

# Logic of Proof Assistants

Prof. Floris van Doorn \*

University of Bonn

27<sup>th</sup> June, 2025

## Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
1.1	Inductive Definitions . . . . .	3
<b>2</b>	<b>First-Order Logic</b>	<b>5</b>
2.1	Provability . . . . .	8
2.2	Semantics . . . . .	9
2.3	Definite descriptions . . . . .	10
<b>3</b>	<b><math>\lambda</math>-calculus</b>	<b>12</b>
3.1	Partial recursive functions . . . . .	18
3.2	Simply typed $\lambda$ -calculus . . . . .	21
<b>4</b>	<b>Simple Type Theory</b>	<b>29</b>

---

\*L<sup>A</sup>T<sub>E</sub>X-realization by Hannah Scholz

# 1 Introduction

The topics of this class are:

- (i) First-order Logic/Set Theory
- (ii) Lambda Calculus
- (iii) Simple Type Theory (Higher-Order Logic)
- (iv) Dependent Type Theory/Homotopy Type Theory

**Example 1.1.** Here are examples of proof assistants for these different types of logics:

- (i) First-order Logic/Set Theory: Mizar, Metamath
- (ii) Simple Type Theory: Isabelle/HoL, HoL Light
- (iii) Dependent Type Theory: Lean, Rocq (formerly Coq), Agda
- (iv) Homotopy Type Theory: cubicaltt, rzk

**Remark 1.2.** You might want to have the following criteria for a logic:

- (i) Appropriate (You can encode mathematical arguments.)
- (ii) Simple (It is relatively easy to understand.)
- (iii) Expressive (Mathematical arguments are convenient to express.)

**Theorem 1.3.** *Let  $\pi$  be the prime counting function, i.e.  $\pi: \mathbb{R} \rightarrow \mathbb{N}$ ,  $x \mapsto |\{p \leq x \mid p \text{ prime}\}|$ . Then  $\lim_{x \rightarrow \infty} \frac{\pi(x)}{x/\log(x)} = 1$ .*

**Remark 1.4.** When formalizing/stating this theorem in a formal logic there are a few things that you need to think about:

- (i) What do you do about division by zero?
- (ii) What does division even mean? (Do you define division for  $\mathbb{R}$  explicitly? Do you define it generally for a field? Or even for a group? How do you ensure that the “correct” field structure on  $\mathbb{R}$  gets used?)
- (iii) How do you define a limit? (Do you define a limit for  $\mathbb{R}$  explicitly? Or for every topological space? How do you ensure the “correct” topology on  $\mathbb{R}$  gets used? How do you deal with potentially non-unique limits (for example in non-Hausdorff spaces)?)

**Remark 1.5.** You can make the following design choices for “a logic”:

- (i) Is the logic typed or untyped?
- (ii) Is the logic constructive or classical?
- (iii) Does the logic support computation?

**Remark 1.6.** In logic there is the **object language** and we reason about it in a **meta-language** (“ordinary mathematical reasoning”).

## 1.1 Inductive Definitions

**Example 1.7.** The natural numbers are inductively defined by  $0 \in \mathbb{N}$  and  $S: \mathbb{N} \rightarrow \mathbb{N}$ ,  $n \mapsto n + 1$ .

**Definition 1.8.** Let  $U$  be a set and  $\mathcal{C} \subseteq \bigcup_{n \in \mathbb{N}} (U^n \rightarrow U)$  a set of **constructors**, where  $c: U^n \rightarrow U$  is called an  **$n$ -ary function** and  $(U^n \rightarrow U)$  is the collection on  $n$ -ary functions.

- (i)  $A \subseteq U$  is **closed under  $\mathcal{C}$**  if for any  $n$ -ary  $c \in \mathcal{C}$  and for all  $x_1, \dots, x_n \in A$  we have that  $c(x_1, \dots, x_n) \in A$ .
- (ii)  $A \subseteq U$  is **generated by  $\mathcal{C}$**  or **inductively defined by  $\mathcal{C}$**  if  $A$  is the smallest set that is closed under  $\mathcal{C}$ , i.e.  $A = \bigcap \{B \subseteq U \mid B \text{ is closed under } \mathcal{C}\}$ .
- (iii)  $A \subseteq U$  is **freely generated by  $\mathcal{C}$**  if
  - (a) each constructor is injective on  $A^n$  and
  - (b) the images of different constructors are disjoint.

**Remark 1.9.**  $\emptyset$  is closed under  $\mathcal{C}$  iff  $\mathcal{C}$  has no nullary constructors.

**Exercise 1.10.**  $\bigcap \{B \subseteq U \mid B \text{ is closed under } \mathcal{C}\}$  is closed under  $\mathcal{C}$ .

**Example 1.11.**

- (i) The free group.
- (ii) The  $\sigma$ -algebra generated by a collection of subsets. (This is not freely generated.)
- (iii) The topology generated by a collection of subsets. (This is not freely generated.)

**Theorem 1.12** (Structural Induction). *If  $A \subseteq U$  is generated by  $\mathcal{C}$  and  $P: A \rightarrow \{\top, \perp\}$  is a predicate on  $A$ , to prove  $\forall a \in A, P(a)$  it suffices to show: for any  $n$ -ary  $c \in \mathcal{C}$  and any  $x_1, \dots, x_n \in A$  if  $P(x_1), \dots, P(x_n)$  then  $P(c(x_1, \dots, x_n))$ .*

*Proof.* Exercise. □

**Remark 1.13.** The base case of the induction is given by nullary constructors.

**Theorem 1.14** (Structural Recursion). *If  $A \subseteq U$  is freely generated by  $\mathcal{C}$ ,  $B$  is a set and for any  $n$ -ary  $c \in \mathcal{C}$  we have a  $g_c: B^n \rightarrow B$  then there is a unique function  $f: A \rightarrow B$  such that  $f(c(a_1, \dots, a_n)) = g_c(f(a_1), \dots, f(a_n))$  for every  $c \in \mathcal{C}$  and  $a_1, \dots, a_n \in A$ .*

*Proof.* Exercise. □

**Example 1.15.** For  $A = \mathbb{N}$  this reduces to  $f(0) := g_0$  and  $f(S(n)) := g_s(f(n))$ .

## 2 First-Order Logic

**Definition 2.1.** A (first-order) **language**  $\mathcal{L}$  is a triple  $(\mathcal{F}, \mathcal{R}, a)$  where  $\mathcal{F}$  is a set of function symbols,  $\mathcal{R}$  is a set of relation symbols,  $\mathcal{F}$  and  $\mathcal{R}$  are disjoint and  $a: \mathcal{F} \cup \mathcal{R} \rightarrow \mathbb{N}$  is the arity function.

**Example 2.2.** A language for groups  $\mathcal{L}_{\text{Group}}$  has  $\mathcal{F} := \{\cdot, {}^{-1}, 1\}$ ,  $\mathcal{R} := \emptyset$ ,  $a(\cdot) = 2$ ,  $a({}^{-1}) = 1$  and  $a(1) = 0$ .

**Definition 2.3.** We fix an infinite set of **variables**  $\mathcal{V} := \{x_0, x_1, \dots\}$ .

**Remark 2.4.** We use  $x$  for variables,  $f$  and  $g$  for functions and  $R$  and  $S$  for relations.

**Definition 2.5.** We can define the **terms**  $T_{\mathcal{L}}$  in the language  $\mathcal{L}$  using the **Backus–Naur form (BNF)**:

$$s, t ::= x \mid f(t_1, \dots, t_n)$$

where  $f$  is an  $n$ -ary function symbol.

**Definition 2.6.** Formally, we define the **terms**  $T_{\mathcal{L}}$  in the language  $\mathcal{L}$  in the following way. We define the set of **symbols**  $S := \mathcal{F} \dot{\cup} \mathcal{V} \dot{\cup} \{“(”, “)”, “,”, “.”\}$  and the set of finite sequences of symbols  $S^*$ . Let  $\mathcal{C}$  be defined as:

- (i) for each variable  $x \in \mathcal{V}$  there is a nullary constructor  $c_x := x$
- (ii) for each  $n$ -ary function symbol  $f$  there is an  $n$ -ary constructor  $c_f: (S^*)^n \rightarrow S^*$ ,  $c_f(t_1, \dots, t_n) := f\text{“}(\text{“}t_1\text{“}, \dots, \text{“}t_n\text{“})\text{”}$

Then  $T_{\mathcal{L}} \subseteq S^*$  is the set generated by  $\mathcal{C}$ .

**Example 2.7.**

- (i) “(“”“)”“, ” $f$  is in  $S^*$  but not in  $T_{\mathcal{L}}$ .
- (ii) If  $f$  is binary then  $f$ “(” $x_0$ “, ” $x_1$ “)” is in  $T_{\mathcal{L}}$ .

**Remark 2.8.** Technically, the brackets and commas are not necessary. They are however necessary when you use infix notation. (For example the meaning of  $a \cdot b + c$  is unclear.)

**Definition 2.9.** First-order **formulas**  $\Phi_{\mathcal{L}}$  are specified by

$$\varphi, \psi ::= \perp \mid s = t \mid R(t_1, \dots, t_n) \mid (\varphi \wedge \psi) \mid (\varphi \vee \psi) \mid (\varphi \rightarrow \psi) \mid (\forall x. \varphi) \mid (\exists x. \varphi)$$

where  $R$  is an  $n$ -ary relation symbol and  $t_1, \dots, t_n \in T_{\mathcal{L}}$ .

**Remark 2.10.** In classical logic one could omit the rules  $(\varphi \wedge \psi)$  and  $(\varphi \vee \psi)$  (as they can be defined using the other rules). They are however necessary for constructive logic.

**Remark 2.11.** We can define other connectives:

- (i)  $\neg \varphi := (\varphi \rightarrow \perp)$
- (ii)  $\varphi \leftrightarrow \psi := ((\varphi \rightarrow \psi) \wedge (\psi \rightarrow \varphi))$

**Remark 2.12.** When writing formulas we omit some parentheses:

- (i)  $\varphi \rightarrow \psi \rightarrow \theta$  means  $\varphi \rightarrow (\psi \rightarrow \theta)$
- (ii)  $\forall x. \varphi \rightarrow \psi$  means  $\forall x. (\varphi \rightarrow \psi)$

**Remark 2.13.** We want  $\forall x. x = x$  and  $\forall y. y = y$  to mean the same thing. Options to achieve this are:

- (i) Define  $(\forall x. x = x) \equiv_{\alpha} (\forall y. y = y)$  to be  **$\alpha$ -equivalent**. And then define the set of formulas to be  $\Phi_{\mathcal{L}} / \equiv_{\alpha}$ .
- (ii) We could not use variable names for bound variables and use **de Bruijn indices** instead.

**Remark 2.14.**  $\forall x. x = y$  has **bound variables**  $\{x\}$  and **free variables**  $\{y\}$ . For a formula  $\varphi$  or a term  $t$  we also write  $\text{fv}(\varphi)$  and  $\text{fv}(t)$  for the set of free variables in  $\varphi$  and  $t$ .

