Logic of Proof Assistants

Prof. Floris van Doorn *
University of Bonn

10^{th} July, 2025

Contents

*LaTeX-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 π be the prime counting function, i.e. $\pi\colon \mathbb{R} \to \mathbb{N}$, $x \mapsto |\{p \leq x \mid p \text{ prime}\}|$. Then $\lim_{x\to\infty} \frac{\pi(x)}{x/\log(x)} = 1$.*

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 $S\colon \mathbb{N} \to \mathbb{N}$, $n \mapsto n+1$.

Definition 1.8. Let U be a set and $\mathcal{C} \subseteq \bigcup_{n\in\mathbb{N}}(U^n \to U)$ a set of **constructors**, where $c\colon U^n \to U$ is called an **n-ary function** and $(U^n \to U)$ is the collection of 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. $\varnothing$ 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 σ-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 $P\colon A \to \{\top, \bot\}$ 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 $x_1, \ldots, x_n \in A$ if $P(x_1), \ldots, P(x_n)$ then $P(c(x_1, \ldots, x_n))$.*

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 $g_c\colon B^n \to B$ then there is a unique function $f\colon A \to B$ such that $f(c(a_1, \ldots, a_n)) = g_c(f(a_1), \ldots, f(a_n))$ for every $c \in \mathcal{C}$ and $a_1, \ldots a_n \in A$.*

Proof. Exercise. □

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

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 $a\colon \mathcal{F} \cup \mathcal{R} \to \mathbb{N}$ is the arity function.

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

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

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** $S \coloneqq \mathcal{F} \dot\cup \mathcal{V} \dot\cup \{\text{“(”}, \text{“)”}, \text{“,”}\}$ 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 $c_x \coloneqq x$

(ii) for each n-ary function symbol f there is an n-ary constructor $c_f\colon (S^*)^n \to S^*$, $c_f(t_1, \dots, t_n) \coloneqq f\text{“(”}t_1\text{“,”}\dots\text{“,”}t_n\text{“)”}$

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 $f\text{“(”}x_0\text{“,”}x_1\text{“)”}$ 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

$$\varphi, \psi ::= \bot \mid s = t \mid R(t_1, \ldots, t_n) \mid (\varphi \wedge \psi) \mid (\varphi \vee \psi) \mid (\varphi \to \psi) \mid (\forall x.\varphi) \mid (\exists x.\varphi)$$

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

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) $\neg\varphi \coloneqq (\varphi \to \bot)$

(ii) $\varphi \leftrightarrow \psi \coloneqq ((\varphi \to \psi) \wedge (\psi \to \varphi))$

Remark 2.12. When writing formulas we omit some parentheses:

(i) $\varphi \to \psi \to \theta$ means $\varphi \to (\psi \to \theta)$

(ii) $\forall x.\varphi \to \psi$ means $\forall x.(\varphi \to \psi)$

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 **α-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 φ or a term t we also write $\mathrm{fv}(\varphi)$ and $\mathrm{fv}(t)$ for the set of free variables in φ 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) $y[t/x] \coloneqq \begin{cases} t & \text{if } y = x \\ y & \text{otherwise} \end{cases}$

(ii) $f(s_1, \ldots, s_n)[t/x] \coloneqq f(s_1[t/x], \ldots, s_n[t/x])$

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 φ **substitution** $\varphi[t/x]$ is defined as:

(i) $(s = s')[t/x] \coloneqq (s[t/x] = s'[t/x])$

(ii) $R(t_1, \ldots, t_n)[t/x] \coloneqq R(t_1[t/x], \ldots, t_n[t(x)])$

(iii) $(\varphi \vee \psi)[t/x] \coloneqq (\varphi[t/x] \vee \psi[t/x])$

(iv) $(\varphi \wedge \psi)[t/x] \coloneqq (\varphi[t/x] \wedge \psi[t/x])$

(v) $(\varphi \to \psi)[t/x] \coloneqq (\varphi[t/x] \to \psi[t/x])$

(vi) $(\forall y.\varphi)[t/x] \coloneqq \begin{cases} \forall y.\varphi & \text{if } y = x \\ \forall z.\varphi[z/y][t/x] & \text{otherwise} \end{cases}$ where z does not occur in t.

(vii) $(\exists y.\varphi)[t/x] \coloneqq \begin{cases} \exists y.\varphi & \text{if } y = x \\ \exists z.\varphi[z/y][t/x] & \text{otherwise} \end{cases}$ where z does not occur in t.

