv_\alpha Q'$ *and* $P =_\beta Q$, *then* $P' =_\beta Q'$.
4. *If* $M \rhd_\beta N$ *and* $P \rhd_\beta Q$, *then* $P[M/x] \rhd_\beta Q[N/x]$.
5. *If* $M =_\beta N$ *and* $P =_\beta Q$, *then* $P[M/x] =_\beta Q[N/x]$.

Proof. Exercise. QED

Lemma 1.11 *The class of all β-normal forms can be defined inductively by the following rules:*

1. *All atoms are in β-normal form.*
2. *If $M_1, \ldots, M_n$ are in β-normal form, then $aM_1 \ldots M_n$ is in β-normal form.*
3. *If M is in β-normal form, then $\lambda x.M$ is in β-normal form.*

That is, a β-normal form has the form $\lambda x_1 \ldots x_n.aM_1 \ldots M_m$, where the M_i have the same form.

Proof. We have to show that M is in β-normal form iff M can be produced by rules (1)-(3). The implication from right to left obvious.
It remains to show by induction on M that if M is in β-normal form, then M can be produced by rules (1)-(3).
Let M be in β-normal form.

- If $M \bumpeq a$, then M can be produced according to rule (1).
- If $M \bumpeq (PQ)$, then (by the induction hypothesis) P and Q can be produced by applications of rules (1)-(3), where P is not an abstraction. Thus $P \bumpeq a$ or $P \bumpeq aM_1 \ldots M_k$. Therefore $M \bumpeq aQ$ or $M \bumpeq aM_1 \ldots M_k Q$, which can be produced by rule (2).
- If $M \bumpeq \lambda x.P$, then (by the induction hypothesis) P can be produced by (1)-(3), and by an application of rule (3) also M. QED

Remark. Lemma 1.11 is about the class of all β-normal forms, *not* about the class of all λ-terms having a β-normal form.

Lemma 1.12 *An arbitrary λ-term has either the form shown in Lemma 1.11, or it contains a subterm of the form $\lambda x_1 \ldots x_n.\underbrace{(\lambda x.M)N}_{\textit{head redex}} M_1 \ldots M_m$, for $m, n \geq 0$.*

Proof. Consider the excluded subcase of $M \bumpeq (PQ)$ in the proof of Lemma 1.11. QED

Theorem 1.13 (**Church & Rosser, 1936**)

1. *If $P \rhd_\beta M$ and $P \rhd_\beta N$, then there exists a term T such that $M \rhd_\beta T$ and $N \rhd_\beta T$.*
2. *If $M =_\beta N$, then there exists a term T such that $M \rhd_\beta T$ and $N \rhd_\beta T$.*

Remarks.
- Property (1) is called *confluence of β-reduction*. Diagrammatically:

$$\begin{tikzcd} & P \arrow[dl, "\rhd_\beta"'] \arrow[dr, "\rhd_\beta"] & \\ M \arrow[dr, "\rhd_\beta"'] & & N \arrow[dl, "\rhd_\beta"] \\ & \exists T & \end{tikzcd}$$

– Property (2) indicates that two β-equal terms have the same interpretation, since there is always a term to which both reduce.

Example.

$$
\begin{tikzcd}
& (\lambda x.(\lambda y.yx)z)v \arrow[dl, "\rhd_{1\beta}"'] \arrow[dr, "\rhd_{1\beta}"] & \\
(\lambda x.zx)v \arrow[dr, "\rhd_{1\beta}"'] & & (\lambda y.yv)z \arrow[dl, "\rhd_{1\beta}"] \\
& zv &
\end{tikzcd}
$$

Proof of Theorem 1.13 (2). By induction on the number of steps from M to N.

Number of steps $= 0$: trivial.

Number of steps $= n + 1$: By the induction hypothesis we have

$$P_1 \rhd_\beta T', \dots, P_n \rhd_\beta T'$$

for a term T'. For the step from n to $n + 1$ the situation is the following:

$$
\begin{tikzcd}
M \bumpeq P_1 \arrow[dr, "\rhd_\beta"'] & =_\beta \cdots =_\beta & P_n \arrow[dl, "\rhd_\beta"] & \overset{\equiv_{1\alpha}}{\underset{\lhd_{1\beta}}{\rhd_{1\beta}}} & P_{n+1} \bumpeq N \\
& T' & & &
\end{tikzcd}
$$

and we have to show:

$$
\begin{tikzcd}
M \bumpeq P_1 \arrow[dr, "\rhd_\beta"'] & =_\beta \cdots =_\beta & P_n \arrow[dl, "\rhd_\beta"] & \overset{\equiv_{1\alpha}}{\underset{\lhd_{1\beta}}{\rhd_{1\beta}}} & P_{n+1} \bumpeq N \arrow[ddll, "\rhd_\beta"] \\
& T' \arrow[dr, "\rhd_\beta"'] & & & \\
& & \exists T & &
\end{tikzcd}
$$

Case 1 ($\equiv_{1\alpha}$): $T :\bumpeq T'$, since $P_{n+1} \equiv_{1\alpha} P_n \rhd_\beta T'$.

Case 2 ($\lhd_{1\beta}$): $T :\bumpeq T'$, since $P_{n+1} \rhd_{1\beta} P_n \rhd_\beta T'$.

Case 3 ($\rhd_{1\beta}$): Since $P_n \rhd_\beta T'$ and $P_n \rhd_{1\beta} P_{n+1}$, there exists according to Theorem 1.13 (1) a term T such that $T' \rhd_\beta T$ and $P_{n+1} \rhd_\beta T$:

$$
\begin{tikzcd}
& P_n \arrow[dl, "\rhd_\beta"'] \arrow[dr, "\rhd_{1\beta}"] & \\
T' \arrow[dr, "\rhd_\beta"'] & & P_{n+1} \arrow[dl, "\rhd_\beta"] \\
& T &
\end{tikzcd}
$$

Thus $M \rhd_\beta T$ (since $M =_\beta P_n$) and $N \rhd_\beta T$. QED

For the proof of Theorem 1.13 (1) we need some more definitions and lemmas. The difficulty lies in the fact that confluence of $\triangleright_\beta$ cannot simply be proved by first proving confluence of $\triangleright_{1\beta}$, that is, by proving

$$\text{if } P \triangleright_{1\beta} M \text{ and } P \triangleright_{1\beta} N, \text{ then } \exists T\colon M \triangleright_{1\beta} T \text{ and } N \triangleright_{1\beta} T. \tag{$*$}$$

Confluence of $\triangleright_\beta$ could be proved from $(*)$ by first proving the special case

$$\text{if } P \triangleright_{1\beta} M \text{ and } P \triangleright_{\beta} N, \text{ then } \exists T\colon M \triangleright_{\beta} T \text{ and } N \triangleright_{1\beta} T$$

by induction on the length of the reduction series (including α-conversions) from P to N; schematically:

$$\begin{tikzcd}
& P \arrow[dl, "\triangleright_{1\beta}"'] \arrow[dr, "\substack{\triangleright_{1\beta}\\ \equiv_\alpha}"] & & & \\
M \arrow[dr, "\triangleright_{1\beta}"'] & & N_1 \arrow[dl, "\triangleright_{1\beta}"] \arrow[dr, "\substack{\triangleright_{1\beta}\\ \equiv_\alpha}"] & & \\
& T_1 \arrow[dr, "\triangleright_{1\beta}"'] & & N_2 \arrow[dl, "\triangleright_{1\beta}"] \arrow[dr, dotted] & \\
& & T_2 \arrow[dr, dotted] & & N_n \simeq N \arrow[dl, "\triangleright_{1\beta}"] \\
& & & T_n \simeq T &
\end{tikzcd}$$

The general case could then be proved by a second induction on the length of the reduction series (again including α-conversions) from P to M:

$$\begin{tikzcd}
& & & P \arrow[dl, "\substack{\triangleright_{1\beta}\\ \equiv_\alpha}"'] \arrow[dr, "\triangleright_{\beta}"] & \\
& & P_1 \arrow[dl, "\substack{\triangleright_{1\beta}\\ \equiv_\alpha}"'] \arrow[dr, "\triangleright_{\beta}"'] & & N \arrow[dl, "\triangleright_{1\beta}"] \\
& P_2 \arrow[dl, dotted] \arrow[dr, "\triangleright_{\beta}"'] & & T_1 \arrow[dl, "\triangleright_{1\beta}"] & \\
M \arrow[dr, "\triangleright_{\beta}"'] & & T_2 \arrow[dl, dotted] & & \\
& T & & &
\end{tikzcd}$$

However, $(*)$ does not hold for all λ-terms. A counterexample is

$$P \simeq (\lambda y.uyy)(\mathbf{I}z) \qquad (\text{where } \mathbf{I} \simeq \lambda x.x)$$

In this case $P \rhd_{1\beta} u(\mathbf{I}z)(\mathbf{I}z)$ and $P \rhd_{1\beta} (\lambda y.uyy)z \rhd_{1\beta} uzz$. But $u(\mathbf{I}z)(\mathbf{I}z)$ cannot be contracted to uzz, that is, $u(\mathbf{I}z)(\mathbf{I}z) \not\rhd_{1\beta} uzz$.

The term $u(\mathbf{I}z)(\mathbf{I}z)$ could be reduced to uzz in one step by contracting the two non-overlapping redexes $(\mathbf{I}z)$ and $(\mathbf{I}z)$ in parallel. One could thus try to prove confluence for a relation of parallel reduction, from which confluence of $\rhd_\beta$ could be proved by two inductions as described for $(*)$.

However, parallel reduction cannot simply be defined as simultaneous non-overlapping contractions, since confluence fails for such a relation as well. A counterexample can be given for the term $P \simeq (\lambda x.R_1)R_2$, where $R_1 \simeq (\lambda y.xyz)w$ and $R_2 \simeq (\lambda u.u)v$ are both redexes. The term P contracts to $(\lambda y.R_2yz)w \simeq M$ by a (trivial) simultaneous non-overlapping contraction. By a simultaneous non-overlapping contraction of R_1 and R_2 in P we get $(\lambda x.xwz)v \simeq N$, which can only be reduced to N itself (by α-conversion) or to $vwz \simeq T$ (by contraction). But it is not possible to reduce M to either of those by using only simultaneous non-overlapping contractions. That is, we have the following situation (where arrows stand for simultaneous non-overlapping contractions):

$$\begin{tikzcd}
 & P \arrow[dl] \arrow[dr] & \\
M \arrow[rr, "/" marking] \arrow[dr, "/" marking] & & N \arrow[dl] \\
 & T &
\end{tikzcd}$$

Nevertheless, a notion of parallel reduction (called *minimal complete development*) can be given, for which confluence holds. We first define certain results of contraction.

Definition 1.14 Let R and S with $R \neq S$ be β-redexes in P, where $R \rhd_{1\beta} R'$, changing P to P'.

The *residual* $\mathrm{Res}(S, R)$ of S with respect to R is a redex in P', defined as follows: *residual*

1. R is a (proper) subterm of S. Then $\mathrm{Res}(S, R) :\simeq S[R']$.

   (That is, S has the form $(\lambda x.M)N$ and R is in M or in N. $R \rhd_{1\beta} R'$ changes M to M' or N to N', changing S to $(\lambda x.M')N$ or $(\lambda x.M)N'$ in P'. Then $\mathrm{Res}(S, R)$ is either $(\lambda x.M')N$ or $(\lambda x.M)N'$.)

2. R is not a subterm of S. Then $\mathrm{Res}(S, R) :\simeq S$.

   (That is, R and S do not overlap in P. Thus contracting R does not change the redex S, and S is the residual w.r.t. R in P'.)

Remark. Due to the way $\mathrm{Res}(S, R)$ will be used, some other possible cases besides (1) and (2) do not have to be considered.

Definition 1.15 Let $\mathcal{R} = \{R_1, \ldots, R_n\}$ be a set of redexes in P. A redex R_i is called *minimal*, if no R_j is a proper subterm of R_i. *minimal*

$P \rhd_{\mathrm{mcd}} Q$ (*minimal complete development* of P to Q), if Q is obtained from P by the *minimal complete development* following (non-deterministic) procedure:

(1) Choose a minimal element R_i in $\mathcal{R}$.

(2) β-contract R_i in P: $P \rhd_{1\beta} P'$.

(3) Let $\mathcal{R}' = \bigcup_{j \neq i} \mathrm{Res}(R_j, R_i)$. ($\mathcal{R}'$ has thus $n - 1$ elements.)

(4) If $\mathcal{R}' \neq \emptyset$, then go to step (1) with $\mathcal{R} := \mathcal{R}'$ and $P :\simeq P'$.

(5) If $\mathcal{R}' = \emptyset$, then let $Q \equiv_\alpha P'$.

(If $P \rhd_{\text{mcd}} Q$, we also say that P *mcd-reduces to* Q.)

Remarks.

1. The relation $\rhd_{\text{mcd}}$ is defined relative to a chosen set $\mathcal{R}$ of redexes. For different sets $\mathcal{R}$ there are thus different relations $\rhd_{\text{mcd}}$, and it would be more appropriate to make this distinction clear by naming them $\rhd^{\mathcal{R}}_{\text{mcd}}$, for each of the respective sets of redexes $\mathcal{R}$. As this would only complicate the presentation, we assume instead that the set of redexes is always chosen adequately, namely in such a way that all redexes of a considered term are elements of $\mathcal{R}$, or, in the proof of confluence for $\rhd_{\text{mcd}}$ (Lemma 1.18), in such a way that the conditions given there are met.
2. The relation $\rhd_{\text{mcd}}$ is not transitive, as the following example shows:

$$(\lambda x.xy)(\lambda z.z) \rhd_{\text{mcd}} (\lambda z.z)y \text{ and } (\lambda z.z)y \rhd_{\text{mcd}} y, \text{ but } (\lambda x.xy)(\lambda z.z) \not\rhd_{\text{mcd}} y.$$

   Note that the redex $(\lambda z.z)y$ is not a residual of $(\lambda x.xy)(\lambda z.z)$.
3. For $\mathcal{R} = \emptyset$ we have $P \rhd_{\text{mcd}} P$.

Example. We consider again $P \simeq (\lambda x.R_1)R_2$, where $R_1 \simeq (\lambda y.xyz)w$ and $R_2 \simeq (\lambda u.u)v$. It is $P \rhd_{\text{mcd}} vwz \simeq Q$. The set of redexes in P is $\mathcal{R} = \{(\lambda x.R_1)R_2, R_1, R_2\}$.

(1) Choose the minimal element R_1.

(2) $P \rhd_{1\beta} (\lambda x.xwz)R_2 \simeq P'$

(3) $\mathcal{R}' = \{\mathrm{Res}(P, R_1), \mathrm{Res}(R_2, R_1)\} = \{P', R_2\}$

(4) $\mathcal{R}' \neq \emptyset$

(1) Choose the minimal element R_2.

(2) $P' \rhd_{1\beta} (\lambda x.xwz)v \simeq P''$

(3) $\mathcal{R}'' = \{\mathrm{Res}(P', R_2)\} = \{P''\}$

(4) $\mathcal{R}'' \neq \emptyset$

(1) Choose the minimal element P''.

(2) $P'' \rhd_{1\beta} vwz \simeq P'''$

(3) $\mathcal{R}''' = \emptyset$

(5) Let $Q \equiv_\alpha P'''$.

Lemma 1.16 (Preservation of $\triangleright_{\mathrm{mcd}}$ by $\equiv_\alpha$)
If $P \triangleright_{\mathrm{mcd}} Q$ and $P \equiv_\alpha P'$, then $P' \triangleright_{\mathrm{mcd}} Q$.

Lemma 1.17 (Preservation of $\triangleright_{\mathrm{mcd}}$ by substitution)
If $M \triangleright_{\mathrm{mcd}} M'$ and $N \triangleright_{\mathrm{mcd}} N'$, then $M[N/x] \triangleright_{\mathrm{mcd}} M'[N'/x]$.

Lemma 1.18 (Confluence of $\triangleright_{\mathrm{mcd}}$)
If $P \triangleright_{\mathrm{mcd}} Q$ and $P \triangleright_{\mathrm{mcd}} R$, then there exists a term T such that $Q \triangleright_{\mathrm{mcd}} T$ and $R \triangleright_{\mathrm{mcd}} T$.

Proof. By Lemma 1.16 we can assume that the given mcd-reductions do not contain α-steps. The proof is by induction on the structure of P.

Case 1: $P \bumpeq a$. Then $Q \bumpeq R \bumpeq P$. Let $T :\bumpeq P$.

Case 2: $P \bumpeq \lambda x.P_1$. Then all redexes in P are in P_1, and since there are no α-steps we have $Q \bumpeq \lambda x.Q_1$ and $R \bumpeq \lambda x.R_1$, where $P_1 \triangleright_{\mathrm{mcd}} Q_1$ and $P_1 \triangleright_{\mathrm{mcd}} R_1$.

By the induction hypothesis there exists a term T_1 such that $Q_1 \triangleright_{\mathrm{mcd}} T_1$ and $R_1 \triangleright_{\mathrm{mcd}} T_1$. Let $T :\bumpeq \lambda x.T_1$.

Case 3: $P \bumpeq (P_1 P_2)$, and all redexes of $\mathcal{R}$ are in P_1 and P_2, that is, P itself is not reduced.

By the induction hypothesis there are terms T_1 and T_2 such that with

$$\begin{tikzcd} & P_1 \arrow[dl, "\triangleright_{\mathrm{mcd}}"'] \arrow[dr, "\triangleright_{\mathrm{mcd}}"] & \\ Q_1 \arrow[dr, "\triangleright_{\mathrm{mcd}}"'] & & R_1 \arrow[dl, "\triangleright_{\mathrm{mcd}}"] \\ & T_1 & \end{tikzcd} \quad \text{and} \quad \begin{tikzcd} & P_2 \arrow[dl, "\triangleright_{\mathrm{mcd}}"'] \arrow[dr, "\triangleright_{\mathrm{mcd}}"] & \\ Q_2 \arrow[dr, "\triangleright_{\mathrm{mcd}}"'] & & R_2 \arrow[dl, "\triangleright_{\mathrm{mcd}}"] \\ & T_2 & \end{tikzcd} \quad \text{also} \quad \begin{tikzcd} & (P_1 P_2) \arrow[dl, "\triangleright_{\mathrm{mcd}}"'] \arrow[dr, "\triangleright_{\mathrm{mcd}}"] & \\ (Q_1 Q_2) \arrow[dr, "\triangleright_{\mathrm{mcd}}"'] & & (R_1 R_2) \arrow[dl, "\triangleright_{\mathrm{mcd}}"] \\ & (T_1 T_2) & \end{tikzcd}$$

holds.

Let $T :\bumpeq (T_1 T_2)$.

Case 4: $P \bumpeq (\lambda x.M)N$, and the residual of P is contracted in only one of the two given mcd-reductions; we assume it is contracted in $P \triangleright_{\mathrm{mcd}} Q$.

Then $P \triangleright_{\mathrm{mcd}} Q$ has the form

$$P \bumpeq (\lambda x.M)N \triangleright_{\mathrm{mcd}} (\lambda x.M')N' \triangleright_{1\beta} M'[N'/x] \bumpeq Q$$

(where $M \triangleright_{\mathrm{mcd}} M'$ and $N \triangleright_{\mathrm{mcd}} N'$).

The other given mcd-reduction $P \triangleright_{\mathrm{mcd}} R$ has the form

$$P \bumpeq (\lambda x.M)N \triangleright_{\mathrm{mcd}} (\lambda x.M'')N'' \bumpeq R$$

(where $M \triangleright_{\mathrm{mcd}} M''$ and $N \triangleright_{\mathrm{mcd}} N''$).

By the induction hypothesis for M and N there exist terms M^+ and N^+ such that

$$
\begin{tikzcd}[column sep=small]
& M \arrow[dl, "\rhd_{\text{mcd}}"'] \arrow[dr, "\rhd_{\text{mcd}}"] & \\
M' \arrow[dr, "\rhd_{\text{mcd}}"'] & & M'' \arrow[dl, "\rhd_{\text{mcd}}"] \\
& M^+ &
\end{tikzcd}
\quad \text{and} \quad
\begin{tikzcd}[column sep=small]
& N \arrow[dl, "\rhd_{\text{mcd}}"'] \arrow[dr, "\rhd_{\text{mcd}}"] & \\
N' \arrow[dr, "\rhd_{\text{mcd}}"'] & & N'' \arrow[dl, "\rhd_{\text{mcd}}"] \\
& N^+ &
\end{tikzcd}
$$

Let $T :\bumpeq M^+[N^+/x]$. Then by Lemma 1.17:

$$Q \bumpeq M'[N'/x] \rhd_{\text{mcd}} M^+[N^+/x]$$

Furthermore, by first separating the α-conversions in the mcd-reductions for M'' and N'' as follows

$$M'' \rhd_{\text{mcd}} M^* \equiv_\alpha M^+ \qquad\qquad N'' \rhd_{\text{mcd}} N^* \equiv_\alpha N^+$$

where we assume that $M'' \rhd_{\text{mcd}} M^*$ and $N'' \rhd_{\text{mcd}} N^*$ are without α-steps, we obtain $R \rhd_{\text{mcd}} T$:

$$R \bumpeq (\lambda x.M'')N'' \rhd_{\text{mcd}} (\lambda x.M^*)N^* \rhd_{1\beta} M^*[N^*/x] \equiv_\alpha M^+[N^+/x]$$

Case 5: $P \bumpeq (\lambda x.M)N$, and both given mcd-reductions contract the residual of P.