**Definition 2.15.** A **sentence** is a formula without free variables.

**Definition 2.16.** **Substitution**  $s[t/x]$  of  $x$  by  $t$  in a term  $s$  is defined recursively by

- (i)  $y[t/x] := \begin{cases} t & \text{if } y = x \\ y & \text{otherwise} \end{cases}$
- (ii)  $f(s_1, \dots, s_n)[t/x] := f(s_1[t/x], \dots, s_n[t/x])$

**Example 2.17.** Defining substitution in formulas is a little bit harder as we need to avoid **variable capture**:  $(\exists x.x \leq z)[(x+1)/z]$  should not be  $\exists x.x \leq x+1$  but  $\exists y.y \leq x+1$ .

**Definition 2.18.** For a formula  $\varphi$  **substitution**  $\varphi[t/x]$  is defined as:

- (i)  $(s = s')[t/x] := (s[t/x] = s'[t/x])$
- (ii)  $R(t_1, \dots, t_n)[t/x] := R(t_1[t/x], \dots, t_n[t/x])$
- (iii)  $(\varphi \vee \psi)[t/x] := (\varphi[t/x] \vee \psi[t/x])$
- (iv)  $(\varphi \wedge \psi)[t/x] := (\varphi[t/x] \wedge \psi[t/x])$
- (v)  $(\varphi \rightarrow \psi)[t/x] := (\varphi[t/x] \rightarrow \psi[t/x])$
- (vi)  $(\forall y.\varphi)[t/x] := \begin{cases} \forall y.\varphi & \text{if } y = x \\ \forall z.\varphi[z/y][t/x] & \text{otherwise} \end{cases}$  where  $z$  does not occur in  $t$ .
- (vii)  $(\exists y.\varphi)[t/x] := \begin{cases} \exists y.\varphi & \text{if } y = x \\ \exists z.\varphi[z/y][t/x] & \text{otherwise} \end{cases}$  where  $z$  does not occur in  $t$ .

**Definition 2.19.**  $\alpha$ -**equivalence** is the **congruence closure** of

- (i)  $(\forall x.\varphi) \equiv_\alpha (\forall y.\varphi[y/x])$
- (ii)  $(\exists x.\varphi) \equiv_\alpha (\exists y.\varphi[y/x])$

i.e. it is the smallest equivalence relation containing these two rules and respecting the connectives:

- (i)  $(\varphi_1 \wedge \varphi_2) \equiv_\alpha (\psi_1 \wedge \psi_2)$  for  $\varphi_1 \equiv_\alpha \psi_1$  and  $\varphi_2 \equiv_\alpha \psi_2$
- (ii)  $(\varphi_1 \vee \varphi_2) \equiv_\alpha (\psi_1 \vee \psi_2)$  for  $\varphi_1 \equiv_\alpha \psi_1$  and  $\varphi_2 \equiv_\alpha \psi_2$
- (iii)  $(\varphi_1 \rightarrow \varphi_2) \equiv_\alpha (\psi_1 \rightarrow \psi_2)$  for  $\varphi_1 \equiv_\alpha \psi_1$  and  $\varphi_2 \equiv_\alpha \psi_2$
- (iv)  $(\forall x.\varphi) \equiv_\alpha (\forall x.\psi)$  if  $\varphi \equiv_\alpha \psi$
- (v)  $(\exists x.\varphi) \equiv_\alpha (\exists x.\psi)$  if  $\varphi \equiv_\alpha \psi$

**Remark 2.20.** We will treat  $\alpha$ -equivalence as an equivalence relation. You could also define the formulas as  $\Phi_{\mathcal{L}} / \equiv_\alpha$  and thus treat  $\alpha$ -equivalence as equality.

## 2.1 Provability

**Definition 2.21.** Let  $\Gamma$  be a set of formulas and  $\varphi$  a formula. Then  $\Gamma \vdash \varphi$  (read : “ $\Gamma$  proves  $\varphi$ ”) is defined inductively by

- (i)  $\overline{\Gamma, \varphi \vdash \varphi}$  (assumption rule)
- (ii)  $\frac{\Gamma \vdash \varphi \quad \Gamma \vdash \psi}{\Gamma \vdash \varphi \wedge \psi}$  ( $\wedge$ -introduction)
- (iii)  $\frac{\Gamma \vdash \varphi_1 \wedge \varphi_2}{\Gamma \vdash \varphi_i}$  for  $i = 1, 2$  ( $\wedge$ -elimination)
- (iv)  $\frac{\Gamma \vdash \varphi_i}{\Gamma \vdash \varphi_1 \vee \varphi_2}$  for  $i = 1, 2$  ( $\vee$ -introduction)
- (v)  $\frac{\Gamma \vdash \varphi \vee \psi \quad \Gamma, \varphi \vdash \theta \quad \Gamma, \psi \vdash \theta}{\Gamma \vdash \theta}$  ( $\vee$ -elimination)
- (vi)  $\frac{\Gamma, \varphi \vdash \psi}{\Gamma \vdash \varphi \rightarrow \psi}$  ( $\rightarrow$ -introduction)
- (vii)  $\frac{\Gamma \vdash \varphi \rightarrow \psi \quad \Gamma \vdash \varphi}{\Gamma \vdash \psi}$  ( $\rightarrow$ -elimination)
- (viii)  $\frac{\Gamma, \neg\varphi \vdash \perp}{\Gamma \vdash \varphi}$  (proof by contradiction)
- (ix)  $\frac{\Gamma \vdash \varphi}{\Gamma \vdash \forall x.\varphi}$  for  $x \notin \text{fv}(\Gamma)$  ( $\forall$ -introduction)
- (x)  $\frac{\Gamma \vdash \forall x.\varphi}{\Gamma \vdash \varphi[t/x]}$  ( $\forall$ -elimination)
- (xi)  $\frac{\Gamma \vdash \varphi[t/x]}{\Gamma \vdash \exists x.\varphi}$  ( $\exists$ -introduction)
- (xii)  $\frac{\Gamma \vdash \exists x.\varphi \quad \Gamma, \varphi \vdash \psi}{\Gamma \vdash \psi}$  for  $x \notin \text{fv}(\Gamma, \psi)$  ( $\exists$ -elimination)
- (xiii)  $\overline{\Gamma \vdash t = t}$  ( $=$ -introduction)
- (xiv)  $\frac{\Gamma \vdash s = t \quad \Gamma \vdash \varphi[t/x]}{\Gamma \vdash \varphi[s/x]}$  ( $=$ -elimination)
- (xv)  $\frac{\Gamma \vdash \varphi}{\Gamma \vdash \psi}$  for  $\varphi \equiv_\alpha \psi$  ( $\alpha$ -equivalence)

**Remark 2.22.** Read “ $\frac{A}{B}$ ” as: “Under the assumptions A we can prove B”. With “ $\Gamma, \varphi$ ” we really mean  $\Gamma \cup \{\varphi\}$ .



**Example 2.23.** If  $\varphi$  and  $\psi$  are formulas then we can show  $\vdash (\varphi \wedge \psi) \rightarrow (\psi \wedge \varphi)$  using the following **proof tree** :

$$\frac{\frac{\frac{\varphi \wedge \psi \vdash \varphi \wedge \psi}{\varphi \wedge \psi \vdash \varphi} \text{Assump.}}{\varphi \wedge \psi \vdash \psi} \wedge\text{-elim.} \quad \frac{\frac{\frac{\varphi \wedge \psi \vdash \varphi \wedge \psi}{\varphi \wedge \psi \vdash \varphi} \text{Assump.}}{\varphi \wedge \psi \vdash \psi} \wedge\text{-elim.}}{\varphi \wedge \psi \vdash \psi \wedge \varphi} \wedge\text{-intro.} \\ \frac{\varphi \wedge \psi \vdash \psi \wedge \varphi}{\vdash (\varphi \wedge \psi) \rightarrow (\psi \wedge \varphi)} \rightarrow\text{-intro.}$$

## 2.2 Semantics

**Definition 2.24.** An  $\mathcal{L}$ -**structure**  $\mathcal{M}$  consists of

- (i) a non-empty set  $|\mathcal{M}|$
- (ii) for any  $n$ -ary function symbol  $f$  a function  $f_{\mathcal{M}}: |\mathcal{M}|^n \rightarrow |\mathcal{M}|$
- (iii) for any  $n$ -ary relation symbol  $R$  a set  $R_{\mathcal{M}} \subseteq |\mathcal{M}|^n$

**Definition 2.25.** If  $t$  is an  $\mathcal{L}$ -term and  $\sigma: \mathcal{V} \rightarrow |\mathcal{M}|$  we define  $\llbracket t \rrbracket_{\mathcal{M}, \sigma}$  as:

- (i)  $\llbracket x \rrbracket_{\mathcal{M}, \sigma} := \sigma(x)$
- (ii)  $\llbracket f(t_1, \dots, t_n) \rrbracket_{\mathcal{M}, \sigma} := f_{\mathcal{M}}(\llbracket t_1 \rrbracket_{\mathcal{M}, \sigma}, \dots, \llbracket t_n \rrbracket_{\mathcal{M}, \sigma})$

For formulas we define recursively that  $\mathcal{M} \models_{\sigma} \varphi$  holds if

- (i)  $\mathcal{M} \models_{\sigma} R(t_1, \dots, t_n)$  iff  $R_{\mathcal{M}}(\llbracket t_1 \rrbracket_{\mathcal{M}, \sigma}, \dots, \llbracket t_n \rrbracket_{\mathcal{M}, \sigma})$
- (ii)  $\mathcal{M} \models_{\sigma} \perp$  never holds
- (iii)  $\mathcal{M} \models_{\sigma} s = t$  iff  $\llbracket s \rrbracket_{\mathcal{M}, \sigma} = \llbracket t \rrbracket_{\mathcal{M}, \sigma}$
- (iv)  $\mathcal{M} \models_{\sigma} \varphi \wedge \psi$  iff  $\mathcal{M} \models_{\sigma} \varphi$  and  $\mathcal{M} \models_{\sigma} \psi$
- (v)  $\mathcal{M} \models_{\sigma} \varphi \vee \psi$  iff  $\mathcal{M} \models_{\sigma} \varphi$  or  $\mathcal{M} \models_{\sigma} \psi$
- (vi)  $\mathcal{M} \models_{\sigma} \varphi \rightarrow \psi$  iff  $\mathcal{M} \models_{\sigma} \varphi$  implies  $\mathcal{M} \models_{\sigma} \psi$
- (vii)  $\mathcal{M} \models_{\sigma} \forall x. \varphi$  iff for all  $a \in |\mathcal{M}|$  we know that  $\mathcal{M} \models_{\sigma, x \mapsto a} \varphi$  where

$$(\sigma, x \mapsto a)(y) := \begin{cases} a & y = x \\ \sigma(y) & \text{otherwise} \end{cases}$$

- (viii)  $\mathcal{M} \models_{\sigma} \exists x. \varphi$  iff there is  $a \in |\mathcal{M}|$  such that  $\mathcal{M} \models_{\sigma, x \mapsto a} \varphi$

**Remark 2.26.** We write  $\varphi(\vec{x})$  to mean that  $\text{fv}(\varphi) \subseteq \vec{x}$  and  $\varphi(\vec{t})$  for  $\varphi[\vec{t}/\vec{x}]$ .