Definition 2.19. α**-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) $(\varphi_1 \to \varphi_2) \equiv_\alpha (\psi_1 \to \psi_2)$ 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 α-equivalence as an equivalence relation. You could also define the formulas as $\Phi_{\mathcal{L}} / \equiv_\alpha$ and thus treat α-equivalence as equality.

2.1 Provability

Definition 2.21. Let Γ be a set of formulas and φ a formula. Then $\boldsymbol{\Gamma \vdash \varphi}$ (read: "Γ proves φ") is defined inductively by

(i) $\dfrac{}{\Gamma, \varphi \vdash \varphi}$ (assumption rule)

(ii) $\dfrac{\Gamma \vdash \varphi \qquad \Gamma \vdash \psi}{\Gamma \vdash \varphi \wedge \psi}$ ($\wedge$-introduction)

(iii) $\dfrac{\Gamma \vdash \varphi_1 \wedge \varphi_2}{\Gamma \vdash \varphi_i}$ for $i = 1, 2$ ($\wedge$-elimination)

(iv) $\dfrac{\Gamma \vdash \varphi_i}{\Gamma \vdash \varphi_1 \vee \varphi_2}$ for $i = 1, 2$ ($\vee$-introduction)

(v) $\dfrac{\Gamma \vdash \varphi \vee \psi \qquad \Gamma, \varphi \vdash \theta \qquad \Gamma, \psi \vdash \theta}{\Gamma \vdash \theta}$ ($\vee$-elimination)

(vi) $\dfrac{\Gamma, \varphi \vdash \psi}{\Gamma \vdash \varphi \to \psi}$ ($\to$-introduction)

(vii) $\dfrac{\Gamma \vdash \varphi \to \psi \qquad \Gamma \vdash \varphi}{\Gamma \vdash \psi}$ ($\to$-elimination)

(viii) $\dfrac{\Gamma, \neg\varphi \vdash \bot}{\Gamma \vdash \varphi}$ (proof by contradiction)

(ix) $\dfrac{\Gamma \vdash \varphi}{\Gamma \vdash \forall x.\varphi}$ for $x \notin \mathrm{fv}(\Gamma)$ ($\forall$-introduction)

(x) $\dfrac{\Gamma \vdash \forall x.\varphi}{\Gamma \vdash \varphi[t/x]}$ ($\forall$-elimination)

(xi) $\dfrac{\Gamma \vdash \varphi[t/x]}{\Gamma \vdash \exists x.\varphi}$ ($\exists$-introduction)

(xii) $\dfrac{\Gamma \vdash \exists x.\varphi \qquad \Gamma, \varphi \vdash \psi}{\Gamma \vdash \psi}$ for $x \notin \mathrm{fv}(\Gamma, \psi)$ ($\exists$-elimination)

(xiii) $\dfrac{}{\Gamma \vdash t = t}$ (=-introduction)

(xiv) $\dfrac{\Gamma \vdash s = t \qquad \Gamma \vdash \varphi[t/x]}{\Gamma \vdash \varphi[s/x]}$ (=-elimination)

(xv) $\dfrac{\Gamma \vdash \varphi}{\Gamma \vdash \psi}$ for $\varphi \equiv_\alpha \psi$ (α-equivalence)

Remark 2.22. Read "$\dfrac{\text{A}}{\text{B}}$" as: "Under the assumptions A we can prove B". With "Γ, φ" we really mean $\Gamma \cup \{\varphi\}$.

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

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

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 $f_{\mathcal{M}} \colon |\mathcal{M}|^n \to |\mathcal{M}|$

(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 $\sigma \colon \mathcal{V} \to |\mathcal{M}|$ we define $[\![t]\!]_{\mathcal{M},\sigma}$ as:

(i) $[\![x]\!]_{\mathcal{M},\sigma} \coloneqq \sigma(x)$

(ii) $[\![f(t_1, \ldots, t_n)]\!]_{\mathcal{M},\sigma} \coloneqq f_{\mathcal{M}}([\![t_1]\!]_{\mathcal{M},\sigma}, \ldots, [\![t_n]\!]_{\mathcal{M},\sigma})$

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

(i) $\mathcal{M} \models_\sigma R(t_1, \ldots, t_n)$ iff $R_{\mathcal{M}}([\![t_1]\!]_{\mathcal{M},\sigma}, \ldots, [\![t_n]\!]_{\mathcal{M},\sigma})$

(ii) $\mathcal{M} \models_\sigma \bot$ never holds

(iii) $\mathcal{M} \models_\sigma s = t$ iff $[\![s]\!]_{\mathcal{M},\sigma} = [\![t]\!]_{\mathcal{M},\sigma}$

(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) $\mathcal{M} \models_\sigma \varphi \to \psi$ 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

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

(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 $\mathrm{fv}(\varphi) \subseteq \vec{x}$ and $\varphi(\vec{t})$ for $\varphi[\vec{t}/\vec{x}]$.