Then $P \rhd_{\text{mcd}} Q$ has the form

$$P \bumpeq (\lambda x.M)N \rhd_{\text{mcd}} (\lambda x.M')N' \rhd_{1\beta} M'[N'/x] \bumpeq Q$$

and $P \rhd_{\text{mcd}} R$ has the form

$$P \bumpeq (\lambda x.M)N \rhd_{\text{mcd}} (\lambda x.M'')N'' \rhd_{1\beta} M''[N''/x] \bumpeq R$$

We argue as in case 4 and choose $T :\bumpeq M^+[N^+/x]$. By Lemma 1.17 we obtain the result. QED

Remark. The proof of confluence of $\rhd_{\text{mcd}}$ depends crucially on the fact that all redexes are already present in the initial term, which enables us to control them.

Proof of Theorem 1.13 (1). Using confluence of $\rhd_{\text{mcd}}$ (Lemma 1.18) one can show that

if $P \rhd_{\text{mcd}} M$ and $P \rhd_\beta N$, then there exists a term T such that $M \rhd_\beta T$ and $N \rhd_{\text{mcd}} T$

by an induction according to the scheme:

$$
\begin{tikzcd}
 & P \arrow[dl, "\rhd_{\mathrm{mcd}}"'] \arrow[dr, "\substack{\rhd_{1\beta}\\ \equiv_\alpha}"] & & & \\
M \arrow[dr, "\rhd_{\mathrm{mcd}}"'] & & N_1 \arrow[dl, "\rhd_{\mathrm{mcd}}"] \arrow[dr, "\substack{\rhd_{1\beta}\\ \equiv_\alpha}"] & & \\
 & T_1 \arrow[dr, "\rhd_{\mathrm{mcd}}"'] & & N_2 \arrow[dl, "\rhd_{\mathrm{mcd}}"] \arrow[dr, dotted] & \\
 & & T_2 \arrow[dr, dotted] & & N_n \simeq N \arrow[dl, "\rhd_{\mathrm{mcd}}"] \\
 & & & T_n \simeq T &
\end{tikzcd}
$$

Note that $P \rhd_{1\beta} Q$ implies $P \rhd_{\mathrm{mcd}} Q$, $P \rhd_{\mathrm{mcd}} Q$ implies $P \rhd_\beta Q$, and $\rhd_\beta$ is transitive. Therefore:

If $P \rhd_{1\beta} M$ and $P \rhd_\beta N$, then there exists a term T such that $M \rhd_\beta T$ and $N \rhd_\beta T$.

From this the result follows by an induction according to the scheme:

$$
\begin{tikzcd}
 & & & P \arrow[dl, "\substack{\rhd_{1\beta}\\ \equiv_\alpha}"'] \arrow[dr, "\rhd_\beta"] & \\
 & & P_1 \arrow[dl, "\substack{\rhd_{1\beta}\\ \equiv_\alpha}"'] \arrow[dr, "\rhd_\beta"] & & N \arrow[dl, "\rhd_\beta"] \\
 & P_2 \arrow[dl, dotted] \arrow[dr, "\rhd_\beta"] & & T_1 \arrow[dl, "\rhd_\beta"] & \\
M \arrow[dr, "\rhd_\beta"'] & & T_2 \arrow[dl, dotted] & & \\
 & T & & &
\end{tikzcd}
$$

That is, we obtain:

$$
\begin{tikzcd}
 & P \arrow[dl, "\rhd_\beta"'] \arrow[dr, "\rhd_\beta"] & \\
M \arrow[dr, "\rhd_\beta"'] & & N \arrow[dl, "\rhd_\beta"] \\
 & T &
\end{tikzcd}
$$

QED

Corollary 1.19

1. *If M and N are β-normal forms of P, then $M \equiv_\alpha N$.*

   (That is, β-normal forms are unique modulo congruence.)

2. *If $M =_\beta N$ and N is in β-nf, then $M \rhd_\beta N$.*

   (By the Church-Rosser Theorem, both M and N reduce to a term T. Since N is in β-normal form, $N \equiv_\alpha T$. Thus $M \rhd_\beta N$.)

3. *If $M =_\beta N$, then either both M and N have no β-nf, or both have the same β-nf (modulo congruence).*

4. *If $M =_\beta N$ and M, N are in β-normal form, then $M \equiv_\alpha N$.*

   (Thus λ-calculus is consistent in the sense that not all λ-terms are β-equal; in other words, the relation $=_\beta$ is not trivial. Consider the two terms $\lambda xy.xy$ and $\lambda xy.yx$. Both terms are in β-normal form, but they are not congruent. By the corollary, they are not β-equal.)

Definition 1.20

– A *leftmost reduction series* (short: *L-reduction series*) is a β-reduction series, in which always the leftmost redex is contracted. A redex $(\lambda x_1.M_1)N_1$ is *to the left of* $(\lambda x_2.M_2)N_2$ (in the considered term), if λx_1 stands to the left of λx_2. *leftmost reduction series*

– A *quasi-leftmost reduction series* (short: *QL-reduction series*) is a β-reduction series $(M_1, M_2, M_3, \ldots)$ such that for each M_i that is not the last element of the series there is an M_j and an M_{j+1} with $j \geq i$ such that in the contraction of M_j to M_{j+1} the leftmost redex in M_j is contracted. *quasi-leftmost reduction series*

Remarks.

1. A quasi-leftmost reduction series is a β-reduction series, in which every now and then the leftmost redex is contracted.
2. Leftmost reduction series correspond to *lazy evaluation* in programming languages.

Example. A leftmost reduction series for the term

$$\overbrace{((\lambda x.x)\underbrace{((\lambda y.yy)(\lambda z.z))}_{\text{redex 2}})}^{\text{redex 1}}\underbrace{((\lambda u.u)(\lambda v.v))}_{\text{redex 3}}$$

is given by:

$$\begin{aligned}
\underbrace{((\lambda x.x)((\lambda y.yy)(\lambda z.z)))}_{\text{leftmost redex}}((\lambda u.u)(\lambda v.v)) &\rhd_{1\beta} \underbrace{((\lambda y.yy)(\lambda z.z))}_{\text{leftmost redex}}((\lambda u.u)(\lambda v.v)) \\
&\rhd_{1\beta} \underbrace{((\lambda z.z)(\lambda z.z))}_{\text{leftmost redex}}((\lambda u.u)(\lambda v.v)) \\
&\rhd_{1\beta} \underbrace{(\lambda z.z)((\lambda u.u)(\lambda v.v))}_{\text{leftmost redex}} \\
&\rhd_{1\beta} \underbrace{(\lambda u.u)(\lambda v.v)}_{\text{leftmost redex}} \\
&\rhd_{1\beta} \lambda v.v
\end{aligned}$$

Theorem 1.21 *If a λ-term M has a β-normal form, then each leftmost reduction series beginning with M terminates (and thus also each quasi-leftmost reduction series for M).*

Remarks.

1. For the proofs see Barendregt (2012), § 13.2.
2. The contraposition of this theorem is especially useful: In order to show that a term M has *no* β-normal form, it is sufficient to show that there exists a non-terminating leftmost or quasi-leftmost reduction series beginning with M.

Theorem 1.22 *There exist fixed-point combinators Y, i.e., combinators such that* *fixed-point combinators*

1. $Yx =_\beta x(Yx)$,

or even

2. $Yx \rhd_\beta x(Yx)$.

Proof.

1. For the combinator
$$\Upsilon :\simeq \lambda x.(\lambda y.x(yy))\underbrace{(\lambda y.x(yy))}_{\simeq:M}$$
we have
$$\Upsilon x \rhd_\beta MM \rhd_\beta x(MM) \lhd_\beta x(\Upsilon x)$$
However, (2) does not hold for Υ. (This fixed-point combinator is due to Curry; cp. Rosenbloom, 1950.)
2. For the combinator
$$\Theta :\simeq (\lambda zx.x(zzx))\underbrace{(\lambda zx.x(zzx))}_{\simeq:N}$$
we have
$$\Theta x \rhd_\beta (\lambda x.x(NNx))x \rhd_\beta x(NNx) \simeq x(\Theta x)$$
Thus (1) obviously holds for Θ, too. (This fixed-point combinator was given by Turing, 1937.) QED

Corollary 1.23 *For each N there is an M such that for $n \geq 0$: $My_1 \ldots y_n =_\beta N[M/x]$.*

Proof. Let $M :\simeq Y(\lambda xy_1 \ldots y_n.N)$ for a fixed-point combinator Y. QED

Remarks.

1. If for Y we choose the fixed-point combinator Θ, i.e. $M :\simeq \Theta(\lambda xy_1 \ldots y_n.N)$, then even $My_1 \ldots y_n \rhd_\beta N[M/x]$ holds.
2. Each "intuitive" equation of the form $xy_1 \ldots y_n = N$ defining x by a term N, where x itself may occur in N (i.e. the equation is "self-referential" in this sense) has some term M as its solution.
3. A solution M does not always represent a computable function. The corollary only tells us that there always is a solution M in the "realm of λ-terms".

Theorem 1.24 *M is a fixed-point combinator, i.e. $Mx =_\beta x(Mx)$, iff M is a fixed point of* **SI**, *i.e. if $M =_\beta$* **SI***M.*

Proof. (See Barendregt, 2012, § 6.5.3.)
It is **SI** $=_\beta \lambda yz.z(yz)$.
Let M be a fixed point of **SI**, i.e. $M =_\beta \mathbf{SI}M$. Then $MN =_\beta \mathbf{SI}MN =_\beta N(MN)$, i.e. M is a fixed-point combinator.
Let $Mx =_\beta x(Mx)$. Then Mx is not in in normal form, since otherwise Mx and $x(Mx)$ would be congruent.
Thus $Mx \rhd_\beta xP$ and $x(Mx) \rhd_\beta xP$ for some term P. Moreover, $M \rhd_\beta \lambda z.N$, since M being a combinator it cannot begin with a variable. Therefore

$$\lambda x.Mx =_\beta \lambda x.(\lambda z.N)x =_\beta \lambda x.N[x/z] =_\beta M$$

(That is, η-conversion, as defined below, is provable for M.)
Thus $M =_\beta \lambda x.Mx =_\beta \lambda x.x(Mx) =_\beta \mathbf{SI}M$. QED

Definition 1.25

– A term of the form $\lambda x.Mx$ is called *η-redex*, if $x \notin \mathrm{FV}(M)$. The term M is its *contractum*. *η-redex* *contractum*

– The operation of *η-contraction* is defined by: *η-contraction*

$$P\big[\lambda x.Mx\big] \rhd_{1\eta} P\big[M\big], \text{ if } x \notin \mathrm{FV}(M).$$

– It is $P \rhd_{\beta\eta} Q$ (i.e. P *$\beta\eta$-reduces* to Q), if $P \simeq P_1 \substack{\equiv_{1\alpha} \\ \rhd_{1\beta} \\ \rhd_{1\eta}} P_2 \substack{\equiv_{1\alpha} \\ \rhd_{1\beta} \\ \rhd_{1\eta}} \cdots \substack{\equiv_{1\alpha} \\ \rhd_{1\beta} \\ \rhd_{1\eta}} P_n \simeq Q$. *$\beta\eta$-reduction*

– It is $P =_{\beta\eta} Q$ (i.e. P is *$\beta\eta$-equal* to Q), if $P \simeq P_1 \substack{\equiv_{1\alpha} \\ \rhd_{1\beta} \\ \lhd_{1\beta} \\ \rhd_{1\eta} \\ \lhd_{1\eta}} P_2 \substack{\equiv_{1\alpha} \\ \rhd_{1\beta} \\ \lhd_{1\beta} \\ \rhd_{1\eta} \\ \lhd_{1\eta}} \cdots \substack{\equiv_{1\alpha} \\ \rhd_{1\beta} \\ \lhd_{1\beta} \\ \rhd_{1\eta} \\ \lhd_{1\eta}} P_n \simeq Q$. *$\beta\eta$-equality*

(The $\beta\eta$-equal terms are also called *$\beta\eta$-convertible*. If no β-contractions occur, then we speak of *η-conversion* $=_\eta$.) *$\beta\eta$-conversion* *η-conversion*

Remark. Intuitively, $\beta\eta$-equality says that the meaning of a term depends only on its behaviour w.r.t. applications to another term (in other words, if extensionality obtains; cp. Lemma 1.38).

Lemma 1.26 *Lemma 1.10 holds as well for $\rhd_{\beta\eta}$.*

Theorem 1.27 *Church-Rosser holds for $\beta\eta$-reduction.*

Proof. See Hindley & Seldin (2008), appendix A2B.

1.2 The λ-definability of recursive functions

Definition 1.28 Let $M^0 N :\simeq N$ and $M^{n+1} N :\simeq M(M^n N)$.
Then *Church numerals* are defined as follows: $\underline{n} :\simeq \lambda xy.x^n y$. *Church numerals*

Remarks.

1. The notation M^n is here used only in combinations $M^n N$, never alone.
2. Cp. Wittgenstein (1922), 6.021: "A number is the exponent of an operation".
3. If $\underline{m} =_\beta \underline{n}$, then $m = n$, since Church numerals are in β-normal form.
4. We have $\underline{n} PQ \rhd_\beta P^n Q$.

Lemma 1.29 *There exist combinators* $\mathbf{N}, \mathbf{V}, \mathbf{D}$ *and* $\mathbf{R}$ *with the following properties:*

1. $\mathbf{N}\underline{k} =_\beta \underline{k+1}$ *(successor)*
2. $\mathbf{V}\underline{k+1} =_\beta \underline{k}$ *(predecessor)*
3. $\mathbf{D}PQ\underline{0} =_\beta P$ *(pairing- or conditional combinator; if-then-else)*
   $\mathbf{D}PQ\underline{k+1} =_\beta Q$
4. $\mathbf{R}PQ\underline{0} =_\beta P$ *(recursion combinator)*
   $\mathbf{R}PQ\underline{k+1} =_\beta Q\underline{k}(\mathbf{R}PQ\underline{k})$

Remarks.

1. $\mathbf{D}\underline{n}\,\underline{m}$ corresponds to the ordered pair $\langle n, m\rangle$, since according to (3) one can select the first or the second element.
2. $\mathbf{D}PQ\underline{n}$ corresponds to the operator *if* $n = 0$ *then P, else Q*.

Proof of Lemma 1.29.

1. $\mathbf{N} :\simeq \lambda uxy.x(uxy)$
3. $\mathbf{D} :\simeq \lambda xyz.z(\mathbf{K}y)x$
2. $\mathbf{V} :\simeq \lambda x.x(\lambda z.\mathbf{D}(\mathbf{N}(z\,\underline{0}))(z\,\underline{0}))(\mathbf{D}\underline{0}\,\underline{0})\underline{1}$

   We show by induction on k:

$$\underbrace{(\lambda z.\mathbf{D}(\mathbf{N}(z\,\underline{0}))(z\,\underline{0}))}_{\simeq:P}{}^{k+1}(\mathbf{D}\underline{0}\,\underline{0}) =_\beta \mathbf{D}\underline{k+1}\,\underline{k}$$

   Induction base:

$$P^1(\mathbf{D}\underline{0}\,\underline{0}) =_\beta \mathbf{D}(\mathbf{N}(\mathbf{D}\underline{0}\,\underline{0}\,\underline{0}))(\mathbf{D}\underline{0}\,\underline{0}\,\underline{0}) =_\beta \mathbf{D}(\mathbf{N}\underline{0})\underline{0} =_\beta \mathbf{D}\underline{1}\,\underline{0}$$

   Induction step: Let $P^{k+1}(\mathbf{D}\underline{0}\,\underline{0}) =_\beta \mathbf{D}\underline{k+1}\,\underline{k}$. Then

$$\begin{aligned} P^{k+2}(\mathbf{D}\underline{0}\,\underline{0}) &=_\beta P(P^{k+1}(\mathbf{D}\underline{0}\,\underline{0})) \\ &=_\beta P(\mathbf{D}\underline{k+1}\,\underline{k}) \qquad \text{(induction hypothesis)} \end{aligned}$$

$$
\begin{aligned}
&=_\beta \mathbf{D}(\mathbf{N}(\mathbf{D}\underline{k+1}\,\underline{k}\,\underline{0}))(\mathbf{D}\underline{k+1}\,\underline{k}\,\underline{0})\\
&=_\beta \mathbf{D}(\mathbf{N}\underline{k+1})\underline{k+1}\\
&=_\beta \mathbf{D}\underline{k+2}\,\underline{k+1}
\end{aligned}
$$

Therefore

$$
\begin{aligned}
\mathbf{V}\underline{k+1} &=_\beta \underline{k+1}P(\mathbf{D}\underline{0}\,\underline{0})\underline{1}\\
&=_\beta P^{k+1}(\mathbf{D}\underline{0}\,\underline{0})\underline{1}\\
&=_\beta \mathbf{D}\underline{k+1}\,\underline{k}\,\underline{1}\\
&=_\beta \underline{k}
\end{aligned}
$$

4. $\mathbf{R} :\equiv \mathbf{\Theta}(\lambda uxyz.\mathbf{D}x(y(\mathbf{V}z)(uxy(\mathbf{V}z)))z)$

   By Corollary 1.23, $\mathbf{R}$ is a solution of $\mathbf{R}xyz =_\beta \mathbf{D}x(y(\mathbf{V}z)(\mathbf{R}xy(\mathbf{V}z)))z$. QED

Remark. The recursion combinator $\mathbf{R}$ can also be given without using a fixed-point combinator. See Hindley & Seldin (2008), proof of theorem 4.11, where the recursion combinator is given as a strongly normalising term.

Definition 1.30 A λ-term P *defines* (or *represents*) a k-ary number-theoretic function f, if for all $m_1,\ldots,m_k$ the following holds: $P\underline{m_1}\ldots\underline{m_k} \simeq_\beta \underline{f(m_1,\ldots,m_k)}$. Using the abbreviation $\vec{m}$ for $m_1,\ldots,m_k$ the latter is also written $P\underline{\vec{m}} \simeq_\beta \underline{f(\vec{m})}$. *λ-definability*

$$
P\underline{\vec{m}} \simeq_\beta \underline{n} \text{ means that } \begin{cases} P\underline{\vec{m}} =_\beta \underline{n} \iff f(\vec{m}) = n, & \text{if } f(\vec{m}) \text{ is defined,}\\ P\underline{\vec{m}} \text{ has no } \beta\text{-normal form,} & \text{if } f(\vec{m}) \text{ is not defined.}\end{cases}
$$

Definition 1.31 The *primitive recursive functions* are defined inductively as follows: *primitive recursive function*

1. The *number* $0\colon \mathbb{N}^0 \to \mathbb{N}$ is a 0-ary primitive recursive function.
2. The *successor function* $s:\mathbb{N}\to\mathbb{N}$ with $s(n) = n+1$ is primitive recursive.
3. For each $n \geq 1$ and $i \leq n$ the *projection* $\pi_i^n : \mathbb{N}^n \to \mathbb{N}$, where

$$
\pi_i^n(m_1,\ldots,m_n) = m_i \qquad (m_1,\ldots,m_n \in \mathbb{N}),
$$

   is primitive recursive.

4. If $n \geq 1$ and $1 \leq i \leq k$, and $h\colon \mathbb{N}^k \to \mathbb{N}$ and all $g_i\colon \mathbb{N}^n \to \mathbb{N}$ are primitive recursive, then so is the function $f:\mathbb{N}^n\to\mathbb{N}$, defined by *composition* as follows:

$$
f(m_1,\ldots,m_n) = h(g_1(m_1,\ldots,m_n),\ldots,g_k(m_1,\ldots,m_n))
$$

5. If $k \geq 0$, and $g\colon \mathbb{N}^k \to \mathbb{N}$ and $h\colon \mathbb{N}^{k+2}\to\mathbb{N}$ are primitive recursive, then so is the

function $f : \mathbb{N}^{k+1} \to \mathbb{N}$, defined by *recursion* as follows:

$$\begin{aligned} f(0, m_1, \ldots, m_k) &= g(m_1, \ldots, m_k) \\ f(n+1, m_1, \ldots, m_k) &= h(n, f(n, m_1, \ldots, m_k), m_1, \ldots, m_k) \end{aligned}$$

Remark. Instead of $h(g_1(\vec{m}), \ldots, g_k(\vec{m}))$ we also write $(h \circ [g_1; \ldots; g_k])(\vec{m})$.