**Remark 2.27.**  $\llbracket t \rrbracket_{\mathcal{M}, \sigma}$  and  $\mathcal{M} \models_{\sigma} \varphi$  only depend on the values  $\sigma(x)$  where  $x \in \text{fv}(t)$  and  $x \in \text{fv}(\varphi)$  respectively. If  $\varphi$  is a sentence then  $\mathcal{M} \models_{\sigma} \varphi$  does not depend on  $\sigma$  and is denoted  $\mathcal{M} \models \varphi$  (read: “ $\mathcal{M}$  realizes  $\varphi$ ”).

**Definition 2.28.** If  $\Gamma$  is a set of formulas and  $\varphi$  is a formula, then  $\Gamma \models \varphi$  means that for any  $\mathcal{L}$ -structure  $\mathcal{M}$  and assignment  $\sigma: \mathcal{V} \rightarrow |\mathcal{M}|$  such that  $\mathcal{M} \models_{\sigma} \psi$  for all  $\psi \in \Gamma$  we have  $\mathcal{M} \models_{\sigma} \varphi$ .

**Theorem 2.29** (Soundness theorem). *If  $\Gamma \vdash \varphi$  then  $\Gamma \models \varphi$ .*

**Theorem 2.30** (Completeness theorem). *If  $\Gamma \models \varphi$  then  $\Gamma \vdash \varphi$ .*

**Theorem 2.31** (Compactness theorem). *If  $\Gamma \models \varphi$  then for some finite  $\Gamma' \subseteq \Gamma$  we have  $\Gamma' \models \varphi$ .*

## 2.3 Definite descriptions

**Definition 2.32.**  $\exists!x.\varphi(x, \vec{z}) := \exists x.(\varphi(x, \vec{z}) \wedge \forall y.(\varphi(y, \vec{z}) \rightarrow y = x))$ .

**Definition 2.33.** Suppose  $\Gamma$  is a set of  $\mathcal{L}$ -sentences,  $\Gamma'$  a set of  $\mathcal{L}'$ -sentences and  $\mathcal{L} \subseteq \mathcal{L}'$ . Then  $\Gamma'$  is **conservative over**  $\Gamma$  if  $\Gamma \subseteq \Gamma'$  and for all  $\mathcal{L}$ -formulas  $\psi$  such that  $\Gamma' \vdash \psi$  we have  $\Gamma \vdash \psi$ .

**Theorem 2.34.** *Suppose that  $\Gamma \vdash \forall \vec{x}.\exists!y.\varphi(\vec{x}, y)$  and that  $f$  is a fresh function symbol (i.e. not among the function symbols of  $\mathcal{L}$ ) then  $\Gamma \cup \{\forall \vec{x}.\varphi(\vec{x}, f(\vec{x}))\}$  is conservative over  $\Gamma$ .*

**Definition 2.35** (Axioms of ZFC).  $\mathcal{L}_{\text{ZFC}}$  has no function symbol and one binary relation “ $\in$ ”. The axioms of ZFC are

- (i) Extensionality :  $\forall x \forall y. (\forall z. z \in x \leftrightarrow z \in y) \rightarrow x = y$
- (ii) Pairing:  $\forall x \forall y \exists z \forall w. w \in z \leftrightarrow w = x \vee w = y$  (“ $z = \{x, y\}$ ”)

This allows us to define  $\{x\} := \{x, x\}$ .

- (iii) Union:  $\forall x \exists y \forall z. z \in y \leftrightarrow \exists w. (w \in x \wedge z \in w)$  (“ $y = \bigcup x$ ”)

This allows us to define  $x \cup y := \bigcup \{x, y\}$ .

- (iv) Power set:  $\forall x \exists y \forall z. z \in y \leftrightarrow z \subseteq x$  (“ $y = \mathcal{P}(x)$ ”)

where  $z \subseteq x$  means  $\forall w. w \in z \rightarrow w \in x$

- (v) Separation (axiom schema): for any formula  $\varphi(\vec{x}, y)$  we have  $\forall \vec{x} \forall y \exists z \forall w. w \in z \leftrightarrow (w \in y \wedge \varphi(\vec{x}, w))$  (“ $z = \{w \in y \mid \varphi(\vec{x}, w)\}$ ”)

- (vi) Infinity:  $\exists x. \emptyset \in x \wedge \forall y. y \in x \rightarrow y \cup \{y\} \in x$  where  $\emptyset := \{w \in y \mid \perp\}$

- (vii) Foundation:  $\forall x. (\exists y. y \in x) \rightarrow \exists y. y \in x \wedge \forall z. z \in x \rightarrow z \not\in y$  (“Every set  $x$  contains an element  $y$  disjoint from  $x$ ”)

- (viii) Replacement (axiom schema): For every formula  $\varphi(z, w, \vec{y})$  we have  $\forall x \forall \vec{y} (\forall z. z \in x \rightarrow \exists! w \varphi(z, w, \vec{y})) \rightarrow \exists u \forall w. w \in u \leftrightarrow \exists z. z \in x \wedge \varphi(z, w, \vec{y})$  (“If  $\varphi$  is a function with domain  $x$  then the image of  $\varphi$  is a set.”)

- (ix) Choice:  $\forall x. \emptyset \notin x \rightarrow \exists f. f \in (x \rightarrow \bigcup x) \wedge \forall y. y \in x \rightarrow f(y) \in y$

where we define  $(x, y) := \{\{x\}, \{x, y\}\}$ ,

$A \times B := \{z \in \mathcal{P}(\mathcal{P}(A \cup B)) \mid \exists x \in A \exists y \in B. z = (x, y)\}$ ,

$(A \rightarrow B) := \{f \in \mathcal{P}(A \times B) \mid \forall x \in A \exists! y. (x, y) \in f\}$  and

$$f(x) := \begin{cases} y & \text{if } (x, y) \in f \\ \emptyset & \text{if no such } y \text{ exists} \end{cases}$$

**Remark 2.36.** The existence of at least one set is provable and therefore the empty set also exists. Nonetheless, the existence of the empty set is often added as an axiom.

**Remark 2.37.** In principle, you can do almost all math in the combination of FOL and ZFC. In practice, you want to do meta-logical operations (e.g. quantifying over formulas). For example:

- (i) You want to be able to define new definitions with definite descriptions.
- (ii) You want to be able to talk about theorem schemes.

Mizar and Metamath are two proof systems implementing FOL + ZFC in a metalogic.

### 3 $\lambda$ -calculus

**Definition 3.1.** Let  $\mathcal{V} = \{x_0, x_1, \dots\}$  a countably infinite set of variables and  $\mathcal{C} = \{c_0, c_1, \dots\}$  be any set of constants. The terms of the  $\lambda$ -calculus are given by the following BNF:

$$s, t ::= x \mid c \mid (st) \mid (\lambda x.t)$$

where  $x$  is a variable and  $c$  a constant.

**Remark 3.2.**

- (i) Think of  $(\lambda x.t)$  as the function  $x \mapsto t(x)$  and  $(st)$  as function application.
- (ii) Anything can apply to any term, e.g.  $(xx)$  is a term.
- (iii) Abbreviate  $((rs)t)u$  to  $rstu$  and  $\lambda x.\lambda y.\lambda z.t$  to  $\lambda xyz.t$ . For example,  $\lambda xy.xy$  means  $(\lambda x.(\lambda y.(xy)))$ .
- (iv)  $\lambda x.t$  binds the variable  $x$ . Like in first-order logic we can define  $\alpha$ -equivalence ( $\equiv_\alpha$ ) and substitution  $(t[s/x])$ . Here we identify  $\alpha$ -equivalent terms, e.g.  $\lambda x.x = \lambda y.y$ .

**Example 3.3.**  $(x(\lambda x.x))[s/x] = s(\lambda x.x)$

**Remark 3.4.** Consider  $(\lambda x.t)s$ . We want this to correspond to  $t[s/x]$ .

**Definition 3.5.**

- (i)  **$\beta$ -contraction** ( $\triangleright_\beta$ ) is defined as  $(\lambda x.t)s \triangleright_\beta t[s/x]$ .  $(\lambda x.t)s$  is called a  **$\beta$ -redex** and  $t[s/x]$  is called its  **$\beta$ -reduct**.
- (ii) **One-step- $\beta$ -reduction** ( $\rightarrow_{\beta,1}$ ) is defined as the compatible closure of  $\triangleright_\beta$ , i.e.
  - (a) If  $s \triangleright_\beta t$  then  $s \rightarrow_{\beta,1} t$ .
  - (b) If  $s \rightarrow_{\beta,1} t$  then  $su \rightarrow_{\beta,1} tu$ ,  $us \rightarrow_{\beta,1} ut$  and  $\lambda x.s \rightarrow_{\beta,1} \lambda x.t$ .
- (iii)  **$\beta$ -reduction** ( $\rightarrow_\beta$ ) is the reflexive transitive closure of  $\rightarrow_{\beta,1}$ , i.e. it is the smallest relation that is reflexive, transitive and contains  $\rightarrow_{\beta,1}$ .
- (iv)  **$\beta$ -equivalence** ( $\equiv_\beta$ ) is the smallest equivalence relation containing  $\rightarrow_\beta$ .

**Example 3.6.**

- (i)  $(\lambda x.xxy)(yz) \triangleright_\beta yz(yz)y$

$$\begin{array}{ccc}
\text{(ii)} & (\lambda x.xx)y((\lambda z.yz)(ww)) & \xrightarrow{\beta,1} (\lambda x.xx)y(y(ww)) \\
& \downarrow \beta,1 & \downarrow \beta,1 \\
& yy((\lambda z.yz)(ww)) & \xrightarrow{\beta,1} yy(y(ww))
\end{array}$$

(iii)  $(\lambda x.xx)(\lambda x.xx) \rightarrow_{\beta,1} (\lambda x.xx)(\lambda x.xx)$

**Definition 3.7.** We define the following combinators:

- (i) **I** =  $\lambda x.x$
- (ii) **K** =  $\lambda xy.x$
- (iii) **K**<sub>\*</sub> =  $\lambda xy.y$
- (iv) **S** =  $\lambda xyz.xz(yz)$

**Remark 3.8.** Every term can be defined using  $K$  and  $S$  up to  $\beta$ -equivalence.

**Example 3.9.**  $SKK \rightarrow_{\beta} \lambda z.Kz(Kz) \rightarrow_{\beta} \lambda z.z = I$ .

**Proposition 3.10.** *There exists a **fixed-point combinator**  $Y$  such that  $Yt \rightarrow_{\beta} t(Yt)$ .*

*Proof.* Let  $A := \lambda fx.x(ffx)$  and let  $Y := AA$  be the **Turing operator**. Then  $Yt = AAt \rightarrow_{\beta} t(AAt) = t(Yt)$ .  $\square$

**Definition 3.11.** We can define the **pairing P** :=  $\lambda stx.xst$  and denote **(s, t)** :=  $Pst \rightarrow_{\beta} \lambda x.xst$ .