Remark 2.27. $[\![t]\!]_{\mathcal{M},\sigma}$ and $\mathcal{M} \models_\sigma \varphi$ only depend on the values $\sigma(x)$ where $x \in \mathrm{fv}(t)$ and $x \in \mathrm{fv}(\varphi)$ respectively. If φ is a sentence then $\mathcal{M} \models_\sigma \varphi$ does not depend on σ and is denoted $\mathcal{M} \models \varphi$ (read: "$\mathcal{M}$ realizes φ").

Definition 2.28. If Γ is a set of formulas and φ is a formula, then $\Gamma \models \varphi$ means that for any $\mathcal{L}$-structure $\mathcal{M}$ and assignment $\sigma : \mathcal{V} \to |\mathcal{M}|$ 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. $\exists! x.\varphi(x, \vec{z}) \coloneqq \exists x.(\varphi(x, \vec{z}) \wedge \forall y.\varphi(y, \vec{z}) \to y = x)$.

Definition 2.33. Suppose Γ is a set of $\mathcal{L}$-sentences, Γ' a set of $\mathcal{L}'$-sentences and $\mathcal{L} \subseteq \mathcal{L}'$. Then Γ' **is conservative over** Γ if $\Gamma \subseteq \Gamma'$ and for all $\mathcal{L}$-formulas ψ 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 Γ.*

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

(i) Extensionality : $\forall x \forall y.(\forall z.z \in x \leftrightarrow z \in y) \to x = y$

(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 $\{x\} \coloneqq \{x, x\}$.

(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 $x \cup y \coloneqq \bigcup \{x, y\}$.

(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 $\forall w. w \in z \to w \in x$

(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: $\exists x. \varnothing \in x \wedge \forall y. y \in x \to y \cup \{y\} \in x$ where $\varnothing \coloneqq \{w \in y \mid \bot\}$

(vii) Foundation: $\forall x.(\exists y. y \in x) \to \exists y. y \in x \wedge \forall z. z \in x \to z \notin y$ ("Every set x contains an element y disjoint from x")

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

(ix) Choice: $\forall x. \varnothing \notin x \to \exists f. f \in (x \to \bigcup x) \wedge \forall y. y \in x \to f(y) \in y$

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

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

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

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

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 λ-calculus

Definition 3.1. Let $\mathcal{V} = \{x_0, x_1, \dots\}$ a countably infinite set of variables and $\mathbf{C} = \{c_0, c_1, \dots\}$ be any set of constants. The terms of the **λ-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 α-equivalence ($\equiv_\alpha$) and substitution ($t[s/x]$). Here we identify α-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) **β-contraction ($\triangleright_\beta$)** is defined as $(\lambda x.t)s \triangleright_\beta t[s/x]$. $(\lambda x.t)s$ is called a **β-redex** and $t[s/x]$ is called its **β-reduct**.

(ii) **One-step-β-reduction ($\to_{\beta,1}$)** is defined as the compatible closure of $\triangleright_\beta$, i.e.

  (a) If $s \triangleright_\beta t$ then $s \to_{\beta,1} t$.

  (b) If $s \to_{\beta,1} t$ then $su \to_{\beta,1} tu$, $us \to_{\beta,1} ut$ and $\lambda x.s \to_{\beta,1} \lambda x.t$.

(iii) **β-reduction ($\to_\beta$)** is the reflexive transitive closure of $\to_{\beta,1}$, i.e. it is the smallest relation that is reflexive, transitive and contains $\to_{\beta,1}$.

(iv) **β-equivalence ($\equiv_\beta$)** is the smallest equivalence relation containing $\to_\beta$.

Example 3.6.

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

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

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

Definition 3.7. We define the following combinators:

(i) $\mathbf{I} = \lambda x.x$

(ii) $\mathbf{K} = \lambda xy.x$

(iii) $\mathbf{K}_* = \lambda xy.y$

(iv) $\mathbf{S} = \lambda xyz.xz(yz)$

Remark 3.8. Every term can be defined using K and S up to β-equivalence.