Theorem 1.32 *Every primitive recursive function is λ-definable.*

Proof. We present λ-terms that correspond to clauses (1)-(5) in Definition 1.31.

1. $0 \colon \mathbb{N}^0 \to \mathbb{N}$ is λ-defined by the term $\underline{0}$.
2. $s \colon \mathbb{N} \to \mathbb{N}$ is λ-defined by the term $\mathbf{N}$.
3. $\pi_i^n \colon \mathbb{N}^n \to \mathbb{N}$ is λ-defined by the term $\lambda x_1 \ldots x_n . x_i$.
4. If $h \colon \mathbb{N}^k \to \mathbb{N}$ and $g_i \colon \mathbb{N}^n \to \mathbb{N}$ are λ-defined by P and Q_i, respectively, $(1 \le i \le k)$, and $f : \mathbb{N}^n \to \mathbb{N}$ is given by $f(\vec{m}) := (h \circ [g_1; \ldots; g_k])(\vec{m})$ for all $\vec{m} = (m_1, \ldots, m_n)$, then the function $f : \mathbb{N}^n \to \mathbb{N}$ is λ-defined by the term $\lambda \vec{x}.P(Q_1\vec{x}) \ldots (Q_k\vec{x})$, where $Q_i\vec{x} :\simeq (\ldots(Q_i x_1) \ldots)x_n$.
5. If $g \colon \mathbb{N}^k \to \mathbb{N}$ and $h \colon \mathbb{N}^{k+2} \to \mathbb{N}$ are λ-defined by λ-terms P and Q, and $f : \mathbb{N}^{k+1} \to \mathbb{N}$ is given by

$$\begin{aligned} f(0, \vec{m}) &= g(\vec{m}) \\ f(n+1, \vec{m}) &= h(n, f(n, \vec{m}), \vec{m}) \end{aligned}$$

then f is λ-defined by the term $\lambda u\vec{x}.\mathbf{R}(P\vec{x})(\lambda uv.Quv\vec{x})u$.

Proof by induction on n:
Induction base:

$$\begin{aligned} (\lambda u\vec{x}.\mathbf{R}(P\vec{x})(\lambda uv.Quv\vec{x})u)\,\underline{0}\,\underline{\vec{m}} &=_\beta \mathbf{R}(P\,\underline{\vec{m}})(\lambda uv.Quv\,\underline{\vec{m}})\,\underline{0} \\ &=_\beta P\,\underline{\vec{m}} \\ &=_\beta \underline{g(\vec{m})} \qquad \text{(by presupposition on } g\text{)} \end{aligned}$$

Induction step:

$$\begin{aligned} (\lambda u\vec{x}.\mathbf{R}(P\vec{x})(\lambda uv.Quv\vec{x})u)\,\underline{n+1}\,\underline{\vec{m}} &=_\beta \mathbf{R}(P\,\underline{\vec{m}})(\lambda uv.Quv\,\underline{\vec{m}})\,\underline{n+1} \\ &=_\beta (\lambda uv.Quv\,\underline{\vec{m}})\,\underline{n}(\mathbf{R}(P\,\underline{\vec{m}})(\lambda uv.Quv\,\underline{\vec{m}})\,\underline{n}) \\ &=_\beta Q\,\underline{n}(\mathbf{R}(P\,\underline{\vec{m}})(\lambda uv.Quv\,\underline{\vec{m}})\,\underline{n})\,\underline{\vec{m}} \\ &=_\beta Q\,\underline{n}((\lambda u\vec{x}.\mathbf{R}(P\vec{x})(\lambda uv.Quv\vec{x})u)\,\underline{n}\,\underline{\vec{m}})\,\underline{\vec{m}} \\ &=_\beta Q\,\underline{n}\,\underline{f(n,\vec{m})}\,\underline{\vec{m}} \qquad \text{(induction hypothesis)} \\ &=_\beta \underline{h(n, f(n,\vec{m}), \vec{m})} \qquad \text{(by presupp. on } h\text{)} \end{aligned}$$

QED

Example. The function $add\colon \mathbb{N}^2 \to \mathbb{N}$ with $add(n,m) = n+m$ is defined primitive recursively as follows:

$$\begin{aligned} add(0,m) &= \pi_1^1(m) \\ add(n+1,m) &= (s \circ \pi_2^3)(n, add(n,m), m) \end{aligned}$$

The λ-definition of the function π_1^1 is the term $\lambda x.x \simeq \mathbf{I}$, and the λ-definition of composition $s \circ \pi_2^3$ is the term $\lambda y_1 y_2 y_3.\mathbf{N}((\lambda x_1 x_2 x_3.x_2) y_1 y_2 y_3)$.
By schema (5) the λ-definition of add is thus the term

$$\mathbf{Add} \simeq \lambda ux.\mathbf{R}(\mathbf{I}x)(\lambda uv.(\lambda y_1 y_2 y_3.\mathbf{N}((\lambda x_1 x_2 x_3.x_2) y_1 y_2 y_3))uvx)u$$

All redexes generated by this schematic translation can be contracted, yielding the following simplified term:

$$\mathbf{Add} \rhd_\beta \lambda ux.\mathbf{R}x(\lambda uv.\mathbf{N}v)u$$

The computation of $1+1$ (using Lemma 1.29, 4 and 1) is as follows:

$$\begin{aligned} \mathbf{Add}\,\underline{1}\,\underline{1} &\rhd_\beta \mathbf{R}\,\underline{1}(\lambda uv.\mathbf{N}v)\,\underline{1} \\ &=_\beta (\lambda uv.\mathbf{N}v)\,\underline{0}(\mathbf{R}\,\underline{1}(\lambda uv.\mathbf{N}v)\,\underline{0}) \\ &\rhd_\beta \mathbf{N}(\mathbf{R}\,\underline{1}(\lambda uv.\mathbf{N}v)\,\underline{0}) \\ &=_\beta \mathbf{N}\,\underline{1} \\ &=_\beta \underline{2} \end{aligned}$$

Definition 1.33 A function $f : \mathbb{N}^n \to \mathbb{N}$ (for $n \geq 0$) is called *partial recursive* iff there exist primitive recursive functions g and h, such that for all $\vec{m} = (m_1, \ldots, m_n)$ *partial recursive function*

$$f(\vec{m}) = h(\mu k.g(\vec{m},k) = 0)$$

where the μ-operator is defined as follows:

$$(\mu k.g(\vec{m},k) = 0) := \begin{cases} \text{the smallest } k\text{, such that } g(\vec{m},k) = 0 \text{ holds,} \\ \qquad\qquad\qquad\qquad \text{if there exists such a } k; \\ \text{undefined, if no such } k \text{ exists.} \end{cases}$$

If a smallest k always exists, then the function $f(\vec{m}) = h(\mu k.g(\vec{m},k) = 0)$ is called *recursive* or *total recursive*. *total recursive function*

Theorem 1.34 *Every partial recursive function is λ-definable.*

Proof. We consider partial recursive functions

$$f(\vec{m}) := h(\mu k.g(\vec{m}, k) = 0),$$

where g and h shall be primitive recursive functions, λ-defined by terms P and Q, respectively (cp. Theorem 1.31).

Approach: To compute $\mu k.g(\vec{m}, k) = 0$ one can write a program F such that $F(k)$ outputs k if $g(\vec{m}, k) = 0$, and otherwise calls $F(k+1)$. If we run this program for $k = 0$ we obtain the smallest k such that $g(\vec{m}, k) = 0$.

We thus have to find a corresponding term M such that (in crude terms) the following holds:

$$M\vec{x}y = (\textit{if } P\vec{x}y = 0, \textit{ then } y, \textit{ else } M\vec{x}y + 1)$$

The function $f(\vec{m})$ could then be represented by the term $\lambda\vec{x}.Q(M\vec{x}\underline{0})$.

Consider the equation

$$M\vec{x}y =_\beta \mathbf{D}y(M\vec{x}(\mathbf{N}y))(P\vec{x}y) \qquad (\star)$$

By Corollary 1.23, $\mathbf{\Theta}\,\underbrace{(\lambda u\vec{x}y.\mathbf{D}y(u\vec{x}(\mathbf{N}y))(P\vec{x}y))}_{\hat{=}:Z}$ is a solution of this equation

Claim: f is λ-defined by the term $\lambda\vec{x}.Q(\mathbf{\Theta}Z\vec{x}\underline{0})$.
It is sufficient to show

$$\mathbf{\Theta}Z\,\underline{\vec{m}}\,\underline{0} =_\beta \underline{k_1}, \text{ if } k_1 \text{ is the smallest } k \text{ with } g(\vec{m}, k) = 0$$

(since then $Q\underline{k_1} =_\beta \underline{f(\vec{m})}$).
We will show:

$$\text{If } g(\vec{m}, k) \neq 0 \text{ for all } k < k_1, \text{ then } \mathbf{\Theta}Z\,\underline{\vec{m}}\,\underline{0} =_\beta \mathbf{D}\underline{k_1}(\mathbf{\Theta}Z\,\underline{\vec{m}}\,\underline{k_1+1})(P\,\underline{\vec{m}}\,\underline{k_1}). \qquad (\star\star)$$

This entails:

$$\text{If } k_1 \text{ is the smallest } k \text{ with } g(\vec{m}, k) = 0, \text{ then } \mathbf{\Theta}Z\,\underline{\vec{m}}\,\underline{0} =_\beta \underline{k_1}, \text{ since } P\,\underline{\vec{m}}\,\underline{k_1} =_\beta \underline{0}.$$

Proof of $(\star\star)$ by induction on k_1:
For $k_1 = 0$: $\mathbf{\Theta}Z\,\underline{\vec{m}}\,\underline{0} =_\beta \mathbf{D}\underline{0}(\mathbf{\Theta}Z\,\underline{\vec{m}}\,\underline{1})(P\,\underline{\vec{m}}\,\underline{0})$ by $(\star)$
For $k_1 > 0$:

$$\mathbf{\Theta}Z\,\underline{\vec{m}}\,\underline{0} =_\beta \mathbf{D}\underline{k_1 - 1}(\mathbf{\Theta}Z\,\underline{\vec{m}}\,\underline{k_1})\underbrace{(P\,\underline{\vec{m}}\,\underline{k_1 - 1})}_{} \qquad \text{(induction hypothesis)}$$

$=_\beta \underline{l+1}$ for an $l \geq 0$, since $g(\vec{m}, k_1 - 1) \neq 0$;
thus $P\,\underline{\vec{m}}\,\underline{k_1 - 1} \neq_\beta \underline{0}$

$=_\beta \mathbf{\Theta} Z \underline{\vec{m}}\, \underline{k_1}$
$=_\beta \mathbf{D}\underline{k_1}(\mathbf{\Theta} Z \underline{\vec{m}}\, \underline{k_1+1})(P\underline{\vec{m}}\, \underline{k_1})$ by ($\star$)

It remains to show:

> If $f(\vec{m})$ is undefined, i.e., if $g(\vec{m}, k) \neq 0$ for all k and given $\vec{m}$, then $\mathbf{\Theta} Z \underline{\vec{m}}\, \underline{0}$ has no β-normal form.

By ($\star$) we have:

$$\mathbf{\Theta} Z \underline{\vec{m}}\, \underline{k_1} =_\beta \mathbf{D}\underline{k_1}(\mathbf{\Theta} Z \underline{\vec{m}}\, \underline{k_1+1})(P\underline{\vec{m}}\, \underline{k_1})$$

and since we have chosen $\mathbf{\Theta}$ (and not $\mathbf{\Upsilon}$) as fixed-point combinator we even have (cp. the remark after Corollary 1.23):

$$\mathbf{\Theta} Z \underline{\vec{m}}\, \underline{k_1} \rhd_\beta \mathbf{D}\underline{k_1}(\mathbf{\Theta} Z \underline{\vec{m}}\, \underline{k_1+1})(P\underline{\vec{m}}\, \underline{k_1})$$

By supposition $P\underline{\vec{m}}\, \underline{k} \neq_\beta \underline{0}$ for every $\underline{k}$. Therefore:

$$\begin{aligned}
\mathbf{\Theta} Z \underline{\vec{m}}\, \underline{0} &\rhd_\beta \mathbf{D}\underline{0}(\mathbf{\Theta} Z \underline{\vec{m}}\, \underline{1})(P\underline{\vec{m}}\, \underline{0}) \rhd_\beta \mathbf{\Theta} Z \underline{\vec{m}}\, \underline{1} \\
&\rhd_\beta \mathbf{D}\underline{1}(\mathbf{\Theta} Z \underline{\vec{m}}\, \underline{2})(P\underline{\vec{m}}\, \underline{1}) \rhd_\beta \mathbf{\Theta} Z \underline{\vec{m}}\, \underline{2} \\
&\rhd_\beta \mathbf{D}\underline{2}(\mathbf{\Theta} Z \underline{\vec{m}}\, \underline{3})(P\underline{\vec{m}}\, \underline{2}) \rhd_\beta \mathbf{\Theta} Z \underline{\vec{m}}\, \underline{3} \\
&\rhd_\beta \cdots
\end{aligned}$$

This β-reduction series is quasi-leftmost. That is, from time to time a leftmost term is contracted, namely a term of the form $\mathbf{D}MN\,\underline{l+1}$. Moreover, the reduction series is not terminating. The term $\mathbf{\Theta} Z \underline{\vec{m}}\, \underline{0}$ has therefore no β-normal form (cp. Theorem 1.20). QED

Theorem 1.35 *Every λ-definable function is partial recursive.*

Sketch of proof. Let f be an n-ary function which is λ-defined by a term P. Then

$f(k_1, \ldots, k_n) =$ that k for which the equation $P\underline{k_1}\ldots\underline{k_n} = \underline{k}$ is endformula of the shortest derivation in $\lambda\beta$ (see Section 1.3) ending with a formula of the form $P\underline{k_1}\ldots\underline{k_n} = \underline{m}$, if there exists such a derivation in $\lambda\beta$.

Otherwise $f(k_1, \ldots, k_n)$ is undefined.

Using Gödelisation (cp. the remark on p. 33) f can be seen to be a partial recursive function. QED

1.3 The formal theories $\lambda\beta$ and $\lambda\beta\eta$

Our treatment of the λ-calculus has so far been based on the operational semantics for λ-terms given by β-contraction, α-conversion and possibly η-contraction. We now consider formal systems that are sound and complete with respect to this semantics.

Definition 1.36 We define the *formal theories* $\lambda\beta$ and $\lambda\beta\eta$, whose *formulas* are all equations of the form $M = N$ for λ-terms M, N. *formal theories $\lambda\beta$ and $\lambda\beta\eta$*

(α) $\lambda x.M = \lambda y.M[y/x]$, if $y \notin \mathrm{FV}(M)$

(β) $(\lambda x.M)N = M[N/x]$

(ρ) $M = M$ (reflexivity)

$\lambda\beta\eta$ has in addition:

(η) $\lambda x.Mx = M$, if $x \notin \mathrm{FV}(M)$

The *rules* are:

(σ) $\dfrac{M = N}{N = M}$ (symmetry)

(τ) $\dfrac{M = N \quad N = P}{M = P}$ (transitivity)

(μ) $\dfrac{N = N'}{MN = MN'}$

(ν) $\dfrac{M = M'}{MN = M'N}$

(ξ) $\dfrac{M = M'}{\lambda x.M = \lambda x.M'}$ (weak extensionality)

(Formulas above the rule bar are called *premisses*, the formula below is called *conclusion*.)

Let Γ be a set of formulas and A a formula. A *derivation* of A from Γ in $\lambda\beta$ or $\lambda\beta\eta$ is a tree (branching upward), *derivation*

- whose leaves are axioms of $\lambda\beta$, resp. $\lambda\beta\eta$, or formulas in Γ,
- whose other (non-leaf) nodes are formulas inferred by a rule application from the immediately preceding formulas (i.e., formulas standing directly above),
- and whose root node is A.

The elements of Γ are also called *assumptions*, and the formula A is also called *endformula*. If $\mathcal{T}$ is a formal theory, then

$$\mathcal{T}, \Gamma \vdash A$$

means that A is derivable in $\mathcal{T}$ from assumptions Γ. If $\Gamma = \emptyset$, then the derivation of A in $\mathcal{T}$ is also called *proof* (of A in $\mathcal{T}$), i.e. *proof*

$$\lambda\beta \vdash M = N \text{ means that } M = N \text{ is provable in } \lambda\beta.$$

$$\lambda\beta\eta \vdash M = N \text{ means that } M = N \text{ is provable in } \lambda\beta\eta.$$

The two *formal theories* $\lambda\beta_{\triangleright}$ and $\lambda\beta\eta_{\triangleright}$ are defined like the systems $\lambda\beta$ and $\lambda\beta\eta$, respectively, but without rule (σ), i.e. without symmetry. *formal theories $\lambda\beta_{\triangleright}$ and $\lambda\beta\eta_{\triangleright}$*

Then

$$\lambda\beta_\rhd \vdash M = N \text{ means that } M = N \text{ is provable in } \lambda\beta_\rhd;$$
$$\lambda\beta\eta_\rhd \vdash M = N \text{ means that } M = N \text{ is provable in } \lambda\beta\eta_\rhd.$$

Example. The derivation

$$\dfrac{(\nu)\;\dfrac{(\xi)\;\dfrac{(\beta)\;\dfrac{}{(\lambda y.yx)z = zx}}{\lambda x.(\lambda y.yx)z = \lambda x.zx}}{(\lambda x.(\lambda y.yx)z)v = (\lambda x.zx)v} \qquad (\beta)\;\dfrac{}{(\lambda x.zx)v = zv}}{(\lambda x.(\lambda y.yx)z)v = zv}\;(\tau)$$

is a proof of $(\lambda x.(\lambda y.yx)z)v = zv$ in any of the above systems.

Lemma 1.37

1. $M \rhd_\beta N \iff \lambda\beta_\rhd \vdash M = N$
2. $M \rhd_{\beta\eta} N \iff \lambda\beta\eta_\rhd \vdash M = N$
3. $M =_\beta N \iff \lambda\beta \vdash M = N$
4. $M =_{\beta\eta} N \iff \lambda\beta\eta \vdash M = N$

Proof. Exercise. QED

Remark. The relations on the left side are based on the operational semantics for λ-terms. The provability relations on the right side are based on the respective formal theories. The lemma thus says that the formal theories are sound ("$\Longleftarrow$") and complete ("$\Longrightarrow$") for the respective corresponding operational semantics.

Lemma 1.38 *If, in the definition of $\lambda\beta\eta$, we replace axiom (η) by the rule*

$$(\chi)\;\frac{MP = NP \textit{ for all } P}{M = N}$$

or by the rule

$$(\zeta)\;\frac{Mx = Nx}{M = N}\;\textit{ if } x \notin \mathrm{FV}(NM)$$

then in the resulting systems the same equations are derivable as before.

Remark. Rule (χ) is a so-called ω-rule, i.e. a rule having infinitely many premisses. We only consider rule (ζ) here. Both rules, as well as the axiom schema (η), express *extensionality* of $=$. *extensionality*

For the equality of functions f and g extensionality means:

For all x: If $f(x) = g(x)$, then $f = g$.

One would not require extensionality for the equality of programs; one would rather

understand equality *intensionally* in this case. The theory $\lambda\beta$ (resp. $\lambda\beta_{\triangleright}$) is intensional *intensional*
as well, i.e. we do *not* have:

$$\text{For all terms } X\text{: If } \lambda\beta \vdash MX = NX\text{, then } \lambda\beta \vdash M = N.$$

(A counterexample can be obtained using the terms $M \simeq y$ and $N \simeq \lambda x.yx$.)
On the other hand, systems having (η), (χ) or (ζ) are extensional.

Proof of Lemma 1.38.
"$(\eta) \implies (\zeta)$":

$$(\tau)\,\dfrac{(\tau)\,\dfrac{(\sigma)\,\dfrac{(\eta)\,\dfrac{}{\lambda x.Mx = M}}{M = \lambda x.Mx} \qquad (\xi)\,\dfrac{Mx = Nx}{\lambda x.Mx = \lambda x.Nx}}{M = \lambda x.Nx} \qquad (\eta)\,\dfrac{}{\lambda x.Nx = N}}{M = N}$$

Thus applications of (ζ) can always be replaced by a derivation of this form.
"$(\zeta) \implies (\eta)$":

$$(\zeta)\,\dfrac{(\beta)\,\dfrac{}{(\lambda x.Mx)x = Mx}}{\lambda x.Mx = M}$$

Thus any application of the axiom (η) can be replaced by a derivation of this form. QED

1.4 Undecidability results

Theorem 1.39 (Church, 1936b)
The set $NF_\beta := \{M \mid M$ has β-normal form$\}$ is not decidable.