**Remark 3.12.** Naming this a pairing makes sense because we have  $(s, t)K \rightarrow_{\beta} Kst \rightarrow_{\beta} s$  and  $(s, t)K_* \rightarrow_{\beta} K_*(s, t) \rightarrow_{\beta} t$ .

**Definition 3.13.** We can define **Church numerals**. If  $n$  is a natural number we encode it as **[n]** :=  $\lambda fx.f^n x$  where  $f^0 x := x$  and  $f^{n+1} x = f(f^n x) = f^n(fx)$ .

**Definition 3.14.** Let  $A$  be a set and  $\rightarrow_1$  a binary relation on  $A$  with reflexive transitive closure  $\rightarrow$ .

- (i)  $t \in A$  is **in normal form** if there is no  $s$  such that  $t \rightarrow_1 s$ .
- (ii)  $t \in A$  **has normal form s** if  $s$  is in normal form and  $t \rightarrow s$ .

- (iii)  $t \in A$  is **strongly normalizing** if there exists no infinite sequence  $t \rightarrow_1 t_1 \rightarrow_1 t_2 \rightarrow_1 t_3 \rightarrow_1 \dots$ .
- (iv)  $\rightarrow_1$  is **(weakly) normalizing** if every  $t \in A$  has a normal form.
- (v)  $\rightarrow_1$  is **strongly normalizing (S.N.)** if every  $t \in A$  is strongly normalizing.
- (vi)  $\rightarrow_1$  is **confluent** (has the **Church-Rosser property**) if whenever  $u \leftarrow t \rightarrow v$  there is an  $s \in A$  with  $u \rightarrow s \leftarrow v$ .

$$\begin{array}{ccc} t & \longrightarrow & v \\ \downarrow & & \downarrow \\ u & \dashrightarrow & s \end{array}$$

**Remark 3.15.**  $\beta$ -reduction is neither weakly nor strongly normalizing.

*Proof.* See the counterexamples presented in Example 3.6 (iii) and Proposition 3.10.  $\square$

**Theorem 3.16** (Church-Rosser).  $\rightarrow_{\beta,1}$  is confluent.

**Remark 3.17.** It does not suffice to prove

$$\begin{array}{ccc} t & \xrightarrow{\beta,1} & v \\ \downarrow \beta,1 & & \downarrow \beta \\ u & \dashrightarrow & s \end{array}$$

i.e. for a general binary relation, Theorem 3.16 does not follow from this.

**Definition 3.18.** **Parallel reduction** ( $\Rightarrow$ ) is defined inductively as

- (i)  $x \Rightarrow x$  and  $c \Rightarrow c$  where  $x$  is a variable and  $c$  is a constant.
  - (ii) If  $t \Rightarrow t'$  then  $\lambda x.t \Rightarrow \lambda x.t'$ .
- If  $t \Rightarrow t'$  and  $u \Rightarrow u'$  then
- (iii)  $tu \Rightarrow t'u'$ .
  - (iv)  $(\lambda x.t)u \Rightarrow t'[u'/x]$ .

**Lemma 3.19.** Parallel induction is reflexive, i.e.  $t \Rightarrow t$ .

*Proof.* We show this by induction on  $t$ . Consider  $t = \lambda x.s$ . Then by induction hypothesis  $s \Rightarrow s$ , so by Definition 3.18 (ii)  $\lambda x.s \Rightarrow \lambda x.s$ . The rest of the cases are similar.  $\square$

**Lemma 3.20.** *If  $t \rightarrow_{\beta,1} s$  then  $t \Rightarrow s$ .*

*Proof.* We show this by induction on  $t \rightarrow_{\beta,1} s$ . If  $t \triangleright_{\beta} s$ , say  $(\lambda x.u)v \triangleright_{\beta} u[v/x]$ . Then  $(\lambda x.u)v \Rightarrow u[v/x]$  by Definition 3.18 (iv). The other cases are also easy to show.  $\square$

**Lemma 3.21.** *If  $t \Rightarrow t'$  then  $t \rightarrow_{\beta} t'$ .*

*Proof.* We show this by induction on  $t \Rightarrow t'$ . Suppose the last rule was Definition 3.18 (iv), i.e. concluding  $(\lambda x.t)u \Rightarrow t'[u'/x]$  from  $t \Rightarrow t'$  and  $u \Rightarrow u'$ . By induction hypothesis we have  $t \rightarrow_{\beta} t'$  and  $u \rightarrow_{\beta} u'$ . Then  $(\lambda x.t)u \rightarrow_{\beta} (\lambda x.t')u \rightarrow_{\beta} (\lambda x.t')u' \rightarrow_{\beta,1} t'[u'/x]$ . The other cases are similar.  $\square$

**Lemma 3.22.** *If  $t \Rightarrow t'$  and  $w \Rightarrow w'$  then  $t[w/y] \Rightarrow t'[w'/y]$ .*

*Proof.* We show this by induction on  $t \Rightarrow t'$ . Suppose the last step was Definition 3.18 (iv), i.e. concluding  $(\lambda x.t)u \Rightarrow t'[u'/x]$  from  $t \Rightarrow t'$  and  $u \Rightarrow u'$ . By induction hypothesis we know that  $t[w/y] \Rightarrow t'[w'/y]$  and  $u[w/y] \Rightarrow u'[w'/y]$ . We have to show  $(\lambda x.t[w/y])u[w/y] \Rightarrow t'[u'/x][w'/y]$ .  $x$  is bound so we may assume that  $x$  is not free in  $w'$ . Then one can prove that  $t'[u'/x][w'/y] = t'[w'/y][(u'[w'/y])/x]$ . Now the claim follows from Definition 3.18 (iv). The rest of the cases are similar.  $\square$

**Definition 3.23.** If  $t$  is a term then  $t^*$  is recursively defined as

- (i)  $x^* = x$  and  $c^* = c$  for variables  $x$  and constants  $c$ .
- (ii)  $(\lambda x.t)^* = \lambda x.t^*$ .
- (iii)  $(ts)^* = t^*s^*$  if  $t$  is not a  $\lambda$ .
- (iv)  $((\lambda x.t)s)^* = t^*[s^*/x]$ .

**Lemma 3.24.** *If  $s \Rightarrow t$  then  $t \Rightarrow s^*$ .*

*Proof.* We show this by induction of the length of the derivation for  $s \Rightarrow t$ . If the last step was Definition 3.18 (iii), i.e. concluding  $tu \Rightarrow t'u'$  from  $t \Rightarrow t'$  and  $u \Rightarrow u'$ , then by induction hypothesis  $t' \Rightarrow t^*$  and  $u' \Rightarrow u^*$ . We need to show that  $t'u' \Rightarrow (tu)^*$ .

If  $t$  is not a  $\lambda$  then  $(tu)^* = t^*u^*$ , so we are done by Definition 3.18 (iii). If  $t = \lambda x.s$  then  $(tu)^* = s^*[u^*/x]$ . We know  $\lambda x.s \Rightarrow t'$  which can only be derived using Definition 3.18 (ii). So  $t' = \lambda x.s'$  with  $s \Rightarrow s'$ . By induction hypothesis  $s' \Rightarrow s^*$ . Then  $(\lambda x.s')u' \Rightarrow s^*[u^*/x]$  follows from Definition 3.23 (iv).

If the last step was Definition 3.18 (iv), i.e. concluding  $(\lambda x.t)u \Rightarrow t'[u'/x]$  from  $t \Rightarrow t'$  and  $u \Rightarrow u'$ , then by induction hypothesis we know that  $t' \Rightarrow t^*$  and  $u' \Rightarrow u^*$ . Then  $t'[u'/x] \Rightarrow ((\lambda x.t)u)^* = t^*[u^*/x]$  by Lemma 3.22.

The rest of the cases are easy. □

**Lemma 3.25.** *If  $t \rightarrow_{\beta,1} u$  and  $t \rightarrow_{\beta} v$  then there is an  $s$  with  $u \rightarrow_{\beta} s \leftarrow_{\beta} v$ .*

$$\begin{array}{ccc} t & \xrightarrow{\beta} & v \\ \downarrow \beta,1 & & \downarrow \beta \\ u & \dashrightarrow & s \end{array}$$

*Proof.* Decomposing  $t \rightarrow_{\beta} v$  into individual steps gives us





which with Lemma 3.20 becomes



which with Lemma 3.24 yields



which implies our desired statement since by Lemma 3.21 parallel reduction implies  $\beta$ -reduction and  $\beta$ -reduction is transitive.  $\square$

**Exercise 3.26.** *Derive Theorem 3.16 (Church-Rosser) from Lemma 3.25.*

**Corollary 3.27.** *Every term  $t$  has at most one  $\beta$ -normal form.*

**Corollary 3.28.** *If  $s \equiv_\beta t$ , then there is  $u$  such that  $s \rightarrow_\beta u \leftarrow_\beta t$ .*

### 3.1 Partial recursive functions

**Definition 3.29.** Let  $\mathbf{A} \rightarrow \mathbf{B}$  be the set of **partial functions** from  $A$  to  $B$ , i.e. a partial function from  $A$  to  $B$  is a pair  $(X, f)$  such that  $X \subseteq A$  and  $f: X \rightarrow B$ . We set  $\text{dom}(\mathbf{f}) := X$  and call it the **domain** of  $f$ .

**Notation 3.30.** We set  $\mathbf{f}(\vec{x})\downarrow := (\vec{x} \in \text{dom}(f))$  and  $\mathbf{f}(\vec{x})\uparrow := (\vec{x} \notin \text{dom}(f))$ .

**Definition 3.31.** The set of **partial recursive functions (P.R. functions)** is the subset of  $\bigcup_{k \geq 0} (\mathbb{N}^k \rightarrow \mathbb{N})$  generated by

- (i)  $0: \mathbb{N}^0 \rightarrow \mathbb{N}$  is P.R.
- (ii)  $\text{Succ}: \mathbb{N} \rightarrow \mathbb{N}, x \mapsto x + 1$  is P.R.
- (iii) The projection  $P_i^k: \mathbb{N}^k \rightarrow \mathbb{N}, P_i^k(x_0, \dots, x_{k-1}) = x_i$  is P.R. for  $0 \leq i < k$ .
- (iv) Composition: if  $f: \mathbb{N}^k \rightarrow \mathbb{N}$  is P.R. and  $g_0, \dots, g_{k-1}: \mathbb{N}^l \rightarrow \mathbb{N}$  are P.R. then  $f \circ \vec{g}: \mathbb{N}^l \rightarrow \mathbb{N}, \vec{x} \mapsto f(g_0(\vec{x}), \dots, g_{k-1}(\vec{x}))$  is P.R. and  $\vec{x} \in \text{dom}(f \circ \vec{g})$  iff  $\vec{x} \in \text{dom}(g_i)$  for all  $i$  and  $(g_0(\vec{x}), \dots, g_{k-1}(\vec{x})) \in \text{dom}(f)$ .
- (v) Primitive recursion: if  $f_0: \mathbb{N}^k \rightarrow \mathbb{N}$  and  $f_s: \mathbb{N}^{k+2} \rightarrow \mathbb{N}$  are P.R. then the function  $g$  defined by  $g(0, \vec{x}) := f_0(\vec{x})$  and  $g(n+1, \vec{x}) := f_s(n, g(n, \vec{x}), \vec{x})$  is P.R.,  $g(0, \vec{x})\downarrow$  iff  $f_0(\vec{x})\downarrow$ , and  $g(n+1, \vec{x})\downarrow$  iff  $g(n, \vec{x})\downarrow$  and  $f_s(n, g(n, \vec{x}), \vec{x})\downarrow$ .
- (vi) Minimization (unbounded search): if  $f: \mathbb{N}^{k+1} \rightarrow \mathbb{N}$  is P.R. then  $\mu_f: \mathbb{N}^k \rightarrow \mathbb{N}$  is P.R. where we set  $\mu_f(\vec{x}) = n$  iff  $f(i, \vec{x})\downarrow$  for  $i \leq n$ ,  $f(i, \vec{x}) > 0$  for  $i < n$  and  $f(n, \vec{x}) = 0$ , and we set  $\mu_f(\vec{x})\uparrow$  if no such  $n$  exists.