Example 3.9. $SKK \to_\beta \lambda z.Kz(Kz) \to_\beta \lambda z.z = I$.

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

Proof. Let $A \coloneqq \lambda fx.x(ffx)$ and let $Y \coloneqq AA$ be the **Turing operator**. Then $Yt = AAt \to_\beta t(AAt) = t(Yt)$. □

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

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

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

Definition 3.14. Let A be a set and $\to_1$ a binary relation on A with reflexive transitive closure $\to$.

(i) $t \in A$ is **in normal form** if there is no s such that $t \to_1 s$.

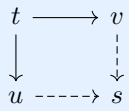
(ii) $t \in A$ **has normal form** s if s is in normal form and $t \to s$.

(iii) $t \in A$ is **strongly normalizing** if there exists no infinite sequence $t \to_1 t_1 \to_1 t_2 \to_1 t_3 \to_1 \cdots$.

(iv) $\to_1$ is **(weakly) normalizing** if every $t \in A$ has a normal form.

(v) $\to_1$ is **strongly normalizing (S.N.)** if every $t \in A$ is strongly normalizing.

(vi) $\to_1$ is **confluent** (has the **Church-Rosser property**) if whenever $u \leftarrow t \to v$ there is an $s \in A$ with $u \to s \leftarrow v$.

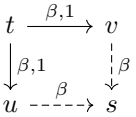


Remark 3.15. β-reduction is neither weakly nor strongly normalizing.

Proof. See the counterexamples presented in Example 3.6 (iii) and Proposition 3.10. □

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

Remark 3.17. It does not suffice to prove



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. □

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

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

Lemma 3.21. *If $t \Rightarrow t'$ then $t \to_\beta t'$.*

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 $t \to_\beta t'$ and $u \to_\beta u'$. Then $(\lambda x.t)u \to_\beta (\lambda x.t')u \to_\beta (\lambda x.t')u' \to_{\beta,1} t'[u'/x]$. The other cases are similar. □

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. □

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 λ-term.

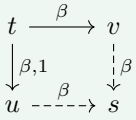
(iv) $((\lambda x.t)s)^* = t^*[s^*/x]$.

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

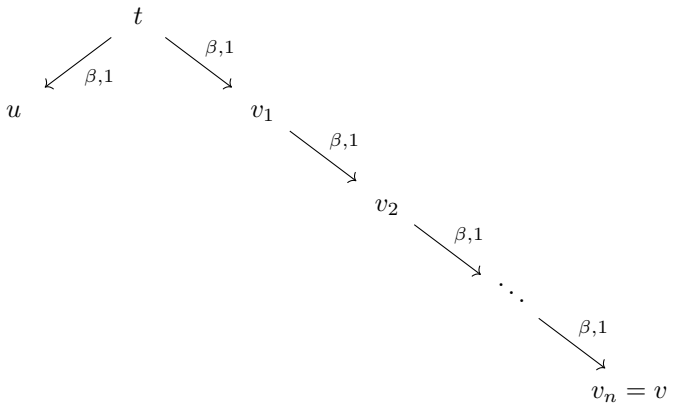
Proof. We show this by induction of the length of the derivation for $s \Rightarrow t$.
If the last step was Definition 3.18 (iii), i.e. concluding $tu \Rightarrow t'u'$ from $t \Rightarrow t'$ and $u \Rightarrow u'$, then by induction hypothesis $t' \Rightarrow t^*$ and $u' \Rightarrow u^*$. We need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

If t is not a λ 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. □

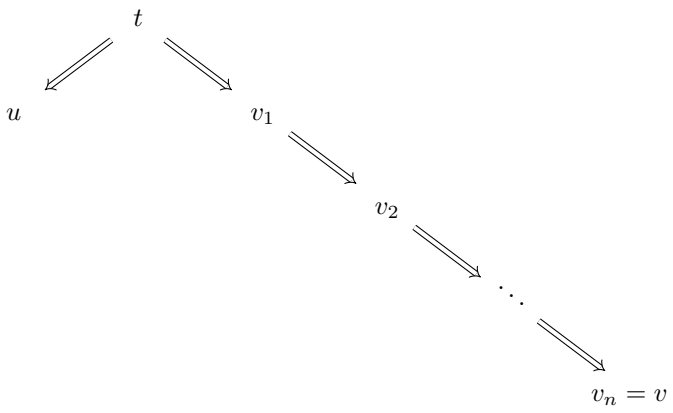
Lemma 3.25. *If $t \to_{\beta,1} u$ and $t \to_\beta v$ then there is an s with $u \to_\beta s \,{}_\beta\!\leftarrow v$.*