Sketch of proof. Consider an enumeration of the unary partial recursive functions $f_1, f_2, \ldots$ such that the function u with $u(m, n) :\simeq f_m(n)$ is partial recursive (such an enumeration exists).
Let u be λ-defined by the term P. Then the following holds:

$$P\,\underline{m}\,\underline{n} \text{ has } \beta\text{-normal form} \iff u(m,n) \text{ is defined.}$$

Suppose NF_β were decidable. Then the following function g would be a total recursive function:

$$g(n) := \begin{cases} u(n,n) + 1 & \text{if } u(n,n) \text{ is defined} \\ 1 & \text{otherwise} \end{cases}$$

Then $g = f_k$ for some k. Since f_k is total, we have

$$u(k,k) = f_k(k) = g(k) = u(k,k) + 1$$

Contradiction. QED

Remark. In what follows we presuppose that λ-terms can be encoded as natural numbers in such a way that different terms are always encoded by different numbers. Such an encoding is called *Gödelisation*. The number encoding a term M is called *Gödel number* of M, written $\ulcorner M \urcorner$. Then the λ-term $\underline{\ulcorner M \urcorner}$ is the church numeral that corresponds to the Gödel number $\ulcorner M \urcorner$ of M.

Theorem 1.40 (**Church, 1936b**) *The relation $=_\beta$ is not decidable.*

Sketch of proof. We can recursively enumerate all terms that are β-equal to a given term. For example, we can produce derivations of the corresponding equations in $\lambda\beta$, and use Lemma 1.37 (3).
Let

$$f(m,k) := \text{Gödel number of the } k\text{-th term that is } \beta\text{-equal to the term with Gödel number } m;$$

$$h(m) := \begin{cases} 0 & \text{if } m \text{ is the Gödel number of a term in } \beta\text{-normal form,} \\ 1 & \text{otherwise.} \end{cases}$$

The functions f and h are primitive recursive. We assume they are λ-defined by the terms F and H, respectively.
Consider the equation

$$Mxy =_\beta \mathbf{D}\underline{1}(Mx(\mathbf{N}y))(H(Fxy))$$

By Corollary 1.23, the following is a solution of this equation:

$$\Upsilon(\lambda uxy.\underbrace{\mathbf{D}\underline{1}(ux(\mathbf{N}y))(H(Fxy))}_{\hat{=}:V})$$

Then the following holds:

– $(\Upsilon(\lambda uxy.V))\underline{m}\,\underline{0} =_\beta \underline{1}$, if m is the Gödel number of a term that is β-equal to a term in β-normal form.

– Otherwise $(\Upsilon(\lambda uxy.V))\underline{m}\,\underline{0}$ has no β-normal form.

Suppose $=_\beta$ were decidable. Then $(\Upsilon(\lambda uxy.V))\underline{\ulcorner M \urcorner}\,\underline{0} =_\beta \underline{1}$ would be decidable. Thus NF_β would be decidable, contradicting Theorem 1.39. QED

Theorem 1.41 (**Church, 1936a**) *First-order logic PL is not decidable.*

Sketch of proof. The relation $=_\beta$ is not decidable.
Thus, by Lemma 1.37 (3), the formal theory $\lambda\beta$ is not decidable either.

$\lambda\beta$ can be translated into a PL-formula:

– we use Gödel numbers to encode λ-terms;

– (Gödel) numbers can be represented by PL-terms $x, f(x), f(f(x)), \ldots$ in an obvious way (interpret the term f as the successor function); we abbreviate these terms as follows:

$$\begin{aligned}
\overline{\overline{0}} &:\simeq x \\
\overline{\overline{1}} &:\simeq f(x) \\
\overline{\overline{2}} &:\simeq f(f(x)) \\
&\vdots
\end{aligned}$$

– we use the binary relation symbol E to represent equality $(=)$ in $\lambda\beta$;

– using E we translate the 8 axiom schemata and rules of $\lambda\beta$ into 8 PL-formulas F_1 to F_8 as follows:

1. Rule (σ)

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

   is translated into the PL-formula

$$E(\overline{\overline{\ulcorner M \urcorner}}, \overline{\overline{\ulcorner N \urcorner}}) \to E(\overline{\overline{\ulcorner N \urcorner}}, \overline{\overline{\ulcorner M \urcorner}})$$

2. Rule (τ)

$$\frac{M = N \quad N = P}{M = P}$$

   is translated into the PL-formula

$$E(\overline{\overline{\ulcorner M \urcorner}}, \overline{\overline{\ulcorner N \urcorner}}) \wedge E(\overline{\overline{\ulcorner N \urcorner}}, \overline{\overline{\ulcorner P \urcorner}}) \to E(\overline{\overline{\ulcorner M \urcorner}}, \overline{\overline{\ulcorner P \urcorner}})$$

And so on for the remaining rules and axiom schemata (variable conditions may be assumed and do not have to be translated).

Then the following holds:

$$\lambda\beta \vdash M = N \iff PL \vdash (F_1 \wedge \ldots \wedge F_8) \to E(\overline{\overline{\ulcorner M \urcorner}}, \overline{\overline{\ulcorner N \urcorner}})$$

Suppose PL were decidable. Then $\lambda\beta$ would be decidable as well, and by Lemma 1.37 (3) also β-equality $(=_\beta)$, contradicting Theorem 1.40. QED

2 Combinatory Logic

Combinatory logic (short: CL) is as powerful as λ-calculus, but without making use of bound variables. This simplifies substitution, and we do not need α-conversion. However, CL-terms are less transparent than λ-terms.
To motivate the abandonment of bound variables we consider the law of commutativity for addition:

$$\text{for all } x, y\text{: } x + y = y + x$$

where the variables x and y occur bound. To avoid this binding of variables we first introduce an addition operator A:

$$A(x, y) = x + y \qquad (\text{for all } x, y)$$

and then define an operator $\mathbf{C}$ by

$$(\mathbf{C}(f))(x, y) = f(y, x) \qquad (\text{for all } f, x, y)$$

The law of commutativity can then be given as follows:

$$A = \mathbf{C}(A)$$

In this formulation no bound variables occur (at least not immediately).
The operator $\mathbf{C}$ is an example of a combinator. Further examples are:

$\mathbf{B}$	$(\mathbf{B}(f, g))(x) = f(g(x))$	composition of two functions
$\mathbf{B}'$	$(\mathbf{B}'(f, g))(x) = g(f(x))$	reverse composition of two functions
$\mathbf{S}$	$(\mathbf{S}(f, g))(x) = f(x, g(x))$	(stronger) composition of two functions
$\mathbf{I}$	$\mathbf{I}(f) = f$	identity
$\mathbf{K}$	$(\mathbf{K}(c))(x) = c$	forms constant functions

2.1 Syntax and operational semantics

Let an infinite series of variables be given (in a fixed order). We assume that these variables are the same as in λ-calculus.
$\mathbf{K}$ and $\mathbf{S}$ shall be given as constants. If a system contains additional constants, then it is called *applied*, otherwise *pure*. We only investigate pure combinatory logic.

Definition 2.1 *CL-terms* are defined as follows: *CL-terms*

1. All variables and constants (i.e. *atoms*) are CL-terms. *atoms*
2. If X and Y are CL-terms, then the *application* (XY) is a CL-term as well, having X and Y as immediate subterms. *application*

Further notions:

- A *closed* CL-term has no variables. *closed*
- A *combinator* has only **K** and **S** as atoms. *combinator*
- *Metalinguistic variables* (also with indexes):
  - for CL-terms: $U, V, W, X, Y, Z, \ldots$
  - for atoms: $a, b, c, \ldots$
- $\mathrm{FV}(X)$ denotes the set of variables in X.
- *On parentheses*: *convention on parentheses*
  - Outermost parentheses can be omitted.
  - We use association to the left, i.e. $UVWX$ stands for $((UV)W)X$.
- $X \simeq Y$ denotes again the *syntactic identity* of X and Y.
- The *length* of a term and the notion of (*proper*) *subterm* are defined as for λ-terms. *length*

Examples.

- $\mathbf{S}xy(\mathbf{K}y)(\mathbf{KKSS})$ is a CL-term.
- $\mathbf{S}(\mathbf{KS})$ is a CL-term (and a combinator).

Definition 2.2 As no bound variables can occur in CL-terms, *substitution* $Y[U/x]$ is simply defined as follows: *substitution*

1. $x[U/x] :\simeq U$,
2. $a[U/x] :\simeq a$, if $x \not\simeq a$,
3. $(VW)[U/x] :\simeq (V[U/x]W[U/x])$.

Definition 2.3

- A term of form $\mathbf{K}XY$ or $\mathbf{S}XYZ$ is called *weak redex* (short: *redex*). *weak redex*
- The operation of *weak contraction* is defined by: *weak contraction*

$$U\big[\mathbf{K}XY\big] \rhd_{1w} U\big[X\big]$$

$$U\big[\mathbf{S}XYZ\big] \rhd_{1w} U\big[XZ(YZ)\big]$$

- If there is a (possibly empty) finite series of weak contractions from a term X to a term Y, i.e. if

$$X \simeq V_1 \rhd_{1w} V_2 \rhd_{1w} \cdots \rhd_{1w} V_n \simeq Y$$

  then X *(weakly) reduces to* Y. Notation: $X \rhd_w Y$. *weak reduction*

- Two terms X and Y are *weakly equal* (or *weakly convertible*), if *weak equality* *weak conversion*

$$X \simeq V_1 \substack{\rhd_{1w} \\ \lhd_{1w}} V_2 \substack{\rhd_{1w} \\ \lhd_{1w}} \cdots \substack{\rhd_{1w} \\ \lhd_{1w}} V_n \simeq Y$$

  (where $U \substack{\rhd_{1w} \\ \lhd_{1w}} W$ means "$U \rhd_{1w} W$ or $W \rhd_{1w} U$"). Notation: $X =_w Y$.

– For a (possibly empty) finite or infinite series of weak contractions

$$X \simeq X_1 \rhd_{1w} X_2 \rhd_{1w} X_3 \rhd_{1w} \cdots$$

we call $(X_1, X_2, X_3, \ldots)$, or the given series itself, a *weak reduction series* for X. *weak reduction series*

Definition 2.4

– X is in *weak normal form* (short: *weak nf*), if X contains no weak redex. *weak normal form*

– If $X \rhd_w Y$ holds, and Y is in weak normal form, then Y is a *weak normal form of* X. (We also say that X *has* the weak normal form Y.)

– X is *(weakly) normalisable*, if there is a weak normal form of X. *normalisable*

– X is *strongly normalisable*, if there is no infinite weak reduction series for X. *strongly normalisable*

Examples.

1. Let $\mathbf{B} :\simeq \mathbf{S}(\mathbf{KS})\mathbf{K}$. Then $\mathbf{B}XYZ \rhd_w X(YZ)$:

$$\begin{aligned}
\mathbf{S}(\mathbf{KS})\mathbf{K}XYZ &\rhd_w \mathbf{KS}X(\mathbf{K}X)YZ && \text{by } \mathbf{S}(\mathbf{KS})\mathbf{K}X \rhd_{1w} \mathbf{KS}X(\mathbf{K}X)\\
&\rhd_w \mathbf{S}(\mathbf{K}X)YZ && \text{by } \mathbf{KS}X \rhd_{1w} \mathbf{S}\\
&\rhd_w \mathbf{K}XZ(YZ) && \text{by } \mathbf{S}(\mathbf{K}X)YZ \rhd_{1w} \mathbf{K}XZ(YZ)\\
&\rhd_w X(YZ) && \text{by } \mathbf{K}XZ \rhd_{1w} X
\end{aligned}$$

(Cp. the operator **B** mentioned earlier.)

2. It is $\mathbf{SKK}X \rhd_w X$, i.e. **SKK** behaves as the identity operator **I** mentioned earlier.

   We define $\mathbf{I} :\simeq \mathbf{SKK}$.

Remarks.

1. Weak reduction $\rhd_w$ is invariant under substitution (cp. Lemma 1.10), i.e. it holds: If $X \rhd_w Y$ and $U \rhd_w V$, then $U[X/z] \rhd_w V[Y/z]$.
2. The Church-Rosser property holds (cp. Theorem 1.13):
   – If $X \rhd_w U$ and $X \rhd_w V$, then there exists a term T such that $U \rhd_w T$ and $V \rhd_w T$.
   – If $X =_w Y$, then there exists a term T such that $X \rhd_w T$ and $Y \rhd_w T$.

2.2 The formal theory CLw

Definition 2.5 We define the *formal theory CLw*, whose *formulas* are all equations of the form $X = Y$ for CL-terms X, Y. *formal theory CLw*

Axioms of CLw are all instances of the following *axiom schemata*:

(K) $\mathbf{K}XY = X$

(S) $\mathbf{S}XYZ = XZ(YZ)$

$(\rho)\quad X = X \qquad$ (reflexivity)

The *rules* are:

$(\sigma)\ \dfrac{X = Y}{Y = X} \qquad$ (symmetry)

$(\tau)\ \dfrac{X = Y \quad Y = Z}{X = Z} \qquad$ (transitivity)

$(\mu)\ \dfrac{X = X'}{YX = YX'}$

$(\nu)\ \dfrac{Y = Y'}{YX = Y'X}$

The notions *derivation* and *proof* are defined analogously to Definition 1.36, and

$$\mathrm{CL}w \vdash X = Y \text{ means that } X = Y \text{ is provable in } \mathrm{CL}w,$$
$$\mathrm{CL}w_{\rhd} \vdash X = Y \text{ means that } X = Y \text{ is provable in } \mathrm{CL}w \text{ without rule } (\sigma).$$

Lemma 2.6

1. $X =_w Y \iff CLw \vdash X = Y.$
2. $X \rhd_w Y \iff CLw_{\rhd} \vdash X = Y.$

Proof. Exercise. QED

2.3 On the relation between λ-calculus and CL

To investigate the relation between λ-calculus and combinatory logic we consider translations of CL-terms into λ-terms and vice versa.

Definition 2.7 For a CL-term X the λ-term X_λ is defined as follows:

1. $x_\lambda :\simeq x$
2. $\mathbf{K}_\lambda :\simeq \lambda xy.x$
3. $\mathbf{S}_\lambda :\simeq \lambda xyz.xz(yz)$
4. $(XY)_\lambda :\simeq X_\lambda Y_\lambda$

(We here identify congruent λ-terms; this means that $M \simeq N$ in case $M \equiv_\alpha N$, for λ-terms M, N.)

Lemma 2.8

1. *If $X \rhd_w Y$, then $X_\lambda \rhd_\beta Y_\lambda$.*
2. *If $X =_w Y$, then $X_\lambda =_\beta Y_\lambda$.*

Proof. Use $\mathrm{CL}w_{\rhd}$ and $\lambda\beta_{\rhd}$, resp. $\mathrm{CL}w$ and $\lambda\beta$. QED

Remark. The converse direction does *not* hold.
For example, we have $\mathbf{S}_\lambda\mathbf{K}_\lambda =_\beta \mathbf{K}_\lambda(\mathbf{S}_\lambda\mathbf{K}_\lambda\mathbf{K}_\lambda)$, but not $\mathbf{SK} =_w \mathbf{K}(\mathbf{SKK})$. None of the CL-terms contracts, while both λ-terms contain redexes.

Definition 2.9 For a λ-term M the CL-term M_{CL} is defined as follows:

1. $x_{\mathrm{CL}} :\simeq x$
2. $(MN)_{\mathrm{CL}} :\simeq M_{\mathrm{CL}} N_{\mathrm{CL}}$
3. $(\lambda x.M)_{\mathrm{CL}} :\simeq [x].M_{\mathrm{CL}}$

where $[x].Y$ ("abstraction x dot Y") for CL-terms Y is defined as follows:

1. $[x].x :\simeq \mathbf{SKK}$ (abbreviated: $\mathbf{I} :\simeq \mathbf{SKK}$);
2. $[x].Y :\simeq \mathbf{K}Y$, if $x \notin \mathrm{FV}(Y)$;
3. $[x].Ux :\simeq U$, if $x \notin \mathrm{FV}(U)$;
4. $[x].(UV) :\simeq \mathbf{S}([x].U)([x].V)$, if none of the preceding cases applies.

Example. It is $[x].xxz \simeq \mathbf{S}([x].xx)([x].z) \simeq \mathbf{S}(\mathbf{S}([x].x)([x].x))(\mathbf{K}z) \simeq \mathbf{S}(\mathbf{SII})(\mathbf{K}z)$.

Remarks.

1. The expression $[x].Y$ is metalinguistic; $[x].Y$ is not a CL-term, but only *represents* a CL-term.
2. It is $x \notin \mathrm{FV}([x].Y)$. In this respect $[x]$ behaves like a variable-binding operator.

Lemma 2.10 *We have* $([x].Y)Z \rhd_w Y[Z/x]$.

Proof. By induction on the structure of Y:

1. $Y \simeq x$: $([x].x)Z \simeq \mathbf{I}Z \rhd_w Z \simeq x[Z/x]$
2. Y is an atom and $Y \not\simeq x$: $([x].Y)Z \simeq \mathbf{K}YZ \rhd_w Y \simeq Y[Z/x]$
3. $Y \simeq (UV)$:
   - $x \notin \mathrm{FV}(Y)$: $([x].Y)Z \simeq \mathbf{K}YZ \rhd_w Y \simeq Y[Z/x]$
   - $x \notin \mathrm{FV}(U)$ and $V \simeq x$: $([x].Y)Z \simeq UZ \simeq Ux[Z/x]$
   - none of the preceding cases:

$$\begin{aligned} ([x].Y)Z &\simeq \mathbf{S}([x].U)([x].V)Z \\ &\rhd_w ([x].U)Z(([x].V)Z) \\ &\rhd_w (U[Z/x])(V[Z/x]) \qquad \text{(induction hypothesis)} \\ &\simeq Y[Z/x] \end{aligned}$$

QED

Corollary 2.11 (Combinatorial completeness)
Let V be a term with $\{x_1, \ldots, x_n\} \subseteq \mathrm{FV}(V)$. Then there exists a term U without occurrences of $x_1, \ldots, x_n$ such that $UX_1 \ldots X_n \rhd_w V[X_1/x_1] \ldots [X_n/x_n]$.

Proof. Choose $U :\simeq [x_1]. \dots [x_n].V$. QED

Remark. Thus every combinator U that is given by a "new" contraction

$$UX_1 \dots X_n \rhd_w W$$

where W is constructed from $X_1, \dots, X_n$ only, can be defined in CL by a variable-free term.
Using only **S** and **K** we can thus express "all possible" combinators.

Lemma 2.12

1. *For CL-terms X we have: $(X_\lambda)_{CL} \simeq X$.*
2. *For λ-terms M we have: $(M_{CL})_\lambda =_{\beta\eta} M$.*

Proof.