**Remark 3.32.**

- (i) The class of functions closed under the rules (i) to (iv) of Definition 3.31 are called the **primitive recursive functions**. All of them are total.
- (ii) The partially recursive functions that are total are called **(totally) recursive**. Not all of them are primitive recursive functions.
- (iii) The Church-Turing thesis is the claim that the partially recursive functions are precisely the functions that capture the intuitive notion of “computability”.

**Example 3.33.**

- (i)  $(x, y) \mapsto x + y$ ,  $(x, y) \mapsto x \cdot y$  and  $(x, y) \mapsto x^y$  are all primitive recursive.

- (ii) The **Ackermann function** is a total partial recursive function that is not primitive recursive.

**Definition 3.34.** A  $\lambda$ -term  $t$  **represents**  $f: \mathbb{N}^k \rightarrow \mathbb{N}$  if for all  $\vec{n} \in \mathbb{N}^k$  we have  $t[n_0] \dots [n_{k-1}] \rightarrow_{\beta} [f(\vec{n})]$  if  $f(\vec{n}) \downarrow$  and that  $t[n_0] \dots [n_{k-1}]$  has no normal form if  $f(\vec{n}) \uparrow$ .

**Proposition 3.35.** If  $f: \mathbb{N}^k \rightarrow \mathbb{N}$  is P.R. then there are primitive recursive functions  $u: \mathbb{N} \rightarrow \mathbb{N}$  and  $t: \mathbb{N}^{k+1} \rightarrow \mathbb{N}$  such that  $f = u \circ \mu_t$ .

**Theorem 3.36.**  $f: \mathbb{N}^k \rightarrow \mathbb{N}$  is P.R. iff it is represented by some  $\lambda$ -term.

*Proof sketch.* For the backward direction of the proof, first notice that primitive recursive functions are expressive enough to encode: pairs of natural numbers, finite sequences of natural numbers,  $\lambda$ -terms, substitution in  $\lambda$ -terms and  $\beta$ -reduction. Then we can show that there is a partial recursive function that searches for a reduction sequence  $t \rightarrow_{\beta,1} t_1 \rightarrow_{\beta,1} t_2 \rightarrow_{\beta,1} \dots \rightarrow_{\beta,1} [m]$ .

For the forward direction we first show that the claim is true for a primitive recursive function  $f$ . We now proceed inductively over the first five constructors of Definition 3.31. For (i) we see that  $[0]$  represents  $0: \mathbb{N}^0 \rightarrow \mathbb{N}$ . Constructor (ii) is an exercise. For (iii) we can show that the projection  $P_i^k$  is represented by  $\lambda x_0 x_1 \dots x_{k-1}. x_i$ . For (iv) assume that  $F$  represents  $f$  and  $G_i$  represents  $g_i$  for all  $i$ . Then  $f \circ \vec{g}$  is represented by  $\lambda \vec{x}. F(G_0 \vec{x})(G_1 \vec{x}) \dots (G_{k-1} \vec{x})$ . Lastly, for (v) suppose that  $F_0$  represents  $f_0$  and  $F_s$  represents  $f_s$ . We want to find  $G$  such that

$$G[0] \vec{x} \equiv_{\beta} F_0 \vec{x} \qquad G[n+1] \vec{x} \equiv_{\beta} F_s[n](G[n] \vec{x}) \vec{x}$$

This is sufficient because  $G[0][\vec{n}] \equiv_{\beta} F_0[\vec{n}] \rightarrow_{\beta} [f_0(\vec{n})]$  so  $G[0][\vec{n}] \rightarrow_{\beta} [f_0(\vec{n})]$  since  $[f_0(\vec{n})]$  is in normal form. It is even enough to show that

$$G[0] \equiv_{\beta} F_0 \qquad G[n+1] \equiv_{\beta} F_s[n](G[n])$$

by  $\lambda$ -abstraction. We want to try to encode the sequence  $([0], G[0]), ([1], G[1]), \dots$ . We set

$$T := \lambda u. (\text{Succ}(uK), F_s(uK)(uK_*))$$

Note that  $T([n], t) \rightarrow_{\beta} (\text{Succ}([n]), F_s[n]t) \rightarrow_{\beta} ([n+1], F_s[n]t)$ . We will show later that

means that  $T$  produces the next pair in the sequence. So we want to iterate  $T$ . Let

$$G := \lambda v.vT([0], F_0)K_*$$

Then

$$G[0] \rightarrow_\beta [0]T([0], F_0)K_* \rightarrow_\beta ([0], F_0)K_* \rightarrow_\beta F_0$$

We want to show by induction that  $[n]T([0], F_0) \equiv_\beta ([n], G[n])$ . For the case  $n = 0$  we get  $[0]T([0], F_0) \rightarrow_\beta ([0], F_0) \beta \leftarrow ([0], G[0])$ . Thus  $[0]T([0], F_0) \equiv_\beta ([0], G[0])$ . For  $n + 1$  we get

$$[n + 1]T([0], F_0) \rightarrow_\beta T([n]T([0], F_0)) \stackrel{IH}{\equiv}_\beta T([n], G[n]) \rightarrow_\beta ([n + 1], F_s[n](G[n]))$$

and

$$G[n + 1] \rightarrow_\beta [n + 1]T([0], F_0)K_* \equiv_\beta ([n + 1], F_s[n](G[n]))K_* \rightarrow_\beta F_s[n](G[n])$$

Thus  $[n + 1]T([0], F_0) \equiv_\beta ([n + 1], G[n + 1])$  and consequently  $G[n + 1] \equiv_\beta F_s[n](G[n])$ .

Now onto the general case. For minimization we use the Turing operator  $Y$  from the proof of Proposition 3.10 and the term  $D$  with  $Dst[0] \rightarrow_\beta s$  and  $Dst[n + 1] \rightarrow_\beta t$ . (The existence of  $D$  is an exercise.) Let  $T$  be a term and  $\vec{x}$  be a sequence of variables. We define

$$W := Y(\lambda vy.Dy(v(\text{Succ}(y)))(Ty\vec{x}))$$

Then we get

$$W[n] \rightarrow_\beta D[n](W[n + 1])(T[n]\vec{x}) \rightarrow_\beta \begin{cases} [n] & \text{if } T[n]\vec{x} \rightarrow_\beta [0] \\ W[n + 1] & \text{if } T[n]\vec{x} \rightarrow_\beta [m + 1] \text{ for some } m \in \mathbb{N} \end{cases}$$

By the previous proposition we can take primitive recursive functions  $u$  and  $t$  such that  $f = u \circ \mu_t$ . Suppose that  $U$  represents  $u$  and  $T$  represents  $t$ . Now we set

$$F := \lambda \vec{x}.U(W[0])$$

If  $f(\vec{x}) \downarrow$  then

$$F[\vec{x}] \rightarrow_\beta U(W[0]) \rightarrow_\beta U([\mu_t(\vec{x})]) \rightarrow_\beta [f(\vec{x})]$$

and if  $f(\vec{x})\uparrow$  then

$$F[\vec{x}] \rightarrow_{\beta} U(W[0]) \rightarrow_{\beta} U(W[1]) \rightarrow_{\beta} \dots$$

which is an infinite reduction sequence. We would have to show that no way of reduction leads to a normal form. This is true but we omit the proof.  $\square$

**Theorem 3.37** (Normalization theorem). *If  $t$  has a  $\beta$ -normal form, then iterated contraction of the leftmost  $\beta$ -redex leads to its normal form.*

## 3.2 Simply typed $\lambda$ -calculus

We want to introduce type judgements such as  $t : \sigma \rightarrow \tau$  specifying the behaviour of  $t$ . For example, for  $ts$  to make sense we need  $t : \sigma \rightarrow \tau$  and  $s : \sigma$ . We define two versions of simply-typed  $\lambda$ -calculus  $\lambda_{\rightarrow}^{\text{Curry}}$  and  $\lambda_{\rightarrow}^{\text{Church}}$ .

**Definition 3.38.**  $\lambda_{\rightarrow}^{\text{Curry}}$  consists of the following.

- (i) For **types**, let  $\{\alpha, \beta, \gamma, \dots\}$  be an infinite set of **type variables** and  $\{B, C, \dots\}$  a set of **type constants**. The types are then given by

$$\sigma, \tau ::= \alpha \mid C \mid (\sigma \rightarrow \tau)$$

where  $\alpha$  is a type variable and  $C$  a type constant.

- (ii) As **preterms** or **raw terms** we have precisely the terms of the  $\lambda$ -calculus.
- (iii) A **context**  $\Gamma$  is a finite set  $\{x_1 : \tau_1, \dots, x_n : \tau_n\}$  where the  $x_i$  are distinct term-variables and the  $\tau_i$  are types.  $\Gamma$  is a partial function with  $\text{dom}(\Gamma) := \{x_1, \dots, x_n\}$  and  $\Gamma(x_i) = \tau_i$ .
- (iv) A **typing judgement** is a triple  $\Gamma \vdash t : \tau$  (read: “ $t$  is a (well-typed) term of type  $\tau$  in the context  $\Gamma$ .”) generated by:

- (a) VAR:  $\frac{}{\Gamma, x : \tau \vdash x : \tau}$  where  $x \notin \text{dom}(\Gamma)$ .
- (b) APP:  $\frac{\Gamma \vdash t : \sigma \rightarrow \tau \quad \Gamma \vdash s : \sigma}{\Gamma \vdash ts : \tau}$
- (c) ABS:  $\frac{\Gamma, x : \sigma \vdash t : \tau}{\Gamma \vdash \lambda x. t : \sigma \rightarrow \tau}$  where  $x \notin \text{dom}(\Gamma)$ .

**Remark 3.39.**

- (i) With  $\Gamma, x : \tau$  we mean  $\Gamma \cup \{x : \tau\}$ .

(ii) We write  $\vdash t : \tau$  for  $\emptyset \vdash t : \tau$ .