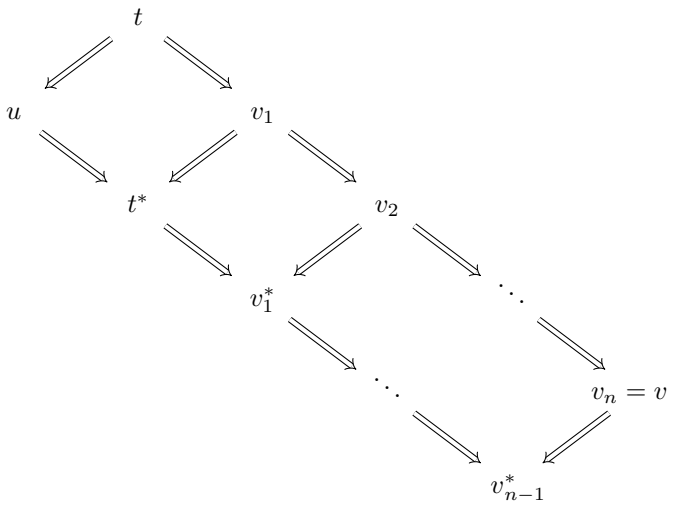
Proof. Decomposing $t \to_\beta v$ into individual steps gives us



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 β-reduction and β-reduction is transitive. □

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

Corollary 3.27. *Every term t has at most one β-normal form.*

Corollary 3.28. *If $s \equiv_\beta t$, then there is u such that $s \to_\beta u \,{}_\beta\!\leftarrow t$.*

3.1 Partial recursive functions

Definition 3.29. Let $\mathbf{A \rightharpoonup B}$ 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 $f\colon X \to B$. We set $\mathrm{dom}(\mathbf{f}) \coloneqq X$ and call it the **domain** of f.

Notation 3.30. We set $\mathbf{f(\bar{x})\downarrow} \coloneqq (\bar{x} \in \mathrm{dom}(f))$ and $\mathbf{f(\bar{x})\uparrow} \coloneqq (\bar{x} \notin \mathrm{dom}(f))$.

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

(i) $0\colon \mathbb{N}^0 \to \mathbb{N}$ is P.R.

(ii) $\mathrm{Succ}\colon \mathbb{N} \to \mathbb{N}, x \mapsto x+1$ is P.R.

(iii) The projection $\mathrm{P}_i^k\colon \mathbb{N}^k \to \mathbb{N}, \mathrm{P}_i^k(x_0, \ldots, x_{k-1}) = x_i$ is P.R. for $0 \leq i < k$.

(iv) Composition: if $f\colon \mathbb{N}^k \to \mathbb{N}$ is P.R. and $g_0, \ldots, g_{k-1}\colon \mathbb{N}^l \to \mathbb{N}$ are P.R. then $f \circ \vec{g}\colon \mathbb{N}^l \to \mathbb{N}, \vec{x} \mapsto f(g_0(\vec{x}), \ldots, g_{k-1}(\vec{x}))$ is P.R. and $\vec{x} \in \mathrm{dom}(f \circ \vec{g})$ iff $\vec{x} \in \mathrm{dom}(g_i)$ for all i and $(g_0(\vec{x}), \ldots, g_{k-1}(\vec{x})) \in \mathrm{dom}(f)$.

(v) Primitive recursion: if $f_0\colon \mathbb{N}^k \to \mathbb{N}$ and $f_s\colon \mathbb{N}^{k+2} \to \mathbb{N}$ are P.R. then the function g defined by $g(0, \vec{x}) \coloneqq f_0(\vec{x})$ and $g(n+1, \vec{x}) \coloneqq f_s(n, g(n, \vec{x}), \vec{x})$ 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 $f\colon \mathbb{N}^{k+1} \to \mathbb{N}$ is P.R. then $\mu_f\colon \mathbb{N}^k \to \mathbb{N}$ 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 λ-term t **respresents** $f\colon \mathbb{N}^k \to \mathbb{N}$ if for all $\vec{n} \in \mathbb{N}^k$ we have $t[n_0]\dots[n_{k-1}] \to_\beta [f(\vec{n})]$ 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 $f\colon \mathbb{N}^k \to \mathbb{N}$ is P.R. then there are primitive recursive functions $u\colon \mathbb{N} \to \mathbb{N}$ and $t\colon \mathbb{N}^{k+1} \to \mathbb{N}$ such that $f = u \circ \mu_t$.*