1. By induction on the structure of X:
   - $(x_\lambda)_{\mathrm{CL}} \simeq x_{\mathrm{CL}} \simeq x$
   - $(\mathbf{K}_\lambda)_{\mathrm{CL}} \simeq (\lambda xy.x)_{\mathrm{CL}} \simeq [x].([y].x) \simeq [x].\mathbf{K}x \simeq \mathbf{K}$
   - $(\mathbf{S}_\lambda)_{\mathrm{CL}} \simeq (\lambda xyz.xz(yz))_{\mathrm{CL}}$
     $\simeq [x].([y].([z].xz(yz)))$
     $\simeq [x].([y].\mathbf{S}([z].xz)([z].yz))$
     $\simeq [x].([y].\mathbf{S}xy)$
     $\simeq [x].\mathbf{S}x$
     $\simeq \mathbf{S}$
   - $((UV)_\lambda)_{\mathrm{CL}} \simeq (U_\lambda V_\lambda)_{\mathrm{CL}} \simeq (U_\lambda)_{\mathrm{CL}}(V_\lambda)_{\mathrm{CL}} \simeq UV$ (the latter by i.h.)
2. By induction on the structure of M:
   - $M \simeq x$: $(x_{\mathrm{CL}})_\lambda \simeq x_\lambda \simeq x$
   - $M \simeq (PQ)$:
     $((PQ)_{\mathrm{CL}})_\lambda \simeq (P_{\mathrm{CL}}Q_{\mathrm{CL}})_\lambda \simeq (P_{\mathrm{CL}})_\lambda(Q_{\mathrm{CL}})_\lambda =_{\beta\eta} PQ$ (the latter by i.h.)
   - $M \simeq (\lambda x.M')$: We have to show $((\lambda x.M')_{\mathrm{CL}})_\lambda \simeq ([x].(M')_{\mathrm{CL}})_\lambda =_{\beta\eta} \lambda x.M'$.
     Induction on the structure of M':
     - $([x].x_{\mathrm{CL}})_\lambda \simeq \mathbf{I}_\lambda \simeq \mathbf{S}_\lambda\mathbf{K}_\lambda\mathbf{K}_\lambda =_\beta \lambda x.x$
     - if $x \notin \mathrm{FV}(P_{\mathrm{CL}})$:

$$\begin{aligned}
([x].P_{\mathrm{CL}})_\lambda &\simeq (\mathbf{K}P_{\mathrm{CL}})_\lambda \\
&\simeq \mathbf{K}_\lambda(P_{\mathrm{CL}})_\lambda \\
&\simeq (\lambda xy.x)(P_{\mathrm{CL}})_\lambda \qquad \text{where } y \text{ new} \\
&=_\beta \lambda y.(P_{\mathrm{CL}})_\lambda \\
&=_{\beta\eta} \lambda y.P \qquad \text{(induction hypothesis)}
\end{aligned}$$

- if $x \notin \mathrm{FV}(P_{\mathrm{CL}})$ and $Q_{\mathrm{CL}} \simeq x$:

$$\begin{aligned}
([x].(PQ)_{\mathrm{CL}})_\lambda &\simeq ([x].(P_{\mathrm{CL}}Q_{\mathrm{CL}}))_\lambda \\
&\simeq ([x].(P_{\mathrm{CL}}x))_\lambda \\
&\simeq (P_{\mathrm{CL}})_\lambda \\
&=_{\beta\eta} P \qquad \text{(induction hypothesis)} \\
&=_\eta \lambda x.(Px) \\
&\simeq \lambda x.(PQ)
\end{aligned}$$

- otherwise: $$\begin{aligned}
([x].(PQ)_{\mathrm{CL}})_\lambda &\simeq (\mathbf{S}([x].P_{\mathrm{CL}})([x].Q_{\mathrm{CL}}))_\lambda \\
&\simeq \mathbf{S}_\lambda([x].P_{\mathrm{CL}})_\lambda([x].Q_{\mathrm{CL}})_\lambda \\
&=_{\beta\eta} \mathbf{S}_\lambda(\lambda x.P)(\lambda x.Q) \qquad \text{(induction hypothesis)} \\
&\simeq (\lambda uvy.uy(vy))(\lambda x.P)(\lambda x.Q) \\
&=_\beta \lambda y.(\lambda x.P)y((\lambda x.Q)y) \\
&=_\beta \lambda x.(PQ)
\end{aligned}$$

QED

Corollary 2.13 *By Lemma 2.8 we thus have:* $M_{CL} =_w N_{CL} \implies M =_{\beta\eta} N$.

Remark. However, we do *not* have $(M_{\mathrm{CL}})_\lambda =_\beta M$.
A counterexample is $((\lambda x.yx)_{\mathrm{CL}})_\lambda \simeq ([x].yx)_\lambda \simeq y_\lambda \simeq y \neq_\beta \lambda x.yx$.

Differences between λ-calculus and CL

None of the following statements holds:

$$\begin{array}{ll}
M_{\mathrm{CL}} \rhd_w N_{\mathrm{CL}} \implies M \rhd_\beta N & \qquad M_{\mathrm{CL}} \rhd_w N_{\mathrm{CL}} \impliedby M \rhd_\beta N \\
M_{\mathrm{CL}} =_w N_{\mathrm{CL}} \implies M =_\beta N & \qquad M_{\mathrm{CL}} =_w N_{\mathrm{CL}} \impliedby M =_\beta N
\end{array}$$

Thus β-reducibility of a λ-term M to a λ-term N *cannot* be shown by using translations ($\rightsquigarrow$) and weak reducibility as follows: $M \rightsquigarrow M_{\mathrm{CL}} \rhd_w N_{\mathrm{CL}} \rightsquigarrow N$.
The problem with "$\Longrightarrow$" is that

$$(\eta) \quad [x].Xx = X, \text{ if } x \notin \mathrm{FV}(X)$$

holds in CLw, since we have

$$(\lambda x.Mx)_{\mathrm{CL}} \simeq [x].M_{\mathrm{CL}}x \simeq M_{\mathrm{CL}}, \text{ if } x \notin \mathrm{FV}(M).$$

The problem with "$\Longleftarrow$" is that

$$(\xi)\ \frac{X = X'}{[x].X = [x].X'}$$

does *not* hold in CLw, since we have

$$\begin{aligned}
[x].\mathbf{S}xyz &\doteq \mathbf{S}([x].\mathbf{S}xy)([x].z) \\
&\doteq \mathbf{S}(\mathbf{S}([x].\mathbf{S}x)([x].y))(\mathbf{K}z) \\
&\doteq \mathbf{S}(\mathbf{SS}(\mathbf{K}y))(\mathbf{K}z)
\end{aligned}$$

and

$$\begin{aligned}
[x].xz(yz) &\doteq \mathbf{S}([x].xz)([x].yz) \\
&\doteq \mathbf{S}(\mathbf{S}([x].x)([x].z))(\mathbf{K}(yz)) \\
&\doteq \mathbf{S}(\mathbf{SI}(\mathbf{K}z))(\mathbf{K}(yz))
\end{aligned}$$

Hence $\mathbf{S}xyz =_w xz(yz)$, but *not* $[x].\mathbf{S}xyz =_w [x].xz(yz)$.

Extensionality

Adding (ξ) to CLw yields *full* extensionality (ζ):

$$\dfrac{(\tau)\,\dfrac{(\sigma)\,\dfrac{(\eta)\,\dfrac{}{[x].Xx = X}}{X = [x].Xx} \qquad (\xi)\,\dfrac{Xx = Yx}{[x].Xx = [x].Yx}}{X = [x].Yx} \qquad (\eta)\,\dfrac{}{[x].Yx = Y}}{X = Y}\,(\tau)$$

In $\lambda\beta$: (ξ) holds eo ipso, while adding (η) yields extensionality.
In CLw: (η) holds eo ipso, while adding (ξ) yields extensionality.
This shows how the two systems differ.

Strong reduction

We define *strong reduction* $\succ\!\!-$ by extending CL$w_{\rhd}$ by *strong reduction*

$$(\xi)\,\frac{X \succ\!\!- Y}{[x].X \succ\!\!- [x].Y}$$

Then we have for λ-terms M, N:

$$M \rhd_{\beta\eta} N \implies M_{\mathrm{CL}} \succ\!\!- N_{\mathrm{CL}}$$

The converse direction $M_{\mathrm{CL}} \succ\!\!- N_{\mathrm{CL}} \implies M \rhd_{\beta\eta} N$ does, however, *not* hold, since $X \succ\!\!- Y$ does not imply $X_\lambda \rhd_{\beta\eta} Y_\lambda$.
We only have

$$M_{\mathrm{CL}} \succ\!\!- N_{\mathrm{CL}} \implies M =_{\beta\eta} N$$

Now η-conversion is an instance of (ρ) and holds in arbitrary directions. For λ-terms

M, N we therefore have:

$$M =_{\beta\eta} N \iff M_{\text{CL}} \succ\!\!\prec N_{\text{CL}}$$

where $\succ\!\!\prec$ is the symmetric closure of $\succ$. Then the following holds for CL-terms X, Y:

$$X \succ\!\!\prec Y \iff X_\lambda =_{\beta\eta} Y_\lambda.$$

Weakening of $[x].Y$

In order to be able to represent β-equality (instead of only $\beta\eta$-equality) in CLw, one can weaken the definition of $[x].Y$ in such a way that (η) does no longer hold automatically; for example by removing clause

3. $[x].Ux :\simeq U$, if $x \notin \text{FV}(U)$

from the definition of $[x].Y$.
However, even in this case (ξ) does *not* hold, since

$$[x].\mathbf{S}xyz \simeq \mathbf{S}(\mathbf{S}(\mathbf{S}(\mathbf{KS})\mathbf{I})(\mathbf{K}y))(\mathbf{K}z)$$

and (as before)

$$[x].xz(yz) \simeq \mathbf{S}(\mathbf{SI}(\mathbf{K}z))(\mathbf{K}(yz))$$

Again, we do *not* have $[x].\mathbf{S}xyz =_w [x].xz(yz)$, while $\mathbf{S}xyz =_w xz(yz)$ does hold.
However, using the modified definition of $[x].Y$ one can show

$$\lambda\beta \vdash M = N \iff (\text{CL}w + \otimes) \vdash M_{\text{CL}} = N_{\text{CL}}$$
$$(\text{CL}w + \otimes) \vdash X = Y \iff \lambda\beta \vdash X_\lambda = Y_\lambda$$

where $\otimes$ is an extension of CLw by a certain rule schema or by a certain finite set of axioms (cp. the formal theory CLζ_β in Hindley & Seldin (2008), Ch. 9).

Soundness

For the modified definition of $[x].Y$ we do *not* have $(\mathbf{S}_\lambda)_{\text{CL}} =_w \mathbf{S}$. However, by a further modification it is possible to obtain for $=_\beta$ the following (instead of Lemma 2.12 for $=_{\beta\eta}$):

1. *For CL-terms X: $(X_\lambda)_{CL} \simeq X$.*
2. *For λ-terms M: $(M_{CL})_\lambda =_\beta M$.*

The evaluation in CLw of translated λ-terms is then sound. This fact is often used in functional programming.

Completeness

Regarding completeness the following holds: We consider an extended language where additional functions can be evaluated (so-called δ-rules). The corresponding reductions are $\rhd_{1\beta\delta}$, $\rhd_{\beta\delta}$ and $\rhd_{1l\beta\delta}$ (l for *leftmost*), resp. $\rhd_{1w\delta}$ and $\rhd_{w\delta}$. We then have:

> If $M \rhd_{1l\beta\delta} N$, where M is closed and does not have the form $[x].P$, then $M_{\mathrm{CL}} \rhd_{1w\delta} N_{\mathrm{CL}}$.

If a second-order typed system with primitive types Int, Bool and Char is given, in which a fixed-point operator Υ exists and where M has a primitive type, then the following holds:

$$M \rhd_{l\beta\delta} N \implies M_{\mathrm{CL}} \rhd_{w\delta} N_{\mathrm{CL}}$$

This means that everything that can be found in $\lambda\beta$ can be found in CLw by a translation. This fact is e.g. used in the functional programming language Miranda (see Turner, 1979).

For a constant c with some primitive type (which by definition is in β-normal form) we thus get

$$\begin{aligned}
\lambda\beta\delta \vdash M = c &\implies M =_{\beta\delta} c \\
&\implies M \rhd_{l\beta\delta} c \\
&\implies M_{\mathrm{CL}} \rhd_{w\delta} c \\
&\implies \mathrm{CL}w\delta \vdash M_{\mathrm{CL}} = c
\end{aligned}$$

as well as the converse

$$\begin{aligned}
\mathrm{CL}w\delta \vdash M_{\mathrm{CL}} = c &\implies M_{\mathrm{CL}} =_{w\delta} c \\
&\implies M_{\mathrm{CL}} \rhd_{w\delta} c \\
&\implies \underbrace{(M_{\mathrm{CL}})_{\lambda}}_{=_{\beta} M} \rhd_{\beta\delta} c \\
&\implies \lambda\beta\delta \vdash M = c
\end{aligned}$$

Every computation of a value for M can thus also be done in CLw.

3 The simply typed λ-calculus

There are two variants of typing λ-terms:

1. *Curry-style typing*: Terms are the terms of the untyped theory. Every term has a (possibly empty) set of possible types (*implicit* typing, *type assignment*). *Curry-style typing*
2. *Church-style typing*: Terms have associated types, which are usually unique (*explicit* typing). *Church-style typing*

We consider only Curry-style typing in the following, and moreover in its most simple form that has only so-called *simple* types. The simply typed λ-calculus is called $\lambda{\rightarrow}$. We follow the treatment in Barendregt (1992). (Since we consider only the simply typed λ-calculus, we omit the specifier "simply".)

Remark. For $\lambda{\rightarrow}$ strong normalisability holds. Hence not all recursive functions are definable in $\lambda{\rightarrow}$; the partial recursive functions are not $\lambda{\rightarrow}$-definable at all, but there are also total recursive functions which are not definable.
We define the function F as follows (for some adequate enumeration of typed terms):

$$F(n,m) = \begin{cases} k & \text{iff the } n\text{-th typed term applied to the} \\ & \text{argument } \underline{m} \text{ has the } \beta\text{-normal form } \underline{k}, \\ 0 & \text{otherwise.} \end{cases}$$

But then the (total) function $g(n) := F(n,n) + 1$ cannot be definable in $\lambda{\rightarrow}$: Let g be defined in $\lambda{\rightarrow}$ by the p-th typed term. Then $g(p) = F(p,p)$; however, by definition $g(p) = F(p,p) + 1$. Contradiction.

3.1 Implicit typing

Remark. To avoid unnecessary complications we exclude certain kinds of λ-terms, namely

– terms in which a variable occurs free as well as bound,

– (sub)terms of the form $\lambda x.M$, in which λx occurs also in M.

(That is, terms like $(x(\lambda x.(\lambda x.x)))$, for example, are not considered.) Note that this is not an essential restriction of the expressive power of our language.

Definition 3.1 The set of *types* of $\lambda{\rightarrow}$ is defined as follows: *types*

1. *Type variables* $\alpha, \beta, \gamma, \delta, \alpha_1, \alpha_2, \ldots$ are types. *type variables*
2. If σ and τ are types, then $(\sigma \rightarrow \tau)$ is a type (also called *function type*). *function type*

It is $\sigma_1 \rightarrow \sigma_2 \rightarrow \cdots \rightarrow \sigma_{n-1} \rightarrow \sigma_n$ an abbreviation of $\sigma_1 \rightarrow (\sigma_2 \rightarrow (\cdots(\sigma_{n-1} \rightarrow \sigma_n)\cdots))$; that is, we use *association to the right* for function types.

– A *judgement* has the form $M:\sigma$ for a λ-term M and a type σ. *judgement*

The term M is called the *subject* of the judgement. (The type σ is then also called the *predicate*. One says "M has type σ" or something like that.) *subject*

– A *declaration* is a judgement whose subject is a term variable. *declaration*

– A *basis* Γ is a finite set of declarations whose subjects are pairwise distinct. *basis*

– A *sequent* has the form $\Gamma \vdash M:\sigma$ for a basis Γ and a judgement $M:\sigma$. *sequent*

Definition 3.2 Sequents $\Gamma \vdash M:\sigma$ expressing that the judgement $M:\sigma$ holds in basis Γ can be derived in the *calculus* $\lambda{\to}$, which is given by the axiom scheme *calculus* $\lambda{\to}$

$$(\mathrm{Id})\quad \Gamma, x:\sigma \vdash x:\sigma$$

together with the following $\to$-introduction and $\to$-elimination rules:

$$({\to}\mathrm{I})\ \frac{\Gamma, x:\sigma \vdash M:\tau}{\Gamma \vdash (\lambda x.M):\sigma\to\tau} \qquad\qquad ({\to}\mathrm{E})\ \frac{\Gamma \vdash M:\sigma\to\tau \qquad \Gamma \vdash N:\sigma}{\Gamma \vdash (MN):\tau}$$

If $\Gamma \vdash M:\sigma$ is *derivable* in $\lambda{\to}$, then we write $\Gamma \vdash_{\lambda\to} M:\sigma$ or $\Gamma \vdash M:\sigma$. (Thus we often identify sequents with the assertion of their derivability. What is meant in each case should be clear from the context.) *derivable*

Examples.

1. It is $\vdash_{\lambda\to} \lambda xy.x:\sigma\to\tau\to\sigma$:

$$({\to}\mathrm{I})\ \dfrac{({\to}\mathrm{I})\ \dfrac{(\mathrm{Id})\ \dfrac{}{x:\sigma, y:\tau \vdash x:\sigma}}{x:\sigma \vdash \lambda y.x:\tau\to\sigma}}{\vdash \lambda x.\lambda y.x:\sigma\to\tau\to\sigma}$$

   That is, the combinator **K** has type $\sigma\to\tau\to\sigma$.

2. It is $\vdash_{\lambda\to} \lambda xyz.xz(yz):(\sigma\to\tau\to\tau')\to(\sigma\to\tau)\to\sigma\to\tau'$:

   Let $\Gamma = \{x:\sigma\to\tau\to\tau', y:\sigma\to\tau, z:\sigma\}$.

$$({\to}\mathrm{I})\ \dfrac{({\to}\mathrm{I})\ \dfrac{({\to}\mathrm{I})\ \dfrac{({\to}\mathrm{E})\ \dfrac{({\to}\mathrm{E})\ \dfrac{(\mathrm{Id})\ \dfrac{}{\Gamma \vdash x:\sigma\to\tau\to\tau'} \quad (\mathrm{Id})\ \dfrac{}{\Gamma \vdash z:\sigma}}{\Gamma \vdash xz:\tau\to\tau'} \quad ({\to}\mathrm{E})\ \dfrac{(\mathrm{Id})\ \dfrac{}{\Gamma \vdash y:\sigma\to\tau} \quad (\mathrm{Id})\ \dfrac{}{\Gamma \vdash z:\sigma}}{\Gamma \vdash yz:\tau}}{x:\sigma\to\tau\to\tau', y:\sigma\to\tau, z:\sigma \vdash xz(yz):\tau'}}{x:\sigma\to\tau\to\tau', y:\sigma\to\tau \vdash \lambda z.xz(yz):\sigma\to\tau'}}{x:\sigma\to\tau\to\tau' \vdash \lambda yz.xz(yz):(\sigma\to\tau)\to\sigma\to\tau'}}{\vdash \lambda xyz.xz(yz):(\sigma\to\tau\to\tau')\to(\sigma\to\tau)\to\sigma\to\tau'}$$

   That is, the combinator **S** has type $(\sigma\to\tau\to\tau')\to(\sigma\to\tau)\to\sigma\to\tau'$.

Remark. One can add constants. Corresponding declarations for constants are then added to every basis. An example is the fixed-point combinator $\mathbf{Y}: (\sigma \to \sigma) \to \sigma$ for all types σ in the programming language ML.

Definition 3.3

- A closed term M is called *typable*, if $\vdash M : \sigma$ for a type σ. *typable*
- A term M with free variables $x_1, \ldots, x_n$ is called *typable*, if $\Gamma \vdash M : \sigma$ for a type σ, where $\Gamma = \{x_1 : \sigma_1, \ldots, x_n : \sigma_n\}$ for certain types $\sigma_1, \ldots, \sigma_n$.
- Let $\Gamma = \{x_1 : \sigma_1, \ldots, x_n : \sigma_n\}$ be a basis. Then let $\Gamma(x_i) := \sigma_i$.
- Let V be a set of variables and Γ a basis. Then the *restriction* of Γ to V is defined by *restriction*

  $$\Gamma|_V := \{x : \sigma \mid x \in V \text{ and } \Gamma(x) = \sigma\}$$

- The *domain* of Γ is $\mathrm{dom}(\Gamma) := \{x_1, \ldots, x_n\}$. *domain*
- *Type substitution*: $\sigma[\tau/\alpha]$ signifies the simultaneous replacement of all occurrences of the type variable α in type σ by type τ. (See also Definition 3.11.) *type substitution*

  For a basis $\Gamma = \{x_1 : \sigma_1, \ldots, x_n : \sigma_n\}$ the expression $\Gamma[\tau/\alpha]$ signifies the result of type substitutions $\sigma_i[\tau/\alpha]$, for $1 \leq i \leq n$, in Γ.

Remark. For sequents $\Gamma \vdash M : \sigma$ one refers to $M : \sigma$ also as a *hypothetical judgement*, since it depends on the basis Γ; for sequents $\vdash M : \sigma$ one refers to $M : \sigma$ as a *categorical judgement*.

Example. The term xx is *not* typable. Consider

$$(\to\mathrm{E})\ \dfrac{(\mathrm{Id})\ \dfrac{}{x : \sigma \to \tau, x : \sigma \vdash x : \sigma \to \tau}\ \lightning \qquad (\mathrm{Id})\ \dfrac{}{x : \sigma \to \tau, x : \sigma \vdash x : \sigma}\ \lightning}{x : \sigma \to \tau, x : \sigma \vdash xx : \tau}\ \lightning$$

This is *not* a correct derivation, since the subjects of the basis $\{x : \sigma \to \tau, x : \sigma\}$ are not pairwise distinct.
Consequently, $\lambda x.xx$ is not typable, since any corresponding derivation would have to begin as shown above. Hence, for example $\mathbf{\Omega}$ has no type as well.