(iii) We write  $\sigma \rightarrow \tau \rightarrow \varrho$  for  $\sigma \rightarrow (\tau \rightarrow \varrho)$ , which is different from  $(\sigma \rightarrow \tau) \rightarrow \varrho$ .

**Example 3.40.** For any types  $\sigma$  and  $\tau$  and  $K$  as in Definition 3.7 we get

$$\begin{array}{c} \text{VAR} \frac{}{x : \sigma, y : \tau \vdash x : \sigma} \\ \text{ABS} \frac{}{x : \sigma \vdash \lambda y. x : \tau \rightarrow \sigma} \\ \text{ABS} \frac{}{\vdash K : \sigma \rightarrow \tau \rightarrow \sigma} \end{array}$$

So e.g. for  $\sigma = (\varrho \rightarrow \varrho)$  and  $\tau = \varrho$  we get  $\vdash (\varrho \rightarrow \varrho) \rightarrow \varrho \rightarrow \varrho \rightarrow \varrho$ . Thus one term can have multiple type judgements.

**Example 3.41.** One can show that  $\vdash [n] : (\sigma \rightarrow \sigma) \rightarrow \sigma \rightarrow \sigma$ .

**Definition 3.42.**  $\lambda_{\rightarrow}^{\text{Curry}} + \mathbb{N}$  is an extension of  $\lambda_{\rightarrow}^{\text{Curry}}$  with

- (i) one constant type  $\mathbb{N}$  (or Nat)
- (ii) constant terms 0, Succ and  $R_{\tau}$  for any type  $\tau$
- (iii) three additional typing rules:
  - (a)  $\frac{}{\vdash 0 : \mathbb{N}}$
  - (b)  $\frac{}{\vdash \text{Succ} : \mathbb{N} \rightarrow \mathbb{N}}$
  - (c)  $\frac{}{\vdash R_{\tau} : \tau \rightarrow (\mathbb{N} \rightarrow \tau \rightarrow \tau) \rightarrow \mathbb{N} \rightarrow \tau}$

Additionally, we define:

- (i)  **$\iota$ -contraction** ( $\triangleright_{\iota}$ ) as  $R_{\tau} x f 0 \triangleright_{\iota} x$  and  $R_{\tau} x f \text{Succ}(n) \triangleright_{\iota} f n (R_{\tau} x f n)$ .
- (ii) **one-step- $\iota$ -reduction** ( $\rightarrow_{\iota,1}$ ) as the compatible closure of  $\iota$ -contraction
- (iii) **one-step- $\beta$ - $\iota$ -reduction** ( $\rightarrow_{\beta\iota,1}$ ) as the compatible closure of  $\iota$ - and  $\beta$ -contraction
- (iv)  **$\iota$ -reduction** ( $\rightarrow_{\iota}$ ) as reflexive transitive closure of one-step- $\iota$ -reduction
- (v)  **$\beta$ - $\iota$ -reduction** ( $\rightarrow_{\beta\iota}$ ) as the reflexive transitive closure of one-step- $\beta$ - $\iota$ -reduction

**Lemma 3.43.**

- (i) In  $\lambda_{\rightarrow}^{\text{Curry}}$ , if  $\Gamma \vdash t : \tau$  then  $\text{fv}(t) \subseteq \text{dom}(\Gamma)$ .

(ii) If  $\Gamma \vdash t : \tau$  and  $\Gamma'$  is context such that

$$\forall x \in \text{fv}(t), \Gamma(x) = \Gamma'(x) \quad (*)$$

then  $\Gamma' \vdash t : \tau$

*Proof.* We prove both statements by induction on  $\Gamma \vdash t : \tau$ . We only show ABS. Suppose the last rule was

$$\frac{\Gamma, x : \sigma \vdash t' : \tau}{\Gamma \vdash \lambda x. t' : \sigma \rightarrow \tau}$$

For (i), we know by induction hypothesis that  $\text{fv}(t') \subseteq \text{dom}(\Gamma, x : \sigma) = \text{dom}(\Gamma) \cup \{x\}$ . Then  $\text{fv}(\lambda x. t') = \text{fv}(t') \setminus \{x\} \subseteq \text{dom}(\Gamma)$ . For (ii), let  $\Gamma'$  be another context satisfying (\*). Then for all  $y \in \text{fv}(t')$ , we have  $(\Gamma, x : \sigma)(y) = (\Gamma', x : \sigma)(y)$ . So by induction hypothesis  $\Gamma', x : \sigma \vdash t' : \tau$  and then with ABS we get  $\Gamma' \vdash \lambda x. t' : \sigma \rightarrow \tau$ .  $\square$

**Lemma 3.44** (Generation).

(i) If  $\Gamma \vdash x : \tau$  then  $\Gamma(x) = \tau$ .

(ii) If  $\Gamma \vdash ts : \tau$  then for some type  $\sigma$  we have that  $\Gamma \vdash t : \sigma \rightarrow \tau$  and  $\Gamma \vdash s : \sigma$ .

(iii) If  $\Gamma \vdash \lambda x. t : \tau$  then for some  $\tau_1$  and  $\tau_2$ , we have  $\tau = \tau_1 \rightarrow \tau_2$  and  $\Gamma, x : \tau_1 \vdash t : \tau_2$ .

*Proof.* We show this by induction on the derivation of the typing judgement. Only one rule can apply in each case.  $\square$

**Definition 3.45.** We can define **substitution** on types  $\tau[\sigma/\alpha]$  as

(i)  $\alpha[\sigma/\alpha] := \sigma$

(ii)  $\beta[\sigma/\alpha] := \beta$  if  $\beta \neq \alpha$

(iii)  $(\tau_1 \rightarrow \tau_2)[\sigma/\alpha] := \tau_1[\sigma/\alpha] \rightarrow \tau_2[\sigma/\alpha]$

**Proposition 3.46.**

(i) *Type substitution:* if  $\Gamma \vdash t : \tau$  then  $\Gamma[\sigma/\alpha] \vdash t : \tau[\sigma/\alpha]$ .

(ii) *Term substitution:* if  $\Gamma, x : \sigma \vdash t : \tau$  and  $\Gamma \vdash s : \sigma$  then  $\Gamma \vdash t[s/x] : \tau$ .

(iii) *Subject reduction: if  $\Gamma \vdash t : \tau$  and  $t \rightarrow_\beta t'$  then  $\Gamma \vdash t' : \tau$ .*

*Proof.* Proving (i) and (ii) is an exercise. We show (iii) by induction on  $t \rightarrow_\beta t'$ . We only consider the case  $t \triangleright_\beta t'$ , i.e.  $t = (\lambda x.t_1)t_2$  and  $t' = t_1[t_2/x]$ . So  $\Gamma \vdash (\lambda x.t_1)t_2 : \tau$ . By generation (Lemma 3.44) we have  $\Gamma \vdash \lambda x.t_1 : \sigma \rightarrow \tau$  and  $\Gamma \vdash t_2 : \sigma$ . So  $\Gamma x : \sigma \vdash t_1 : \tau$ , and thus by term substitution  $\Gamma \vdash t_1[t_2/x] : \tau$ .  $\square$

**Remark 3.47.** If  $t \rightarrow_\beta t'$  and  $\Gamma \vdash t' : \sigma$ , then  $t$  is not necessarily typeable.

**Definition 3.48.**  $\Gamma \vdash t : \tau$  is a **principal typing judgement** if for any  $\Gamma'$  and  $\tau'$  with  $\Gamma' \vdash t : \tau'$  there is a type substitution  $S = [\sigma_1/\alpha_1, \dots, \sigma_k/\alpha_k]$  with  $S(\tau) = \tau'$  and  $S(\Gamma) \subseteq \Gamma'$ .

**Theorem 3.49.** *If  $t$  is well-typed, then  $t$  has a principal typing judgement.*

**Example 3.50.** Let's find a principal typing judgement for  $S = \lambda xyz.xz(yz)$ . First, assign to each  $\lambda$ -abstraction a type variable:  $x : \alpha_x$ ,  $y : \alpha_y$  and  $z : \alpha_z$ . Then consider the constraints to these variables that follow from the application: from  $xz$  we know  $\alpha_x = (\alpha_z \rightarrow \beta)$ , from  $yz$  we  $\alpha_y = (\alpha_z \rightarrow \gamma)$  and from  $(xz)(yz)$  we know  $\beta = (\gamma \rightarrow \varrho)$ . Then we get

$$x : \alpha_z \rightarrow \gamma \rightarrow \varrho, y : \alpha_z \rightarrow \gamma, z : \alpha_z \vdash (xz)(yz) : \varrho$$

which using abstraction gives us

$$\vdash S : (\alpha_z \rightarrow \gamma \rightarrow \varrho) \rightarrow (\alpha_z \rightarrow \gamma) \rightarrow \alpha_z \rightarrow \varrho$$

One can show that this is a principal judgement.

**Theorem 3.51.** *There are (polynomial-time) algorithms for*

- (i) *typeability: given  $t$ , construct  $\Gamma$  and  $\tau$  such that  $\Gamma \vdash t : \tau$ .*
- (ii) *type-checking: given  $t$ ,  $\Gamma$  and  $\tau$  does  $\Gamma \vdash t : \tau$  hold?*

**Definition 3.52.**  $\lambda_{\rightarrow}^{\text{Church}}$  has the same types as  $\lambda_{\rightarrow}^{\text{Curry}}$ . We change the **preterms** to be

$$s, t ::= x \mid c \mid (st) \mid \lambda x : \tau. t$$



where  $x$  is a variable,  $c$  a constant and  $\tau$  a type. Additionally we keep the **typing judgements** VAR and APP but change ABS to be

$$\frac{\Gamma, x : \sigma \vdash t : \tau}{\Gamma \vdash \lambda x : \sigma. t : \sigma \rightarrow \tau}$$

where  $x \notin \text{dom}(\Gamma)$ .

**Remark 3.53.** We have substitution and  $\beta$ -reduction as usual, i.e.  $(\lambda x : \tau. t)s \triangleright_{\beta} t[s/x]$  generates  $\beta$ -reduction.

**Remark 3.54.** Term substitution and subject reduction hold the same way as before.

**Proposition 3.55** (Unique typing). *In  $\lambda_{\rightarrow}^{\text{Church}}$ , if  $\Gamma \vdash t : \tau$  and  $\Gamma \vdash t : \sigma$  then  $\sigma = \tau$ .*

*Proof.* We prove this by induction on  $\Gamma \vdash t : \tau$ . Showing it for VAR and APP is easy. If the last rule was ABS, i.e.

$$\frac{\Gamma, x : \sigma' \vdash t' : \tau'}{\Gamma \vdash \lambda x : \sigma'. t' : \sigma' \rightarrow \tau'}$$

with  $\tau = (\sigma' \rightarrow \tau')$  and  $t = \lambda x : \sigma'. t'$ , then  $\Gamma \vdash t : \sigma$  must have ABS as its last rule.  $t$  specifies that the domains are equal. By induction hypothesis the codomains are equal.  $\square$

**Remark 3.56.** We write  $t^{\tau}$  to say  $t : \tau$  in an implicitly given context.

**Theorem 3.57.**  $\rightarrow_{\beta,1}$  is strongly normalizing among well-typed terms in both  $\lambda_{\rightarrow}^{\text{Curry}}$  and  $\lambda_{\rightarrow}^{\text{Church}}$ .