Theorem 3.36. *$f\colon \mathbb{N}^k \to \mathbb{N}$ is P.R. iff it is represented by some λ-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, λ-terms, substitution in λ-terms and β-reduction. Then we can show that there is a partial recursive function that searches for a reduction sequence $t \to_{\beta,1} t_1 \to_{\beta,1} t_2 \to_{\beta,1} \dots \to_{\beta,1} [m]$.

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 $0\colon \mathbb{N}^0 \to \mathbb{N}$. 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

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

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

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

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

$$T \coloneqq \lambda u.(\mathrm{Succ}(uK), F_s(uK)(uK_*))$$

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

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

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

Then

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

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

$$[n+1]T([0], F_0) \to_\beta T([n]T([0], F_0)) \overset{IH}{\equiv_\beta} T([n], G[n]) \to_\beta ([n+1], F_s[n](G[n]))$$

and

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

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 $Dst[0] \to_\beta s$ and $Dst[n+1] \to_\beta t$. (The existence of D is an exercise.) Let T be a term and $\vec{x}$ be a sequence of variables. We define

$$W \coloneqq Y(\lambda vy.Dy(v(\mathrm{Succ}(y)))(Ty\vec{x}))$$

Then we get

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

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

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

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

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

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

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

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. □

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

3.2 Simply typed λ-calculus

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

Definition 3.38. $\lambda_\to^{\text{Curry}}$ 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

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

where α is a type variable and C a type constant.

(ii) As **preterms** or **raw terms** we have precisely the terms of the λ-calculus.

(iii) A **context** Γ is a finite set $\{x_1 : \tau_1, \dots, x_n : \tau_n\}$ where the x_i are distinct term-variables and the τ_i are types. Γ 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 τ in the context Γ.") generated by:

(a) VAR: $\dfrac{}{\Gamma, x : \tau \vdash x : \tau}$ where $x \notin \text{dom}(\Gamma)$.

(b) APP: $\dfrac{\Gamma \vdash t : \sigma \to \tau \qquad \Gamma \vdash s : \sigma}{\Gamma \vdash ts : \tau}$

(c) ABS: $\dfrac{\Gamma, x : \sigma \vdash t : \tau}{\Gamma \vdash \lambda x.t : \sigma \to \tau}$ 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 $\varnothing \vdash t : \tau$.

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

Example 3.40. For any types σ an τ and K as in Definition 3.7 we get

$$\dfrac{\dfrac{\dfrac{}{x : \sigma, y : \tau \vdash x : \sigma}\,\text{VAR}}{x : \sigma \vdash \lambda y.x : \tau \to \sigma}\,\text{ABS}}{\vdash K : \sigma \to \tau \to \sigma}\,\text{ABS}$$

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

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

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

(i) one constant type $\mathbb{N}$ (or Nat)

(ii) constant terms 0, Succ and R_τ for any type τ

(iii) three additional typing rules:

(a) $\dfrac{}{\vdash 0 : \mathbb{N}}$

(b) $\dfrac{}{\vdash \text{Succ} : \mathbb{N} \to \mathbb{N}}$

(c) $\dfrac{}{\vdash R_\tau : \tau \to (\mathbb{N} \to \tau \to \tau) \to \mathbb{N} \to \tau}$

Additionally, we define:

(i) **ι-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-ι-reduction ($\to_{\iota,1}$)** as the compatible closure of ι-contraction

(iii) **one-step-β-ι-reduction ($\to_{\beta\iota,1}$)** as the compatible closure of ι- and β-contraction

(iv) **ι-reduction ($\to_\iota$)** as reflexive transitive closure of one-step-ι-reduction

(v) **β-ι-reduction ($\to_{\beta\iota}$)** as the reflexive transitive closure of one-step-β-ι-reduction

Lemma 3.43.