Lemma 3.4

1. *If* $\Gamma \subseteq \Gamma'$ *for bases* Γ *and* Γ', *then* $(\Gamma \vdash M : \sigma \implies \Gamma' \vdash M : \sigma)$. *(monotony)*
2. *If* $\Gamma \vdash M : \sigma$, *then* $\mathrm{FV}(M) \subseteq \mathrm{dom}(\Gamma)$.
3. *If* $\Gamma \vdash M : \sigma$, *then* $\Gamma|_{\mathrm{FV}(M)} \vdash M : \sigma$.
4. *If* $\Gamma \vdash x : \sigma$, *then* $(x : \sigma) \in \Gamma$.
5. *If* $\Gamma \vdash (\lambda x.M) : \sigma$, *then* $\sigma \doteq \sigma_1 \to \sigma_2$ *for some* σ_1, σ_2 *and* $\Gamma, x : \sigma_1 \vdash M : \sigma_2$.
6. *If* $\Gamma \vdash MN : \sigma$, *then* $\Gamma \vdash M : \tau \to \sigma$ *and* $\Gamma \vdash N : \tau$ *for a* τ.
7. *If* $\Gamma \vdash M : \sigma$ *and* M' *is subterm of* M, *then* $\Gamma' \vdash M' : \sigma'$ *for some* Γ', σ'.
8. *If* $\Gamma \vdash M : \sigma$, *then* $\Gamma[\tau/\alpha] \vdash M : \sigma[\tau/\alpha]$.

9. *If* $\Gamma, x:\sigma \vdash M:\tau$ *and* $\Gamma \vdash N:\sigma$, *then* $\Gamma \vdash M[N/x]:\tau$.

10. *If* $\Gamma \vdash M:\sigma$ *and* $M \rhd_\beta M'$, *then* $\Gamma \vdash M':\sigma$. *(subject reduction)*

Remarks.

1. The latter says that types are invariant under subject reduction.
2. However, invariance under *subject expansion* $\lhd_\beta$ does *not* hold; that is, if $M \lhd_\beta M'$ and $\Gamma \vdash M:\sigma$, then in general $\Gamma \vdash M':\sigma$ does *not* hold.

   Example: Although $\mathbf{I} \lhd_\beta \mathbf{KI}(\lambda x.xx)$ and $\vdash \mathbf{I}:\sigma\to\sigma$, we have $\nvdash \mathbf{KI}(\lambda x.xx):\sigma\to\sigma$.
3. Lemma 3.4 (4)-(6) is also referred to as "generation lemma" in the literature.

In the following we show that all typable terms are strongly normalising. (Weak normalisability was already shown by Turing; strong normalisability goes back to Tait, 1967.) The converse does not hold: For example, the term $\lambda x.xx$ in β-nf is not typable.

Definition 3.5

- Let SN be the set of strongly normalisable λ-terms. SN
- For sets A, B of λ-terms let $A\to B := \{M \mid \text{for all } N \in A \text{ it is } MN \in B\}$.
- The *interpretation* of types is defined inductively as follows: *interpretation (of types)*

$$\llbracket\alpha\rrbracket := \text{SN, for all type variables } \alpha;$$
$$\llbracket\sigma\to\tau\rrbracket := \llbracket\sigma\rrbracket\to\llbracket\tau\rrbracket.$$

Remark. The interpretation of a function type is a set of λ-terms having the desired transition property $A\to B$ w.r.t. the given domain A and co-domain B.

Definition 3.6 A set A of terms is called *saturated*, if the following holds (for $n \geq 0$): *saturated*

(a) $A \subseteq \text{SN}$,

(b) $xR_1\ldots R_n \in A$, if x is a term variable and $R_1,\ldots,R_n \in \text{SN}$,

(c) $(\lambda x.M)NR_1\ldots R_n \in A$, if $(M[N/x])R_1\ldots R_n \in A$ for $N, R_1,\ldots,R_n \in \text{SN}$.

$\text{SAT} := \{A \mid A \text{ saturated}\}$. SAT

Lemma 3.7 *For every type* σ *of* $\lambda\to$ *it holds that* $\llbracket\sigma\rrbracket$ *is saturated.*

Proof.

1. σ is a type variable. We have to show: SN is saturated.
   (a) $\text{SN} \subseteq \text{SN}$.
   (b) $xR_1\ldots R_n \in \text{SN}$, if $R_1,\ldots,R_n \in \text{SN}$.
   (c) Let $(M[N/x])R_1\ldots R_n \in \text{SN}$ with $N, R_1,\ldots,R_n \in \text{SN}$.
   Then also $M \in \text{SN}$, since otherwise $(M[N/x])R_1\ldots R_n$ could not be strongly

normalisable. We consider $(\lambda x.M)NR_1 \dots R_n$. Then every reduction series has the following form, where $M \rhd_\beta M'$, $N \rhd_\beta N'$ and $R_i \rhd_\beta R_i'$ for $1 \le i \le n$:

$$\begin{aligned}(\lambda x.M)NR_1 \dots R_n &\rhd_\beta (\lambda x.M')N'R_1' \dots R_n' \\ &\rhd_{1\beta} M'[N'/x]R_1' \dots R_n' \\ &\rhd_\beta \dots\end{aligned}$$

We know that if $S \rhd_\beta T$ and $P \rhd_\beta Q$, then $P[S/x] \rhd_\beta Q[T/x]$. Thus:

$$(M[N/x])R_1 \dots R_n \rhd_\beta (M'[N'/x])R_1' \dots R_n' \rhd_\beta \cdots$$

Since this series terminates, the first series terminates as well.

Therefore $(\lambda x.M)NR_1 \dots R_n$ is strongly normalisable.

2. σ is a function type $\sigma_1 \to \sigma_2$. Let $A = [\![\sigma_1]\!]$ and $B = [\![\sigma_2]\!]$.

   If A and B are saturated, then so is $A \to B$:

   (a) Let $M \in A \to B$. By Definition 3.6 (b) we have $x \in A$ for all variables x (case $n = 0$). Thus $Mx \in B$. Since Mx is strongly normalisable, also M is strongly normalisable. Therefore $A \to B \subseteq \mathrm{SN}$.

   (b) Let $M \in A$. Then $M \in \mathrm{SN}$. Hence $xR_1 \dots R_n M \in B$ for $R_i \in \mathrm{SN}$. Thus also $xR_1 \dots R_n \in A \to B$.

   (c) Let $M \in A$. Then $M \in \mathrm{SN}$.
   Then $(\lambda x.P)NR_1 \dots R_n M \in B$, if $(P[N/x])R_1 \dots R_n M \in B$.
   Then $(\lambda x.P)NR_1 \dots R_n \in A \to B$, if $(P[N/x])R_1 \dots R_n \in A \to B$. QED

Definition 3.8

– A *valuation* is a function $\rho\colon$ variables $\to \lambda$-terms. *valuation*

– Let ρ be a valuation. Then the *interpretation* of a term M under ρ is: *interpretation (of terms)*

$$[\![M]\!]_\rho := M[\rho(x_1)/x_1, \dots, \rho(x_n)/x_n]$$

where $\{x_1, \dots, x_n\}$ is the set of *all* free variables in M.

– A valuation ρ *satisfies* $M : \sigma$, if $[\![M]\!]_\rho \in [\![\sigma]\!]$. Notation: $\rho \vDash M : \sigma$. *satisfiability*

– A valuation ρ *satisfies* a basis Γ, if $\rho \vDash x : \sigma$ for all $(x : \sigma) \in \Gamma$.

  Notation: $\rho \vDash \Gamma$.

– A basis Γ *satisfies* $M : \sigma$, if:

  For all valuations ρ: If $\rho \vDash \Gamma$, then $\rho \vDash M : \sigma$.

  Notation: $\Gamma \vDash M : \sigma$.

Lemma 3.9 (Soundness) *If* $\Gamma \vdash M : \sigma$, *then* $\Gamma \models M : \sigma$.

Proof. By induction on the structure of the derivation of $\Gamma \vdash M : \sigma$.

Case (Id): Let $\Gamma = \Gamma' \cup \{x : \sigma\}$ and $M \doteq x$. It is $\Gamma', x : \sigma \vdash x : \sigma$.

Suppose $\rho \models \Gamma' \cup \{x : \sigma\}$, then $\rho \models x : \sigma$ as a special case. Hence $\Gamma', x : \sigma \models x : \sigma$.

Case ($\to$I): Let $M \doteq \lambda x.N$. Then $\sigma \doteq \sigma_1 \to \sigma_2$ and $\Gamma, x : \sigma_1 \vdash N : \sigma_2$.

By the induction hypothesis we have $\Gamma, x : \sigma_1 \models N : \sigma_2$.

That is, if $\rho \models \Gamma$ and $\rho(x) \in [\![\sigma_1]\!]$, then $[\![N]\!]_\rho \in [\![\sigma_2]\!]$.

Then $[\![(\lambda x.N)x]\!]_\rho \in [\![\sigma_2]\!]$, since $[\![\sigma_2]\!]$ is saturated.

That is, if $\rho \models \Gamma$ and $\rho(x) = Q \in [\![\sigma_1]\!]$, then $[\![\lambda x.N]\!]_\rho Q = [\![(\lambda x.N)x]\!]_\rho \in [\![\sigma_2]\!]$.

Hence $[\![\lambda x.N]\!]_\rho \in [\![\sigma_1 \to \sigma_2]\!]$.

Case ($\to$E): Let $M \doteq PQ$. Then $\Gamma \vdash P : \tau \to \sigma$ and $\Gamma \vdash Q : \tau$.

By the induction hypothesis we have $\Gamma \models P : \tau \to \sigma$ and $\Gamma \models Q : \tau$.

That is, if $\rho \models \Gamma$, then $[\![P]\!]_\rho \in [\![\tau \to \sigma]\!]$ and $[\![Q]\!]_\rho \in [\![\tau]\!]$.

Then $[\![PQ]\!]_\rho \in [\![\sigma]\!]$ by Definition of $[\![\ ]\!]$, since $[\![PQ]\!]_\rho = [\![P]\!]_\rho [\![Q]\!]_\rho$. QED

Theorem 3.10 *If* $\Gamma \vdash M : \sigma$, *then* M *is strongly normalisable.*

Proof. Suppose $\Gamma \vdash M : \sigma$. Then by soundness $\Gamma \models M : \sigma$. That is, it holds for all valuations ρ: If $\rho \models \Gamma$, then $\rho \models M : \sigma$.
Let $\rho_{id}(x) := x$ for every free variable x in M. Then $\rho_{id} \models \Gamma$, since $x \in [\![\sigma]\!]$ for every x (since $[\![\sigma]\!]$ is saturated). Therefore $\rho_{id} \models M : \sigma$, i.e. $M \doteq [\![M]\!]_{\rho_{id}} \in [\![\sigma]\!]$. Since $[\![\sigma]\!]$ is saturated, M is strongly normalisable. QED

3.2 The type assignment algorithm

In the following, we present an algorithm that assigns types to terms. If a given term is typable, the algorithm assigns a type; otherwise the algorithm outputs fail. This immediately solves the two most important decidability problems concerning implicit type assignment:

1. Given M and σ, does $\vdash M : \sigma$ hold? (*type checking*) *type checking*

2. Given M, is there a σ with $\vdash M : \sigma$? (*typability*) *typability*

A further decidability problem is the following:

3. Given σ, is there an M with $\vdash M : \sigma$? (*type inhabitation*) *type inhabitation*

This is solved by the Curry-Howard isomorphism (see Section 3.3).

We begin with some preliminaries on substitution and unification of type variables and types.

Definition 3.11

- A *substitution (in types)* is a function s : type variables $\rightarrow$ types, where $s(\alpha) \neq \alpha$ only for finitely many α. *substitution (in types)*

  We write s also as $\{\alpha_1 = \sigma_1, \ldots, \alpha_n = \sigma_n\}$, if $s(\alpha_i) = \sigma_i$.

- Obviously, s determines a function $\bar{s}$ from types to types:

$$\bar{s}(\alpha) :\simeq s(\alpha)$$
$$\bar{s}(\sigma \rightarrow \tau) :\simeq \bar{s}(\sigma) \rightarrow \bar{s}(\tau)$$

  We identify s and $\bar{s}$ and write $s(\sigma)$ or $\sigma[\sigma_1/\alpha_1, \ldots, \sigma_n/\alpha_n]$.

- For a basis $\Gamma = \{x_1 : \sigma_1, \ldots, x_n : \sigma_n\}$ let $s(\Gamma) = \{x_1 : s(\sigma_1), \ldots, x_n : s(\sigma_n)\}$.

- For substitutions s_1 and s_2 the composition $s_1 \circ s_2$ (short: $s_1 s_2$) is defined naturally.

  Correspondingly, $s_1 \circ s_2(\sigma) \simeq s_1(s_2(\sigma))$.

- A *unifier* for σ and τ is an s with $s(\sigma) \simeq s(\tau)$. *unifier*

  A unifier for a set of equations $E = \{\sigma_1 = \tau_1, \ldots, \sigma_n = \tau_n\}$ is an s with $s(\sigma_i) \simeq s(\tau_i)$ for all i with $1 \leq i \leq n$.

- A *most general unifier* (*mgu*) for σ and τ (w.r.t. E) is a unifier s such that for every other unifier s' for σ and τ (w.r.t. E) the following holds: $s' = s_1 \circ s$ for a substitution s_1. *most general unifier*

  We write $s = \text{mgu}(\sigma, \tau)$, respectively $s = \text{mgu}(E)$.

- It is τ a *variant* of σ, if there are s_1 and s_2 with $s_1(\tau) \simeq \sigma$ and $s_2(\sigma) \simeq \tau$. *variant*

Examples.

1. $\alpha \rightarrow (\beta \rightarrow \alpha)$ and $\alpha \rightarrow (\beta \rightarrow \beta)$ have $\{\alpha/\beta\}$ as mgu.
2. $\beta \rightarrow (\alpha \rightarrow \beta)$ and $(\gamma \rightarrow \gamma) \rightarrow \delta$ have $\{\gamma \rightarrow \gamma/\beta, \alpha \rightarrow (\gamma \rightarrow \gamma)/\delta\}$ as mgu.

Remark. Two mgus w.r.t. the same set are always variants of each other. In this sense mgus are unique.

Theorem 3.12 *There exists an algorithm (called "unification algorithm") which yields for every system of equations E an mgu for E, if E is unifiable, and outputs* fail, *if E is not unifiable.* *unification algorithm*

Proof. See logic programming. One algorithm (due to Herbrand) consists of transformation rules for systems of equations $E = \{\sigma_1 = \tau_1, \ldots, \sigma_n = \tau_n\}$ where $E_1 \dot{\cup} E_2$ denotes the union of disjoint sets E_1 and E_2:

$$(id)\ \frac{E \dot{\cup} \{\sigma = \sigma\}}{E}$$

$$(sym)\ \frac{E \dot{\cup} \{\sigma = \alpha\}}{E \cup \{\alpha = \sigma\}} \text{ if } \sigma \text{ is not a type variable}$$

$$(\mathit{fail})\ \frac{E \mathbin{\dot\cup} \{\alpha = \sigma\}}{\mathsf{fail}} \text{ if } \alpha \text{ occurs in } \sigma$$

$$(\mathit{subst})\ \frac{E \mathbin{\dot\cup} \{\alpha = \sigma\}}{E[\sigma/\alpha] \cup \{\alpha = \sigma\}} \text{ if } \alpha \text{ does not occur in } \sigma \text{ and } \alpha \text{ occurs in } E$$

$$(\mathit{func})\ \frac{E \mathbin{\dot\cup} \{\tau_1 \to \tau_2 = \sigma_1 \to \sigma_2\}}{E \cup \{\tau_1 = \sigma_1, \tau_2 = \sigma_2\}}$$

One can show that the application of these rules to a system of equations always terminates. The result is either fail or $\{\alpha_1 = \sigma_1, \ldots, \alpha_n = \sigma_n\}$, where $\{\alpha_1 = \sigma_1, \ldots, \alpha_n = \sigma_n\}$ is the mgu for E. QED

Remarks.

1. The rule name (*sym*) refers to the symmetry of equations, which is made use of in the rule to shift type variables from right to left (but not the other way around).
2. In applications of (*subst*) we will note the respective substitutions to the right of the rule bar.

We now map sequents $\Gamma \vdash M : \sigma$ to such systems of equations E.

Definition 3.13

For sequents $\Gamma \vdash M : \sigma$ the *associated system of equations* $E(\Gamma \vdash M : \sigma)$ is defined as follows:

associated system of equations

1. $E(\Gamma \vdash x : \sigma) := \{\sigma = \Gamma(x)\}$,
2. $E(\Gamma \vdash \lambda x.M : \sigma) := \{\sigma = \alpha \to \beta\} \cup E(\Gamma, x : \alpha \vdash M : \beta)$ for new type variables α, β,
3. $E(\Gamma \vdash MN : \sigma) := E(\Gamma \vdash M : \alpha \to \sigma) \cup E(\Gamma \vdash N : \alpha)$ for a new type variable α.

Remark. The *type assignment algorithm* comprises two main steps: (1) formulate the associated system of equations $E(\Gamma \vdash M : \sigma)$; (2) apply the unification algorithm.

type assignment algorithm

Examples.

1. $E(x : \alpha \vdash x : \beta) = \{\beta = \alpha\}$
2. $$\begin{aligned} E(\vdash \lambda xy.x : \alpha \to \beta) &= \{\alpha \to \beta = \alpha_1 \to \alpha_2\} \cup E(x : \alpha_1 \vdash \lambda y.x : \alpha_2) \\ &= \{\alpha \to \beta = \alpha_1 \to \alpha_2, \alpha_2 = \alpha_3 \to \alpha_4\} \cup E(x : \alpha_1, y : \alpha_3 \vdash x : \alpha_4) \\ &= \{\alpha \to \beta = \alpha_1 \to \alpha_2, \alpha_2 = \alpha_3 \to \alpha_4, \alpha_4 = \alpha_1\} \end{aligned}$$

   Solution:

$$\begin{array}{rcl} (\mathit{func}) & \dfrac{\{\alpha \to \beta = \alpha_1 \to \alpha_2, \alpha_2 = \alpha_3 \to \alpha_4, \alpha_4 = \alpha_1\}}{\{\alpha = \alpha_1, \beta = \alpha_2, \alpha_2 = \alpha_3 \to \alpha_4, \alpha_4 = \alpha_1\}} & \\ (\mathit{subst}) & \dfrac{}{\{\alpha = \alpha_1, \beta = \alpha_3 \to \alpha_4, \alpha_2 = \alpha_3 \to \alpha_4, \alpha_4 = \alpha_1\}} & [\alpha_3 \to \alpha_4/\alpha_2] \\ (\mathit{subst}) & \dfrac{}{\{\alpha = \alpha_1, \beta = \alpha_3 \to \alpha_1, \alpha_2 = \alpha_3 \to \alpha_1, \alpha_4 = \alpha_1\}} & [\alpha_1/\alpha_4] \end{array}$$

   Therefore $\vdash \lambda xy.x : \alpha_1 \to (\alpha_3 \to \alpha_1)$.

Remark: We here started with $E(\vdash \lambda xy.x : \alpha \to \beta)$ instead of $E(\vdash \lambda xy.x : \sigma)$. This is not a problem in this example. However, starting with a type of a specific form might result in not finding solutions that could be found by starting with the unspecific σ. Starting with σ is thus preferable in general.

3. $$\begin{aligned} E(\vdash \lambda x.xx : \sigma) &= \{\sigma = \alpha_1 \to \alpha_2\} \cup E(x : \alpha_1 \vdash xx : \alpha_2) \\ &= \{\sigma = \alpha_1 \to \alpha_2\} \cup E(x : \alpha_1 \vdash x : \alpha_3 \to \alpha_2) \cup E(x : \alpha_1 \vdash x : \alpha_3) \\ &= \{\sigma = \alpha_1 \to \alpha_2, \alpha_3 \to \alpha_2 = \alpha_1, \alpha_1 = \alpha_3\} \end{aligned}$$

   Attempted solution:

$$\cfrac{\cfrac{\cfrac{\cfrac{\{\sigma = \alpha_1 \to \alpha_2, \alpha_3 \to \alpha_2 = \alpha_1, \alpha_1 = \alpha_3\}}{\{\sigma = \alpha_1 \to \alpha_2, \alpha_1 = \alpha_3 \to \alpha_2, \alpha_1 = \alpha_3\}}\,(sym)}{\{\sigma = (\alpha_3 \to \alpha_2) \to \alpha_2, \alpha_1 = \alpha_3 \to \alpha_2, \alpha_1 = \alpha_3\}}\,(subst)\ [\alpha_3 \to \alpha_2/\alpha_1]}{\{\sigma = (\alpha_3 \to \alpha_2) \to \alpha_2, \alpha_3 = \alpha_3 \to \alpha_2, \alpha_1 = \alpha_3\}}\,(subst)\ [\alpha_3/\alpha_1]}{\mathsf{fail}}\,(fail)$$

   The system of equations cannot be solved, since $\alpha_3 = \alpha_3 \to \alpha_2$ blocks termination. Hence, the term $\lambda x.xx$ is not typable.