**Exercise 3.58.** Prove that the two claims in the previous theorem are equivalent.

**Lemma 3.59.** A term  $t$  (in  $\lambda_{\rightarrow}^{\text{Church}}$ ) is strongly normalizing iff there is a natural reduction number  $n$  such that every reduction sequence  $t \rightarrow_{\beta,1} t_1 \rightarrow_{\beta,1} t_2 \rightarrow_{\beta,1} \dots$  has length at most  $n$ .

For a strongly normalizing term  $t$ , we write  $h(t)$  for the maximal length of a reduction sequence starting with  $t$ .

*Proof.* The backward direction is trivial. For the forward direction consider  $\downarrow t := \{s \mid t \rightarrow_{\beta} s\}$  and let  $A := \{s \in \downarrow t \mid \downarrow s \text{ is finite}\}$ .

**Claim 1:** Let  $s \in \downarrow t$ . If for every  $s \rightarrow_{\beta,1} s'$  we have  $s' \in A$  then  $s \in A$ .

*Proof.* The claim follows from the following observations:

- (i)  $\#\{s' \mid s \rightarrow_{\beta,1} s'\}$  is finite because  $s$  has finitely many  $\beta$ -redexes.
- (ii)  $\downarrow s = \{s\} \cup \bigcup_{s \rightarrow_{\beta,1} s'} \downarrow s'$

By assumption  $\downarrow s'$  is finite for all  $s \rightarrow_{\beta,1} s'$ . Thus  $\downarrow s$  is a finite union of finite sets and therefore  $s \in A$ .

**Claim 2:**  $t \in A$

*Proof.* Suppose  $t \notin A$ . Then  $t \rightarrow_{\beta,1} t_1$  with  $t_1 \notin A$  and  $t_1 \rightarrow_{\beta,1} t_2$  with  $t_2 \notin A$ . Repeating this we get

$$t \rightarrow_{\beta,1} t_1 \rightarrow_{\beta,1} t_2 \rightarrow_{\beta,1} \dots$$

which contradicts that  $t$  is strongly normalizing. So  $t \in A$ .

Take  $n := \#A$ . Then the claim of the lemma follows. □

**Definition 3.60.** We define the **computable** terms of type  $\tau$  which we write  $C(\tau)$  by recursion on  $\tau$ :

- (i) if  $\tau$  is a variable or a constant, then  $t \in C(\tau)$  iff  $t : \tau$  and  $t$  is strongly normalizing.
- (ii)  $t \in C(\sigma \rightarrow \tau)$  iff  $t : \sigma \rightarrow \tau$  and for every  $s \in C(\sigma)$  we have  $ts \in C(\tau)$ .

**Lemma 3.61.**

- (i) If  $t \in C(\tau)$  then  $t$  is strongly normalizing.
- (ii) If  $t \in C(\tau)$  and  $t \rightarrow_{\beta} t'$ , then  $t' \in C(\tau)$ .
- (iii) Suppose  $t : \tau$  and  $t$  is not a  $\lambda$ -abstraction. If

$$\forall t', (t \rightarrow_{\beta,1} t' \implies t' \in C(\tau)) \tag{*}$$

then  $t \in C(\tau)$ .

*Proof.* We do a simultaneous proof of (i) - (iii) by induction on  $\tau$ . If  $\tau$  is a variable or constant, then (i) - (iii) are easy. Suppose  $t : \sigma \rightarrow \tau$ .

For (i) assume that  $t \in C(\sigma \rightarrow \tau)$ . We need to prove that  $t$  is strongly normalizing. Let  $x : \sigma$ , then by induction hypothesis (iii),  $x \in C(\sigma)$ . So now  $tx \in C(\tau)$  by definition. By induction hypothesis (i),  $tx$  is strongly normalizing. Therefore,  $t$  is strongly normalizing. Indeed, if  $t \rightarrow_{\beta,1} t_1 \rightarrow_{\beta,1} t_2 \rightarrow_{\beta,1} \dots$  then  $tx \rightarrow_{\beta,1} t_1x \rightarrow_{\beta,1} t_2x \rightarrow_{\beta,1} \dots$ .

For (ii) assume  $t \in C(\sigma \rightarrow \tau)$  and  $t \rightarrow_{\beta,1} t'$ . To show that  $t' \in C(\sigma \rightarrow \tau)$ , take  $s \in C(\sigma)$ . We now need to prove that  $t's \in C(\tau)$ . By definition we know  $ts \in C(\tau)$ , so by induction hypothesis (ii) and the fact that  $ts \rightarrow_{\beta} t's$ , we conclude that  $t's \in C(\tau)$ .

For part (iii), let  $t : \sigma \rightarrow \tau$  satisfy  $(*)$  and not be a  $\lambda$ -abstraction. Let  $s \in C(\sigma)$ . We need to show that  $ts \in C(\tau)$ . By induction hypothesis (i),  $s$  is strongly normalizing. We prove  $ts \in C(\tau)$  by strong induction on  $h(s)$ . We show that  $(*)$  hold for  $ts$ . There are two ways that  $ts$  can reduce: firstly,  $ts \rightarrow_{\beta,1} t's$  where  $t \rightarrow_{\beta,1} t'$  or secondly,  $ts \rightarrow_{\beta,1} ts'$  where  $s \rightarrow_{\beta,1} s'$ . For the first option we know  $t' \in C(\sigma \rightarrow \tau)$  because  $t$  satisfies  $(*)$ . By definition,  $t's \in C(\tau)$ . For the second option, if  $s \rightarrow_{\beta,1} s'$ , then by the induction hypothesis of the nested induction and  $h(s') < h(s)$ , we know  $ts' \in C(\tau)$ . This finishes (iii).  $\square$

**Lemma 3.62.** Suppose  $t : \tau$ ,  $x : \sigma$  and

$$\forall x \in C(\sigma), t[s/x] \in C(\tau) \quad (*)$$

Then  $\lambda x : \sigma. t \in C(\sigma \rightarrow \tau)$ .

*Proof.* Suppose  $t$  satisfies  $(*)$ . By definition, if  $s \in C(\sigma)$ , we need to show  $(\lambda x : \sigma. t)s \in C(\tau)$ . By  $(*)$ ,  $t \in C(\tau)$ . Indeed, take  $s = x$  and use  $t[x/x] = t$ . By Lemma 3.61 (ii),  $t$  and  $s$  are strongly normalizing. We prove  $(\lambda x : \sigma. t)s \in C(\tau)$  by induction on  $h(s) + h(t)$ . We want to use Lemma 3.61 (iii), so assume  $(\lambda x : \sigma. t)s \rightarrow_{\beta,1} r$ . We need to show  $r \in C(\tau)$ . There are three cases:

(i)  $r = (\lambda x : \sigma. t)s'$  where  $s \rightarrow_{\beta,1} s'$

(ii)  $r = (\lambda x : \sigma. t')s$  where  $t \rightarrow_{\beta,1} t'$

(iii)  $r = t[s/x]$

For (i), since  $h(s') < h(s)$ , by induction hypothesis  $r \in C(\tau)$ .

For (ii) we have  $h(t) < h(t')$ . To conclude by induction hypothesis we have to show that  $t'$  satisfies  $(*)$ . So let  $s \in C(\sigma)$ . Since  $t$  satisfies  $(*)$  we know  $t[s/x] \in C(\tau)$ . So by Lemma 3.61 (ii) and  $t[s/x] \rightarrow_{\beta} t'[s/x]$  we have  $t'[s/x] \in C(\tau)$ .

For (iii)  $t$  satisfies  $(*)$ , which gives us  $t[s/x] \in C(\tau)$ .  $\square$

**Lemma 3.63.** *If  $t \in C(\tau)$ ,  $x_0 : \sigma_0, \dots, x_{k-1} : \sigma_{k-1}$  are variables and  $s_0 \in C(\sigma_0), \dots, s_{k-1} \in C(\sigma_{k-1})$ . Then  $t[\vec{s}/\vec{x}] = t[s_0/x_0] \dots [s_{k-1}/x_{k-1}] \in C(\tau)$ .*

*Proof.* We show this by induction on  $t$ . If  $t$  is a variable or a constant, this is easy. If  $t = \lambda x'. t'$ , then by Lemma 3.44 (iii)  $\tau = (\sigma' \rightarrow \tau')$ ,  $x' : \tau'$  and  $t' : \tau'$  for some  $\sigma'$  and  $\tau'$ . We want to show  $(\lambda x'. t')[\vec{s}/\vec{x}] = \lambda x'. (t'[\vec{s}/\vec{x}]) \in C(\sigma' \rightarrow \tau')$ . By Lemma 3.62 it is enough to show that for every  $s' \in C(\sigma)$ ,  $t'[\vec{s}/\vec{x}][s'/x']$ . This follows by induction hypothesis. Lastly, let us consider  $t = t's'$ . By Lemma 3.44 (ii) there is  $\sigma$  such that  $t' : \sigma \rightarrow \tau$  and  $s' : \sigma$ . Then  $t[\vec{s}/\vec{x}] = (t'[\vec{s}/\vec{x}]) (s'[\vec{s}/\vec{x}])$ . By induction hypothesis on  $t'$  and  $s'$ , we know that  $t'[\vec{s}/\vec{x}] \in C(\sigma \rightarrow \tau)$  and  $s'[\vec{s}/\vec{x}] \in C(\sigma)$ . By definition,  $t[\vec{s}/\vec{x}] \in C(\tau)$ .  $\square$

*Proof of Theorem 3.57.* Let  $t : \tau$ . By Lemma 3.61 (i) it is enough to show that  $t \in C(\tau)$ . But this is true by the previous lemma for  $k = 0$ .  $\square$

**Remark 3.64.** Theorem 3.57 implies that for well-typed terms  $s$  and  $t$  we can decide  $s \equiv_\beta t$  by reducing both  $s$  and  $t$  to their normal forms and checking whether those are the same.

## 4 Simple Type Theory

Simple type theory was first presented by Church in 1940.