Lemma 3.14 (Soundness and completeness)

1. *Let s be a solution to $E(\Gamma \vdash M : \sigma)$. Then $s(\Gamma) \vdash M : s(\sigma)$ holds.*
2. *If $s(\Gamma) \vdash M : s(\sigma)$, then the following holds: There exists an s' which interprets the type variables in Γ and σ like s, and s' is a solution to $E(\Gamma \vdash M : \sigma)$. Type variables which are interpreted differently by s and s' can always be chosen from a fixed set of type variables V with $V \cap \mathrm{FV}(\Gamma \cup \{\sigma\}) = \emptyset$.*

Proof.

1. Induction on the structure of M:

   Case $M \doteq x$: The substitution s is a solution to $E(\Gamma \vdash x : \sigma)$, i.e. $s(\sigma) = s(\Gamma(x))$. Hence $x : s(\sigma)$ occurs in Γ. Therefore $s(\Gamma) \vdash x : s(\sigma)$.

   Case $M \doteq \lambda x.P$: If s is a solution to $E(\Gamma \vdash \lambda x.P : \sigma)$, then s is a solution to

$$\{\sigma = \alpha \to \beta\} \cup E(\Gamma, x : \alpha \vdash P : \beta).$$

   By the induction hypothesis $s(\Gamma), x : s(\alpha) \vdash P : s(\beta)$ holds. Therefore

$$s(\Gamma) \vdash \lambda x.P : (s(\alpha) \to s(\beta)) = s(\sigma)$$

   holds.

   Case $M \doteq PQ$: If s is a solution to $E(\Gamma \vdash PQ : \sigma)$, then s is a solution to

$$E(\Gamma \vdash P : \alpha \to \sigma) \quad \text{and} \quad E(\Gamma \vdash Q : \alpha).$$

By the induction hypothesis we have

$$s(\Gamma) \vdash P : s(\alpha \to \sigma) \quad \text{and} \quad s(\Gamma) \vdash Q : s(\alpha).$$

Therefore $s(\Gamma) \vdash PQ : s(\sigma)$ holds.

2. Induction on the structure of M:

   Case $M \doteq x$: It is $s(\Gamma) \vdash x : s(\sigma)$, i.e. $(x : s(\sigma)) \in s(\Gamma)$. Hence $s(\sigma) = s(\Gamma(x))$ holds.

   Thus s itself is a solution to $E(\Gamma \vdash x : \sigma)$.

   Case $M \doteq \lambda x.P$: It is $s(\Gamma) \vdash \lambda x.P : s(\sigma)$, i.e. $s(\sigma) = \sigma_1 \to \sigma_2$ for certain σ_1, σ_2, and it is $s(\Gamma), x : \sigma_1 \vdash P : \sigma_2$.

   Let s' be similar to s, but extended by $\alpha_1 \mapsto \sigma_1$ and $\alpha_2 \mapsto \sigma_2$ with new α_1, α_2. Then $s'(\Gamma), x : \alpha_1 \vdash P : s'(\alpha_2)$.

   Hence s and s' coincide for all type variables occurring in Γ and σ, and s' is a solution to $\{\sigma = \alpha_1 \to \alpha_2\}$.

   By the induction hypothesis there exists a solution s'' to $E(\Gamma, x : \alpha_1 \vdash P : \alpha_2)$ such that s'' coincides with s' for all type variables in $\Gamma, \alpha_1, \alpha_2$.

   Moreover, we can assume that type variables which are interpreted differently by s' and s'' do not occur in σ, i.e. $s'(\sigma) = s''(\sigma)$.

   Hence s'' also solves $\{\sigma = \alpha_1 \to \alpha_2\} \cup E(\Gamma, x : \alpha_1 \vdash P : \alpha_2)$, i.e. $E(\Gamma \vdash \lambda x.P : \sigma)$, and s'' coincides with s for all type variables in Γ and σ.

   Case $M \doteq PQ$: It is $s(\Gamma) \vdash PQ : \sigma$, i.e. $s(\Gamma) \vdash P : \tau \to s(\sigma)$ and $s(\Gamma) \vdash Q : \tau$ for some τ.

   We define s' as s, extended by $\alpha \mapsto \tau$, where α is new. Then s' coincides with s for all type variables in Γ and σ. We thus have $s'(\Gamma) \vdash P : s'(\alpha) \to s'(\sigma)$ and $s'(\Gamma) \vdash Q : s'(\alpha)$.

   By the induction hypothesis there exist solutions s_1'' and s_2'' to $E(\Gamma \vdash P : \alpha \to \sigma)$ and $E(\Gamma \vdash Q : \alpha)$, respectively, which coincide with s' for all type variables in Γ, σ, α.

   Moreover, we can assume that new type variables introduced in the construction of $E(\Gamma \vdash P : \alpha \to \sigma)$ and $E(\Gamma \vdash Q : \alpha)$ are different to each other.

   Then $s_1'' \cup s_2''$ is a solution to $E(\Gamma \vdash P : \alpha \to \sigma) \cup E(\Gamma \vdash Q : \alpha)$, i.e. to $E(\Gamma \vdash PQ : \sigma)$, and it coincides with s on all type variables in Γ and σ. QED

In $\lambda\to$ different types can be assigned to a given typable term. However, all such types are substitution instances of a so-called principal type.

Definition 3.15

- We call σ a *principal type* for a closed term M, if the following holds: If $\vdash M : \sigma$ and $\vdash M : \sigma'$, then there exists a substitution s such that $\sigma' = s(\sigma)$. *principal type*
- We call $\langle \Gamma, \sigma \rangle$ a *principal pair* for M, if the following holds: If $\Gamma \vdash M : \sigma$ and $\Gamma' \vdash M : \sigma'$, then there exists a substitution s such that $\sigma' = s(\sigma)$ and $\Gamma' \supseteq s(\Gamma)$. *principal pair*

Examples.

1. $\alpha \to \alpha$ is a principal type of $\mathbf{I}$.
2. $\langle \{x : (\alpha \to \alpha) \to \beta\}, \beta \rangle$ is a principal pair for $x\mathbf{I}$.

Theorem 3.16 *There exists an algorithm which yields for every closed term M a principal type and for every open (or closed) term M a principal pair, if M is typable at all, and which outputs* fail *otherwise.*

Proof. We consider arbitrary terms M. The result for closed terms is then a special case for the empty basis.
Let $\Gamma_0 := \{x_1 : \alpha_1, \ldots, x_n : \alpha_n\}$ and $\sigma_0 := \beta$, where $x_1, \ldots, x_n$ are the free variables in M. Every mgu-solution to $E(\Gamma_0 \vdash M : \sigma_0)$ is a principal pair for M, if there exists a solution; otherwise we get output fail.

1. $$\begin{aligned} M \text{ has a type } &\iff \Gamma \vdash M : \sigma \text{ for certain } \Gamma, \sigma \\ &\iff s(\Gamma_0) \vdash M : s(\sigma_0) \text{ for a certain } s \\ &\iff E(\Gamma_0 \vdash M : \sigma_0) \text{ is solvable (Lemma 3.14)} \end{aligned}$$

2. Let s be an mgu-solution to $E(\Gamma_0 \vdash M : \sigma_0)$. Then $s(\Gamma_0) \vdash M : s(\sigma_0)$.

   Let now $\Gamma' \vdash M : \sigma'$ and $\widetilde{\Gamma} := \Gamma'|_{\mathrm{FV}(M)}$. Then $\widetilde{\Gamma} \vdash M : \sigma'$.

   Choose s' such that $s'(\Gamma_0) = \widetilde{\Gamma}$, and $s'(\sigma_0) = \sigma'$. Then $s'(\Gamma_0) \vdash M : s'(\sigma_0)$.

   Then by Lemma 3.14 (2) the following holds for some s'' which interprets the type variables in Γ_0 and σ_0 like s': s'' is a solution to $E(\Gamma_0 \vdash M : \sigma_0)$.

   Since s is an mgu, we have $s'' = s_1 \circ s$ for some s_1, i.e. $\sigma' = s''(\sigma_0) = s_1(s(\sigma_0))$.

   Moreover, we have that $\Gamma \supseteq \widetilde{\Gamma}$ with $\widetilde{\Gamma} = s''(\Gamma_0) = s_1(s(\Gamma_0))$.

   Hence $\langle s(\Gamma_0), s(\sigma_0) \rangle$ is a principal pair for M. QED

Theorem 3.17

1. *Typability is decidable.*
2. *Type checking is decidable.*

Proof.

1. Let a closed term M be given. The algorithm of Theorem 3.16 outputs a principal type σ, if M is typable, otherwise fail.
2. To check whether $\vdash M : \sigma'$ for a given σ' we apply the algorithm of Theorem 3.16 to M. If M has a type, then we obtain a type assignment $\vdash M : \sigma$ with principal type σ. Since σ is principal, there must then be a substitution s such that $\sigma' = s(\sigma)$, if σ' is a type of M. Whether there is such a substitution s can be checked by a simple algorithm. QED

Theorem 3.18 *Type inhabitation is decidable.*

Proof. There is a term M with $\vdash M : \sigma$ iff there is a proof of σ (as formula) in positive implication logic. This follows from the *Curry-Howard isomorphism* (see the next section). Since positive implication logic is decidable, type inhabitation is decidable as well. QED

3.3 The Curry-Howard isomorphism

There is a certain correspondence between typed λ-calculus and logic, which can be roughly described as follows:

Typed λ-calculus	*Logic*
type	formula, proposition
typable open term	derivation with assumptions
typable closed term	proof (derivation without assumptions)
β-contraction	contraction of derivation
typable term in β-normal form	derivation in normal form
β-equality	equality of derivations

Definition 3.19

- Type variables are also called *propositional variables*, types also *(implicational) formulas*. *formulas*
- A finite set of formulas is called *context*. *context*

  Metalinguistic variables for contexts are $\Delta, \Delta', \ldots$
- *Positive implication logic $P\!\to$* is given by the axiom scheme *positive implication logic $P\!\to$*

$$(\mathrm{Id})\ \ \Delta, \sigma \vdash \sigma$$

  and the two rules:

$$(\to\mathrm{I})\,\frac{\Delta, \sigma \vdash \tau}{\Delta \vdash \sigma \to \tau} \qquad\qquad (\to\mathrm{E})\,\frac{\Delta \vdash \sigma \to \tau \quad \Delta \vdash \sigma}{\Delta \vdash \tau}$$

  ($P\!\to$ is called *positive* implication logic, since negation does not occur.)
- $\Delta \vdash_{P\to} \sigma$ means that $\Delta \vdash \sigma$ is *derivable* in $P\!\to$. *derivable*
- For a judgement $M : \sigma$ let $(M : \sigma)^\circ \simeq \sigma$.
- For a basis $\Gamma = \{x_1 : \sigma_1, \ldots, x_n : \sigma_n\}$ let Γ° be the context $\{\sigma_1, \ldots, \sigma_n\}$.

Lemma 3.20 *If $\Gamma \vdash_{\lambda\to} M : \sigma$, then $\Gamma^\circ \vdash_{P\to} \sigma$.*

Proof. By application of $^\circ$ to every judgement in the $\lambda\!\to$-derivation of $\Gamma \vdash M : \sigma$ one obtains a $P\!\to$-derivation of $\Gamma^\circ \vdash \sigma$. QED

Lemma 3.21 *There exists an algorithm which yields for every typable term M a derivation of $\Delta \vdash \sigma$ in $P\!\to$ such that $\Delta = \Gamma^\circ$ and $\Gamma \vdash_{\lambda\to} M : \sigma$.*

Proof. The algorithm mentioned in Theorem 3.16 can generate for every typable term M its principal pair $\langle \Gamma, \sigma \rangle$; this can then be transformed directly into a $P{\rightarrow}$-sequent $\Delta \vdash \sigma$ with $\Delta = \Gamma^{\circ}$. The $P{\rightarrow}$-rule application necessary to derive this sequent in the last step is always determined by the form of M. QED

This means: Every typable term M encodes a derivation in $P{\rightarrow}$. From this derivation one can obtain by substitution all derivations of $\Gamma^{\circ} \vdash \sigma$ in $P{\rightarrow}$ which correspond to derivations of $\Gamma \vdash M : \sigma$ in $\lambda{\rightarrow}$.

Lemma 3.22 *For every derivation of* $\Delta \vdash \sigma$ *in* $P{\rightarrow}$ *one can construct a term* M *and a derivation of* $\Gamma \vdash M : \sigma$ *in* $\lambda{\rightarrow}$ *such that* $\Gamma^{\circ} = \Delta$.

Proof. Induction on the structure of the derivation of $\Delta \vdash \sigma$ in $P{\rightarrow}$ (where $\Delta = \{\sigma_1, \ldots, \sigma_n\}$):

Case (Id): All formulas σ occurring in instances of (Id) of $P{\rightarrow}$ are replaced by type declarations $x : \sigma$. The variable x is chosen in such a way that

- all occurrences of a formula σ have the *same* corresponding declaration $x : \sigma$;
- *different* formulas σ and τ have corresponding declarations $x : \sigma$ and $y : \tau$ with *different* variables x and y.

Case ($\rightarrow$I): The derivation in $P{\rightarrow}$ ends with

$$(\rightarrow\mathrm{I})\ \frac{\sigma_1, \ldots, \sigma_n, \sigma \vdash \tau}{\sigma_1, \ldots, \sigma_n \vdash \sigma \rightarrow \tau}$$

For the premiss $\sigma_1, \ldots, \sigma_n, \sigma \vdash \tau$ there is by the induction hypothesis a derivation in $\lambda{\rightarrow}$ of $x_1 : \sigma_1, \ldots, x_n : \sigma_n, x : \sigma \vdash M : \tau$. We extend this derivation by an application of ($\rightarrow$I) in $\lambda{\rightarrow}$ to obtain $x_1 : \sigma_1, \ldots, x_n : \sigma_n \vdash \lambda x.M : \sigma \rightarrow \tau$.

Case ($\rightarrow$E): The derivation in $P{\rightarrow}$ ends with

$$(\rightarrow\mathrm{E})\ \frac{\sigma_1, \ldots, \sigma_n \vdash \sigma \rightarrow \tau \qquad \sigma_1, \ldots, \sigma_n \vdash \sigma}{\sigma_1, \ldots, \sigma_n \vdash \tau}$$

For the premisses $\sigma_1, \ldots, \sigma_n \vdash \sigma \rightarrow \tau$ and $\sigma_1, \ldots, \sigma_n \vdash \sigma$ there are by the induction hypothesis derivations in $\lambda{\rightarrow}$ of

$$x_1 : \sigma_1, \ldots, x_n : \sigma_n \vdash M : \sigma \rightarrow \tau \qquad \text{and} \qquad x_1 : \sigma_1, \ldots, x_n : \sigma_n \vdash N : \sigma$$

Note that each type σ_i is assigned to exactly one variable x_i. By an application of ($\rightarrow$E) we thus obtain a derivation in $\lambda{\rightarrow}$ of $x_1 : \sigma_1, \ldots, x_n : \sigma_n \vdash MN : \tau$. QED

Theorem 3.23 (Curry-Howard isomorphism)
Let M_P *be the derivation in* $P{\rightarrow}$ *which corresponds to a term* M *typable in* $\lambda{\rightarrow}$*, as given by Lemma 3.21. Let* Π_λ *be the* $\lambda{\rightarrow}$*-term which corresponds to a derivation* Π *in* $P{\rightarrow}$*, as given by Lemma 3.22. Then the following holds:*

1. $(\Pi_\lambda)_P$ *is a derivation in* $P{\to}$ *from which we can obtain* Π *by substitution of formulas for propositional variables.*
2. $(M_P)_\lambda$ *is (modulo the renaming of free and/or bound variables) a term which results from* M *by identification of free or bound variables.*

Proof. By Lemmas 3.21 and 3.22. QED

An example for (2) is the λ-term

$$u(zx)(zy)$$

for which the type assignment algorithm yields

$$u:\alpha\to\alpha\to\beta, z:\gamma\to\alpha, x:\gamma, y:\gamma \vdash_{\lambda\to} u(zx)(zy):\beta$$

For $\Gamma = \{u:\alpha\to\alpha\to\beta, z:\gamma\to\alpha, x:\gamma, y:\gamma\}$ we have $\Gamma^\circ = \{\alpha\to\alpha\to\beta, \gamma\to\alpha, \gamma\}$. The corresponding derivation $(u(zx)(zy))_P$ in $P{\to}$

$$(\to\mathrm{E})\,\dfrac{(\to\mathrm{E})\,\dfrac{(\mathrm{Id})\,\dfrac{}{\Gamma^\circ\vdash\alpha\to(\alpha\to\beta)} \quad (\to\mathrm{E})\,\dfrac{(\mathrm{Id})\,\dfrac{}{\Gamma^\circ\vdash\gamma\to\alpha} \quad (\mathrm{Id})\,\dfrac{}{\Gamma^\circ\vdash\gamma}}{\Gamma^\circ\vdash\alpha}}{\Gamma^\circ\vdash\alpha\to\beta} \quad (\to\mathrm{E})\,\dfrac{(\mathrm{Id})\,\dfrac{}{\Gamma^\circ\vdash\gamma\to\alpha} \quad (\mathrm{Id})\,\dfrac{}{\Gamma^\circ\vdash\gamma}}{\Gamma^\circ\vdash\alpha}}{\Gamma^\circ\vdash\beta}$$

yields

$$\alpha\to\alpha\to\beta, \gamma\to\alpha, \gamma \vdash_{P\to} \beta.$$

According to Lemma 3.22, a λ-term of the form $u(zx)(zx)$, where x and y are identified, corresponds to this derivation.
The reason for this identification of variables is that information is lost in going from $\lambda{\to}$ to $P{\to}$, which cannot be regained by going from $P{\to}$ to $\lambda{\to}$. Note that by Lemma 3.22 the mapping of variables to formulas (= types) in instances of (Id) of $\lambda{\to}$ can never result in two variables having the same type. That is, by going from $\lambda{\to}$ to $P{\to}$ a sequent of the form $\Gamma, x:\sigma, y:\sigma \vdash M:\tau$ cannot occur.
In view of this loss of information one might prefer the weaker term *correspondence* instead of isomorphism. However, for a variant of natural deduction this loss of information can be avoided (see Troelstra & Schwichtenberg, 2001, Ch. 6; cp. also Sørensen & Urzyczyn, 2006, Ch. 4).

The Curry-Howard isomorphism induces reducibility and equality relations for derivations which correspond to $\rhd_\beta$ and $=_\beta$. These relations are investigated in proof theory, most prominently on the basis of the calculus of natural deduction (see Prawitz 2006).

Consider the β-redex $(\lambda x.M)N$ with type τ:

$$\dfrac{(\to\mathrm{I})\,\dfrac{\begin{array}{c}\mathscr{D}_1\\ \Gamma, x:\sigma \vdash M:\tau\end{array}}{\Gamma \vdash \lambda x.M:\sigma\to\tau} \qquad \begin{array}{c}\mathscr{D}_2\\ \Gamma \vdash N:\sigma\end{array}}{\Gamma \vdash (\lambda x.M)N:\tau}\,(\to\mathrm{E})$$

It is

$$(\lambda x.M)N \rhd_{1\beta} M[N/x]$$

To this there corresponds a contraction $\rhd^{\circ}_{1\beta}$ for derivations in natural deduction:

$$\dfrac{\dfrac{\begin{array}{c}[\sigma]^n\\ \mathscr{D}_1^{\circ}\\ \tau\end{array}}{\sigma\to\tau}\,(\to\mathrm{I})^n \qquad \begin{array}{c}\mathscr{D}_2^{\circ}\\ \sigma\end{array}}{\tau}\,(\to\mathrm{E}) \qquad \rhd^{\circ}_{1\beta} \qquad \begin{array}{c}\mathscr{D}_2^{\circ}\\ \sigma\\ \mathscr{D}_1^{\circ}\\ \tau\end{array}$$

where all occurrences of the assumption σ which are discharged by $(\to\mathrm{I})$ are replaced by copies of the derivation ending with σ:

$$\begin{array}{c}\mathscr{D}_2^{\circ}\\ \sigma\end{array}$$

If there are no such occurrences, then the derivation is transformed into

$$\begin{array}{c}\mathscr{D}_1^{\circ}\\ \tau\end{array}$$

The replacement of all occurrences of the assumption σ corresponds to the replacement of all occurrences of x in M by N, i.e. to the substitution $M[N/x]$.
Normalisability of λ-terms corresponds to normalisability of derivations in natural deduction and vice versa. If two derivations have the same normal form, then they are equal in the sense of β-equality.
The left derivation represents an argumentation in which a lemma of the form $\sigma\to\tau$ is used. This lemma does no longer occur in the right, contracted derivation. Normalisability of derivations ensures that by using a lemma only those things can be shown that could also be shown directly, i.e. without the lemma. This justifies the use of lemmas, which allows in general for shorter derivations.