**Definition 4.1.** We add to  $\lambda_{\rightarrow}^{\text{Church}}$  the type constants

(i)  $I$  (“**individuals**”)

(ii)  $\text{Prop}$

and the term constants

(i)  $\text{Forall}_{\tau} : (\tau \rightarrow \text{Prop}) \rightarrow \text{Prop}$

(ii)  $\Rightarrow : \text{Prop} \rightarrow \text{Prop} \rightarrow \text{Prop}$

(iii)  $=_{\tau} : \tau \rightarrow \tau \rightarrow \text{Prop}$

(iv)  $\varepsilon_{\tau} : (\tau \rightarrow \text{Prop}) \rightarrow \tau$

We view  $A : \text{Prop}$  as a formula. We write:

(i)  $\forall x : \tau. A$  for  $\text{Forall}_{\tau}(\lambda x : \tau. A)$

(ii)  $A \Rightarrow B : \text{Prop}$  for  $\Rightarrow(A)(B)$  where  $A, B : \text{Prop}$

(iii)  $s =_{\tau} t : \text{Prop}$  for  $=_{\tau}(s)(t)$  where  $s, t : \tau$

Now we can define:

(i)  $\perp := \forall P : \text{Prop}. P$

(ii)  $\top := \perp \rightarrow \perp$

(iii)  $\neg \mathbf{A} := A \Rightarrow \perp$

(iv)  $\mathbf{A} \vee \mathbf{B} := \neg A \Rightarrow B$

(v)  $\mathbf{A} \wedge \mathbf{B} := \neg(\neg A \vee \neg B)$

(vi)  $\mathbf{A} \Leftrightarrow \mathbf{B} := (A \Rightarrow B) \wedge (B \Rightarrow A)$

(vii)  $\exists \mathbf{x} : \tau. \mathbf{A} := \neg(\forall x : \tau. \neg A)$

**Remark 4.2.** See sheet 6 exercise 6 for a different (equivalent) way to define  $A \wedge B$ ,  $A \vee B$  and  $\exists x : \tau. A$ .

**Definition 4.3.** In addition to the typing judgements  $\Gamma \vdash t : \tau$  we can now define **provability judgements**  $\Delta \vdash_{\mathbf{r}} \mathbf{A}$  (or  $\Delta \vdash A$  for short) where  $\Delta$  is a set of terms  $B$  such that  $\Gamma \vdash B : \text{Prop}$  and  $\Gamma \vdash A : \text{Prop}$ . This gives a proof system with the following rules where the typing constraints are marked in gray:

- (i) (Ass)  $\frac{}{\Delta, A \vdash A}$
- (ii) ( $\Rightarrow$  I)  $\frac{\Delta, A \vdash B}{\Delta \vdash A \Rightarrow B}$
- (iii) ( $\Rightarrow$  E)  $\frac{\Delta \vdash A \Rightarrow B \quad \Delta \vdash A}{\Delta \vdash B}$
- (iv) ( $\forall$  I)  $\frac{\Delta \vdash A \quad x \notin \text{fv}(\Delta) \quad \Gamma \vdash x : \tau}{\Delta \vdash \forall x : \tau. A}$
- (v) ( $\forall$  E)  $\frac{\Delta \vdash \forall x : \tau. A \quad \Gamma \vdash t : \tau}{\Delta \vdash A[t/x]}$
- (vi) (= Refl)  $\frac{\Gamma \vdash t : \tau}{\Delta \vdash t =_{\tau} t}$
- (vii) (= Symm)  $\frac{\Delta \vdash t =_{\tau} s}{\Delta \vdash s =_{\tau} t}$
- (viii) (= Trans)  $\frac{\Delta \vdash t =_{\tau} s \quad \Delta \vdash s =_{\tau} r}{\Delta \vdash t =_{\tau} r}$
- (ix) (App Congr)  $\frac{\Delta \vdash t =_{\sigma \rightarrow \tau} t' \quad \Delta \vdash s =_{\sigma} s'}{\Delta \vdash ts =_{\tau} t's'}$
- (x) ( $\beta$  Reduce)  $\frac{\Gamma, x : \sigma \vdash t : \tau \quad \Gamma \vdash s : \sigma}{\Delta \vdash (\lambda x : \sigma. t)s =_{\tau} t[s/x]}$
- (xi) (Conv)  $\frac{\Delta \vdash A \quad \Delta \vdash A =_{\text{Prop}} B}{\Delta \vdash B}$
- (xii) (Propositional extensionality)
$$\frac{}{\Delta \vdash \forall P, Q : \text{Prop}. (P \Leftrightarrow Q) \Rightarrow P =_{\text{Prop}} Q}$$
- (xiii) (Functional extensionality)
$$\frac{}{\Delta \vdash \forall f, g : \sigma \rightarrow \tau. (\forall x : \sigma. fx =_{\tau} gx) \Rightarrow f =_{\sigma \rightarrow \tau} g}$$
- (xiv) (Infinity)
$$\frac{}{\Delta \vdash \exists f : I \rightarrow I. (\forall x, x' : I. fx =_I fx' \Rightarrow x =_I x') \wedge \exists y : I. \forall x : I. \neg (fx =_I y)}$$
- (xv) (Hilbert  $\varepsilon_{\tau}$  function / global choice operator)
$$\frac{}{\Delta \vdash \forall P : \tau \rightarrow \text{Prop}. \forall x : \tau. Px \Rightarrow P(\varepsilon_{\tau} P)}$$

**Remark 4.4.** Interpret  $\varepsilon$  as a choice operator.  $\varepsilon_\tau P$  for  $P : \tau \rightarrow \text{Prop}$  is an arbitrary choice of a  $t : \tau$  such that  $Pt$  holds if such a  $t$  exists. One could also define the rule to be:

$$\frac{}{\Delta \vdash \forall P : \text{Prop}. (\exists x : \tau. Px) \Rightarrow P(\varepsilon_\tau P)}$$

**Lemma 4.5.** *We can derive*

$$\frac{\Delta \vdash Ps \quad \Delta \vdash s =_\tau t \quad \Gamma \vdash P : \tau \rightarrow \text{Prop}}{\Delta \vdash Pt}$$

**Remark 4.6.**

- (i) All types are non-empty. Consider  $\varepsilon_\tau(\lambda x : \tau. \perp)$  which has type  $\tau$  in any context.
- (ii) Given a type  $\tau$ , we can define the collection of subsets of  $\tau$  as  $(\text{Set } \tau) := (\tau \rightarrow \text{Prop})$ . We view  $P : \tau \rightarrow \text{Prop}$  as the set  $\{x : \tau \mid Px\}$ .
- (iii) Proof assistants that implement simple type theory/higher order logic: HOL4, HOL Light, Isabelle/HOL.
- (iv) These proof assistants give another way to define new types: Given a type  $\tau$  and  $P : \tau \rightarrow \text{Prop}$ , if  $P$  is not always false, then you can define the subtype of  $\tau$  where  $P$  holds.

**Example 4.7.** We can define  $\alpha \times \beta$  as the subtype of  $\tau = (\alpha \rightarrow \beta \rightarrow \text{Prop})$  where  $P : \tau \rightarrow \text{Prop}$  holds where  $P$  is

$$\lambda f : \tau. \exists x : \alpha. \exists y : \beta. fxy \wedge \forall x' : \alpha. \forall y' : \beta. fx'y' \Rightarrow x =_\alpha x' \wedge y =_\beta y'$$

**Remark 4.8.** If you remove  $\varepsilon$  from Definition 4.3, you get a constructive version of simple type theory.

**Exercise 4.9.** *Prove that  $\neg(\top = \perp)$  holds.*

**Proposition 4.10** (Diaconescu).  $\forall P : \text{Prop}. P \wedge \neg P$ .

*Proof.* We give an informal argument. Let  $P : \text{Prop}$ . Define

$$U := \lambda A : \text{Prop}. (A =_{\text{Prop}} \top) \wedge P : \text{Prop} \rightarrow \text{Prop}$$

and

$$V := \lambda A : \text{Prop}. (A =_{\text{Prop}} \perp) \wedge P : \text{Prop} \rightarrow \text{Prop}$$

**Claim 1:**  $P \Rightarrow \varepsilon U = \varepsilon V$

*Proof.* If  $P$  holds then  $UA$  and  $VA$  are true for any  $A : \text{Prop}$  so in particular  $UA \Leftrightarrow VA$ . Therefore by propositional extensionality  $UA =_{\text{Prop}} VA$  so then by functional extensionality  $U =_{\text{Prop} \rightarrow \text{Prop}} V$ . So  $\varepsilon U =_{\text{Prop}} \varepsilon V$ .

Notice that  $U\top$  is provable, so by the choice axiom  $U(\varepsilon U)$  is provable, so  $(\varepsilon U = \top) \wedge P$  holds. Similarly,  $V\perp$  holds, so  $(\varepsilon V = \perp) \wedge P$  does as well.

Do case distinctions on both of these disjunctions. In three cases  $P$  holds and we are done. In the last case we know  $\varepsilon U = \top$  and  $\varepsilon V = \perp$ . Thus by Exercise 4.9,  $\varepsilon U \neq \varepsilon V$ . If  $P$  holds, then  $\varepsilon U = \varepsilon V$  which is a contradiction. So we conclude  $\neg P$ .  $\square$

**Remark 4.11.**

- (i)  $\frac{\Delta, P \vdash \perp}{\Delta \vdash \neg P}$  is constructively valid.
- (ii)  $\frac{\Delta, \neg P \vdash \perp}{\Delta \vdash P}$  is constructively false.

**Example 4.12.** Define  $(\exists x : \tau.A) := \forall P : \text{Prop}. (\forall x : \tau.A \Rightarrow P) \Rightarrow P$ . We will prove

$$\frac{\Gamma \vdash t : \tau \quad \Delta \vdash A[t/x]}{\exists x : \tau.A}$$

as an example. First one would need to show a weakening rule, i.e. that for  $\Delta \subseteq \Delta'$  we know

$$\frac{\Delta \vdash P}{\Delta' \vdash P}$$

for any  $P : \text{Prop}$ . Set  $B := (\forall x : \tau.A \Rightarrow P)$ . Then we can prove that claim by the following proof tree

$$\begin{array}{c} \text{Ass } \frac{}{\Delta, B \vdash B} \quad \Gamma \vdash t : \tau \quad \text{Weak } \frac{\Delta \vdash A[t/x]}{\Delta, B \vdash A[t/x]} \\ \forall \text{ E } \frac{}{\Delta, B \vdash A[t/x] \Rightarrow P} \quad \Rightarrow \text{ E } \frac{}{\Delta, B \vdash P} \\ \Rightarrow \text{ I } \frac{}{\Delta \vdash (\forall x : \tau.A \Rightarrow P) \Rightarrow P} \\ \forall \text{ I } \frac{}{\Delta \vdash \exists x : \tau.A} \end{array}$$

**Example 4.13.** We can define  $\tau$  having a group structure by considering

$$\text{GroupStruct } \alpha := (\alpha \rightarrow \alpha \rightarrow \alpha) \times \alpha \times (\alpha \rightarrow \alpha)$$

and a predicate  $\text{GroupStruct } \alpha \rightarrow \text{Prop}$  expressing that the group axioms are satisfied where we use the first component to represent the group operation, the second for the identity and the third for the inverse.



**Remark 4.14.** Differentiating types and terms can have annoying consequences. While we can define `GroupStruct  $\mathbb{Z}$` , we cannot define the group  $\mathbb{Z}/n\mathbb{Z}$  as a type for a general  $n$ . This is because  $\mathbb{Z}/n\mathbb{Z}$  cannot be a type as it depends on a term. In the same way we can define  $\mathbb{R}^2$  and  $\mathbb{R}^3$  but cannot define  $\mathbb{R}^n$  as a type for a general  $n$ . Due to the distinction of types and terms, we also cannot quantify over types.