4 The polymorphic typed λ-calculus

The polymorphic typed λ-calculus is also called *System F* or *typed λ-calculus of 2nd order*, short: $\lambda 2$.

Motivation: For example, the identity function $id \doteq \lambda x.x : \alpha \to \alpha$ should be independent of a special type α. That is, one would like to have $id \doteq \lambda x.x : \forall\alpha.(\alpha \to \alpha)$.

Definition 4.1

– The set of *types* of $\lambda 2$ is defined as follows: *types*

1. *Type variables* $\alpha, \beta, \gamma, \delta, \alpha_1, \alpha_2, \ldots$ are types. *type variables*
2. If σ and τ are types, then $(\sigma \to \tau)$ is a type (called *function type*). *function type*
3. If α is a type variable and σ is a type, then $\forall\alpha.\sigma$ is a type (called *universal (polymorphic) type*). *universal type*

– The universal quantifier $\forall$ binds stronger than $\to$.

$\forall\alpha_1\alpha_2\ldots\alpha_n.\sigma$ stands for $(\forall\alpha_1.(\forall\alpha_2.(\ldots(\forall\alpha_n.\sigma))))$.

– Occurrences of the type variable α in $\forall\alpha.\sigma$ are called *bound*. *bound*

The *set of free type variables* $\mathrm{FV}(\sigma)$ for a type σ is defined analogously to $\mathrm{FV}(M)$ for λ-terms M. *free type variable*

– A substitution $\sigma[\tau/\alpha]$ is only allowed, if τ is freely substitutable for α in σ (i.e. if τ does not contain a variable which would in $\sigma[\tau/\alpha]$ become bound by $\forall$).

Definition 4.2 *Type assignment* in $\lambda 2$ is defined by the axiom scheme *type assignment*

$$(\mathrm{Id})\ \ \Gamma, x:\sigma \vdash x:\sigma$$

and the rules:

$$(\to\mathrm{I})\ \frac{\Gamma, x:\sigma \vdash M:\tau}{\Gamma \vdash (\lambda x.M):\sigma\to\tau} \qquad (\to\mathrm{E})\ \frac{\Gamma \vdash M:\sigma\to\tau \quad \Gamma \vdash N:\sigma}{\Gamma \vdash MN:\tau}$$

$$(\forall\mathrm{I})\ \frac{\Gamma \vdash M:\sigma}{\Gamma \vdash M:\forall\alpha.\sigma}\ \text{if } \alpha \notin \mathrm{FV}(\Gamma) \qquad (\forall\mathrm{E})\ \frac{\Gamma \vdash M:\forall\alpha.\sigma}{\Gamma \vdash M:\sigma[\tau/\alpha]}$$

If $\Gamma \vdash M:\sigma$ is *derivable* in $\lambda 2$, then we write $\Gamma \vdash_{\lambda 2} M:\sigma$. *derivable*

Examples.

1. It is $\vdash_{\lambda 2} \lambda x.x : \forall\alpha.(\alpha\to\alpha)$:

$$(\forall\mathrm{I})\ \cfrac{(\to\mathrm{I})\ \cfrac{(\mathrm{Id})\ \cfrac{}{x:\alpha \vdash x:\alpha}}{\vdash \lambda x.x:\alpha\to\alpha}}{\vdash \lambda x.x:\forall\alpha.(\alpha\to\alpha)}$$

It is also $\vdash_{\lambda 2} \lambda x.x : \forall\alpha.\alpha \to \forall\beta.\beta$ and $\vdash_{\lambda 2} \lambda x.x : \forall\beta.(\forall\alpha.\alpha\to\beta)$.

2. It is $\vdash_{\lambda 2} \lambda xy.y : \forall\alpha.\forall\beta.(\alpha \to (\beta \to \beta))$:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{x:\alpha, y:\beta \vdash y:\beta}\,(\mathrm{Id})}{x:\alpha \vdash \lambda y.y:\beta\to\beta}\,(\to\mathrm{I})}{\vdash \lambda xy.y:\alpha\to(\beta\to\beta)}\,(\to\mathrm{I})}{\vdash \lambda xy.y:\forall\beta.(\alpha\to(\beta\to\beta))}\,(\forall\mathrm{I})}{\vdash \lambda xy.y:\forall\alpha.\forall\beta.(\alpha\to(\beta\to\beta))}\,(\forall\mathrm{I})$$

3. It is $\vdash_{\lambda 2} \lambda x.xx : \forall\beta.(\forall\alpha.\alpha \to \beta)$:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{x:\forall\alpha.\alpha \vdash x:\forall\alpha.\alpha}\,(\mathrm{Id})}{x:\forall\alpha.\alpha \vdash x:\alpha\to\beta}\,(\forall\mathrm{E})\,[\alpha\to\beta/\alpha] \quad \dfrac{\dfrac{}{x:\forall\alpha.\alpha \vdash x:\forall\alpha.\alpha}\,(\mathrm{Id})}{x:\forall\alpha.\alpha \vdash x:\alpha}\,(\forall\mathrm{E})\,[\alpha/\alpha]}{x:\forall\alpha.\alpha \vdash xx:\beta}\,(\to\mathrm{E})}{\vdash \lambda x.xx:\forall\alpha.\alpha\to\beta}\,(\to\mathrm{I})}{\vdash \lambda x.xx:\forall\beta.(\forall\alpha.\alpha\to\beta)}\,(\forall\mathrm{I})$$

It is also $\vdash_{\lambda 2} \lambda x.xx : \forall\beta.(\forall\alpha.\alpha \to (\beta\to\beta))$ and $\vdash_{\lambda 2} \lambda x.xx : \forall\alpha.\alpha \to \forall\beta.\beta$.

Definition 4.3 We define a *transition relation* $\sigma \sqsupset \tau$ and its transitive closure $\sigma \sqsupseteq \tau$ as follows: *transition relation*

– Let $\sigma \sqsupset \tau$ ("σ transitions into τ"), if

$\tau \doteq \forall\alpha.\sigma$ for a type variable α (τ is a *generalisation* of σ),

or

$\sigma \doteq \forall\alpha.\sigma_1$ and $\tau \doteq \sigma_1[\sigma_2/\alpha]$ for a type σ_2 (τ is a *specialisation* of σ).

– Let $\sigma \sqsupseteq \tau$ iff there are $n \geq 0$ such that $\sigma \doteq \sigma_1 \sqsupset \cdots \sqsupset \sigma_n \doteq \tau$.

Remarks.

1. The relation $\sqsupset$ is not symmetric:

   It holds that if $\sigma \sqsupset \forall\alpha.\sigma$, then $\forall\alpha.\sigma \sqsupset \sigma$. But if $\forall\alpha.\sigma_1 \sqsupset \sigma_1[\sigma_2/\alpha]$, then in general we do *not* have $\sigma_1[\sigma_2/\alpha] \sqsupset \forall\alpha.\sigma_1$.

2. Intuitively, $\sigma \sqsupseteq \tau$ means that $\Gamma \vdash M:\tau$ is derivable from $\Gamma \vdash M:\sigma$ by applications of $\forall$-rules only.

Lemma 4.4 *Let Γ be given. Then $\sigma \sqsupseteq \tau$ for $\sigma \doteq \sigma_1 \sqsupset \cdots \sqsupset \sigma_n \doteq \tau$, where no type variable occurring free in Γ is generalised in a step from σ_i to σ_{i+1}, iff there is a (sub-) derivation of the following form:*

$$\left.\begin{array}{c} \Gamma \vdash M:\sigma \\ \vdots \\ \Gamma \vdash M:\tau \end{array}\right\}\ \textit{only } \forall\textit{-rules.}$$

Proof. Definition of $\sqsupseteq$. QED

Lemma 4.5

1. *If* $\Gamma \vdash x : \sigma$*, then there is a type* σ' *with* $\sigma' \sqsupseteq \sigma$ *such that* $(x : \sigma') \in \Gamma$.
2. *If* $\Gamma \vdash \lambda x.M : \xi$*, then there are types* σ *and* τ *such that* $\Gamma, x : \sigma \vdash M : \tau$ *and* $(\sigma \to \tau) \sqsupseteq \xi$.
3. *If* $\Gamma \vdash MN : \tau$*, then there are* σ *and* τ' *with* $\tau' \sqsupseteq \tau$ *such that* $\Gamma \vdash M : \sigma \to \tau'$ *and* $\Gamma \vdash N : \sigma$.
4. *If* $\Gamma, x : \sigma \vdash M : \tau$ *and* $\Gamma \vdash N : \sigma$*, then* $\Gamma \vdash M[N/x] : \tau$.

Remark. Cp. Lemma 3.4 (4)-(6) and (9). The other assertions in Lemma 3.4 hold for $\lambda 2$ as well.

We will prove subject reduction (cp. Lemma 3.4, 10) in the following.

Definition 4.6 Let σ^0 be the type σ without quantifier prefix (i.e. initial quantifiers are discarded).

Examples.

1. $(\forall \alpha_1 \ldots \forall \alpha_n . \sigma)^0 \doteq \sigma$
2. $(\forall \alpha . (\forall \beta . \beta \to \alpha))^0 \doteq \forall \beta . \beta \to \alpha$

Lemma 4.7 *If* $(\sigma \to \tau) \sqsupseteq (\sigma' \to \tau')$*, then* $(\sigma' \to \tau') \doteq s(\sigma \to \tau)$ *for a substitution* s*, i.e.* $\sigma' \to \tau'$ *is more special than* $\sigma \to \tau$.

Proof. Let $(\sigma \to \tau) \doteq \sigma_1 \sqsupset \cdots \sqsupset \sigma_n \doteq (\sigma' \to \tau')$.
We show: $\sigma_i^0 \doteq s_i(\sigma \to \tau)$ for every s_i $(1 \leq i \leq n)$.
This implies the lemma, since $(\sigma' \to \tau')^0 \doteq (\sigma' \to \tau')$.

Proof by induction on n:

For $n = 1$: s_1 is the empty substitution.

For $n = m + 1$: Let $\sigma_m^0 \doteq s_m(\sigma \to \tau)$.

- Let $\sigma_{m+1} \doteq \forall \alpha . \sigma_m$. Then $\sigma_{m+1}^0 \doteq \sigma_m^0$, i.e. $s_{m+1} := s_m$.
- Let $\sigma_m \doteq \forall \alpha . \rho$ and $\sigma_{m+1} \doteq \rho[\rho_1/\alpha]$. Then let $s_{m+1} := s_m[\rho_1/\alpha]$, where $s_m[\rho_1/\alpha]$ differs from s_m only by $\alpha \mapsto \rho_1$. QED

Theorem 4.8 (Subject reduction) *If* $\Gamma \vdash M : \sigma$ *and* $M \rhd_\beta M'$*, then* $\Gamma \vdash M' : \sigma$.

Proof. We consider the case $M \doteq (\lambda x.P)Q \rhd_{1\beta} P[Q/x] \doteq M'$. From this the rest follows.

Suppose $\Gamma \vdash (\lambda x.P)Q : \sigma$ holds.

By Lemma 4.5 (3) there are then types τ and σ' with $\sigma' \sqsupseteq \sigma$ such that

$$\Gamma \vdash \lambda x.P : \tau \to \sigma' \quad \text{and} \quad \Gamma \vdash Q : \tau.$$

By Lemma 4.5 (2) there are then types τ and $\sigma' \sqsupseteq \sigma$ as well as types τ' and σ'' such that

$$\Gamma, x:\tau' \vdash P:\sigma'' \quad \text{and} \quad \Gamma \vdash Q:\tau$$

with $(\tau' \to \sigma'') \sqsupseteq (\tau \to \sigma')$.
By Lemma 4.7 there is then a type τ and $\sigma' \sqsupseteq \sigma$ such that

$$\Gamma, x:\tau \vdash P:\sigma' \quad \text{and} \quad \Gamma \vdash Q:\tau$$

where τ and σ' are more special than τ' and σ'', respectively.
By Lemma 4.5 (4) there is thus a type $\sigma' \sqsupseteq \sigma$ such that $\Gamma \vdash P[Q/x]:\sigma'$.
By Lemma 4.4 therefore $\Gamma \vdash P[Q/x]:\sigma$. QED

Next we will prove that every term typable in $\lambda 2$ is strongly normalisable.

Definition 4.9

- A *valuation* in SAT is a function v: type variables $\to$ SAT. *valuation*
- We define a *semantics* of types relative to valuations v; A is a set of terms: *semantics*
  1. $[\![\alpha]\!]_v := v(\alpha)$,
  2. $[\![\sigma \to \tau]\!]_v := [\![\sigma]\!]_v \to [\![\tau]\!]_v$,
  3. $[\![\forall\alpha.\sigma]\!]_v := \bigcap_{A \in \mathrm{SAT}} [\![\sigma]\!]_{v[\alpha \mapsto A]}$.

Lemma 4.10 *For every type σ and every valuation v it holds that $[\![\sigma]\!]_v$ is saturated (cp. Lemma 3.7).*

Proof. Analogously to the proof of Lemma 3.7. It remains to show that SAT is closed under intersection; but this is trivial. QED

Definition 4.11 We define *satisfiability* as follows, where for valuations ρ and interpretations $[\![M]\!]_\rho$ of terms Definition 3.8 applies: *satisfiability*

1. $\rho, v \vDash M:\sigma \;:\Longleftrightarrow\; [\![M]\!]_\rho \in [\![\sigma]\!]_v$;
2. $\rho, v \vDash \Gamma \;:\Longleftrightarrow\; \rho, v \vDash x:\sigma$ for all $(x:\sigma) \in \Gamma$;
3. $\Gamma \vDash M:\sigma \;:\Longleftrightarrow\;$ For all valuations ρ, v it holds: If $\rho, v \vDash \Gamma$, then $\rho, v \vDash M:\sigma$.

Lemma 4.12 (Soundness) *If $\Gamma \vdash M:\sigma$, then $\Gamma \vDash M:\sigma$.*

Theorem 4.13 *If $\Gamma \vdash M:\sigma$, then M is strongly normalisable.*

Proof. Suppose $\Gamma \vdash M:\sigma$. Then by soundness $\Gamma \vDash M:\sigma$ holds. That is, for all valuations ρ, v it holds: If $\rho, v \vDash \Gamma$, then $\rho, v \vDash M:\sigma$.
Since $[\![\sigma]\!]_v$ is saturated for every valuation v, we have $\rho_{id}, v \vDash \Gamma$ for every valuation v, where $\rho_{id}(x) := x$ for every variable x. Hence $\rho_{id}, v \vDash M:\sigma$ holds, i.e. $M \in [\![\sigma]\!]_v$. Therefore M is strongly normalisable. QED

Remarks.

1. For $\lambda 2$ the problems of typability, type checking and type inhabitation are undecidable.
2. $\lambda 2$ has more typable terms than $\lambda\!\to$.

   The term $\lambda x.xx$ is an example of a strongly normalisable term which is not typable in $\lambda\!\to$ but which is typable in $\lambda 2$.
3. Is typability in $\lambda 2$ = normalisability?

   No. However, every term in normal form is typable in $\lambda 2$; i.e. for every term M in normal form the following holds:

   $x_1 : \forall\alpha.\alpha, \dots, x_n : \forall\alpha.\alpha \vdash M : \sigma$ for a type σ, where $x_1, \dots, x_n$ are free variables in M.
4. But it is strong normalisability = typability in $\lambda\cap$ (System D).

   There are no universal types in $\lambda\cap$, but so-called *intersection types* $\sigma \cap \tau$, for which the following type assignment rules hold:

   $$(\cap\mathrm{I})\ \frac{\Gamma \vdash M : \sigma \qquad \Gamma \vdash M : \tau}{\Gamma \vdash M : \sigma \cap \tau} \qquad (\cap\mathrm{E})\ \frac{\Gamma \vdash M : \sigma \cap \tau}{\Gamma \vdash M : \sigma} \qquad (\cap\mathrm{E})\ \frac{\Gamma \vdash M : \sigma \cap \tau}{\Gamma \vdash M : \tau}$$

   Thus $M : \sigma \cap \tau$ expresses that M has both type σ and type τ.

   It is $\vdash_{\lambda\cap} \lambda x.xx : ((\sigma \to \tau) \cap \sigma) \to \tau$:

   $$(\to\mathrm{I})\ \dfrac{(\to\mathrm{E})\ \dfrac{(\cap\mathrm{E})\ \dfrac{(\mathrm{Id})\ \dfrac{}{x : (\sigma\to\tau)\cap\sigma \vdash x : (\sigma\to\tau)\cap\sigma}}{x : (\sigma\to\tau)\cap\sigma \vdash x : \sigma\to\tau} \qquad (\cap\mathrm{E})\ \dfrac{(\mathrm{Id})\ \dfrac{}{x : (\sigma\to\tau)\cap\sigma \vdash x : (\sigma\to\tau)\cap\sigma}}{x : (\sigma\to\tau)\cap\sigma \vdash x : \sigma}}{x : (\sigma\to\tau)\cap\sigma \vdash xx : \tau}}{\vdash \lambda x.xx : ((\sigma\to\tau)\cap\sigma)\to\tau}$$

Bibliography

Henk P. Barendregt: *Lambda calculi with types*, in: S. Abramsky, D. M. Gabbay and T. S. Maibaum (Hrsg.), *Handbook of Logic in Computer Science (vol. 2)*, pp. 117–309, Oxford University Press, 1992.

Henk P. Barendregt: *The Lambda Calculus. Its Syntax and Semantics*, College Publications, 2012. (Reprint of the 2nd edition, 1984.)

Alonzo Church: *A set of postulates for the foundation of logic*, Annals of Mathematics **33** (1932), 346–366.

Alonzo Church: *A Note on the Entscheidungsproblem*, Journal of Symbolic Logic **1** (1936a), 40–41.

Alonzo Church: *An unsolvable problem of elementary number theory*, American Journal of Mathematics **58** (1936b), 345–363.

Alonzo Church: *A formulation of the simple theory of types*, Journal of Symbolic Logic **5** (1940), 56–68.

Alonzo Church and J. Barkley Rosser: *Some properties of conversion*, Transactions of the American Mathematical Society **39** (1936), 472–482.

Haskell B. Curry: *Grundlagen der Kombinatorischen Logik*, American Journal of Mathematics **52** (1930), 509–536.

J. Roger Hindley and Jonathan P. Seldin: *Lambda-Calculus and Combinators, an Introduction*, Cambridge University Press, 2008 (reprinted 2010).

Dag Prawitz: *Natural Deduction. A Proof-Theoretical Study*, Almqvist & Wiksell, 1965. Reprinted 2006, Dover Publications.

Paul C. Rosenbloom: *The Elements of Mathematical Logic*, Dover Publications, 1950 (reprinted 2005).

Moses Schönfinkel: *Über die Bausteine der mathematischen Logik*, Mathematische Annalen **92** (1924), 305–316.

Moses Schönfinkel and Paul Bernays: *Zum Entscheidungsproblem der mathematischen Logik*, Mathematische Annalen **99** (1929), 342–72.

Morten H. Sørensen and Paweł Urzyczyn: *Lectures on the Curry-Howard Isomorphism*, Elsevier, 2006.

William W. Tait: *Intensional interpretations of functionals of finite type I*, Journal of Symbolic Logic **32** (1967), 198–212.

Anne S. Troelstra and Helmut Schwichtenberg: *Basic Proof Theory*, 2nd edition, Cambridge University Press, 2001.

Alan M. Turing: *The* $\mathfrak{p}$*-function in* λ*-*K*-conversion*, Journal of Symbolic Logic **2** (1937), 164.

David A. Turner: *A new implementation technique for applicative languages*, Software: Practice and Experience **9** (1979), 31–49.

Ludwig Wittgenstein: *Tractatus Logico-Philosophicus*, Kegan Paul, 1922.

Index