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

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

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

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

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

For (i), we know by induction hypothesis that $\mathrm{fv}(t') \subseteq \mathrm{dom}(\Gamma, x : \sigma) = \mathrm{dom}(\Gamma) \cup \{x\}$. Then $\mathrm{fv}(\lambda x.t') = \mathrm{fv}(t') \setminus \{x\} \subseteq \mathrm{dom}(\Gamma)$. For (ii), let Γ' be another context satisfying $(*)$. Then for all $y \in \mathrm{fv}(t')$, 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 $\Gamma' \vdash \lambda x.t' : \sigma \to \tau$. □

Lemma 3.44 (Generation)**.**

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

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

(iii) If $\Gamma \vdash \lambda x.t : \tau$ *then for some* τ_1 *and* τ_2*, we have* $\tau = \tau_1 \to \tau_2$ *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. □

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

(i) $\alpha[\sigma/\alpha] \coloneqq \sigma$

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

(iii) $(\tau_1 \to \tau_2)[\sigma/\alpha] \coloneqq \tau_1[\sigma/\alpha] \to \tau_2[\sigma/\alpha]$

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* $t \to_\beta t'$ *then* $\Gamma \vdash t' : \tau$.

Proof. Proving (i) and (ii) is an exercise. We show (iii) by induction on $t \to_\beta t'$. 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 $\Gamma \vdash \lambda x.t_1 : \sigma \to \tau$ and $\Gamma \vdash t_2 : \sigma$. So $\Gamma x : \sigma \vdash t_1 : \tau$, and thus by term substitution $\Gamma \vdash t_1[t_2/x] : \tau$. □

Remark 3.47. If $t \to_\beta t'$ 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 Γ' and τ' with $\Gamma' \vdash t : \tau'$ there is a type substitution $S = [\sigma_1/\alpha_1, \ldots, \sigma_k/\alpha_k]$ 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 λ-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 $\alpha_x = (\alpha_z \to \beta)$, from yz we $\alpha_y = (\alpha_z \to \gamma)$ and from $(xz)(yz)$ we know $\beta = (\gamma \to \varrho)$. Then we get

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

which using abstraction gives us

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

One can show that this is a principal judgement.

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

(i) typeability: given t*, construct* Γ *and* τ *such that* $\Gamma \vdash t : \tau$.

(ii) type-checking: given t*,* Γ *and* τ *does* $\Gamma \vdash t : \tau$ *hold?*

Definition 3.52. $\lambda_\to^{\text{Church}}$ has the same types as $\lambda_\to^{\text{Curry}}$. 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 τ a type. Additionally we keep the **typing judgements** VAR and APP but change ABS to be

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

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

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

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

Proposition 3.55 (Unique typing)**.** *In $\lambda_\to^{\mathrm{Church}}$, 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.

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

with $\tau = (\sigma' \to \tau')$ 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. □

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

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

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

Lemma 3.59. *A term t (in $\lambda_\to^{\mathrm{Church}}$) is strongly normalizing iff there is a natural reduction number n such that every reduction sequence $t \to_{\beta,1} t_1 \to_{\beta,1} t_2 \to_{\beta,1} \dots$ 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 $\downarrow t := \{s \mid t \to_\beta s\}$ and let $A := \{s \in \downarrow t \mid \downarrow s \text{ is finite}\}$.

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

Proof. The claim follows from the following observations:

(i) $\#\{s' \mid s \to_{\beta,1} s'\}$ is finite because s has finitely many β-redexes.

(ii) $\downarrow s = \{s\} \cup \bigcup_{s \to_{\beta,1} s'} \downarrow s'$

By assumption $\downarrow s'$ is finite for all $s \to_{\beta,1} s'$. 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 $t \to_{\beta,1} t_1$ with $t_1 \notin A$ and $t_1 \to_{\beta,1} t_2$ with $t_2 \notin A$. Repeating this we get

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

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

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

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

(i) if τ is a variable or a constant, then $t \in C(\tau)$ iff $t : \tau$ and t is strongly normalizing.

(ii) $t \in C(\sigma \to \tau)$ iff $t : \sigma \to \tau$ 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 $t \to_\beta t'$, then $t' \in C(\tau)$.

(iii) Suppose $t : \tau$ and t is not a λ-abstraction. If

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

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

Proof. We do a simultaneous proof of (i) - (iii) by induction on τ. If τ is a variable or constant, then (i) - (iii) are easy. Suppose $t : \sigma \to \tau$.
For (i) assume that $t \in C(\sigma \to \tau)$. 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 $t \to_{\beta,1} t_1 \to_{\beta,1} t_2 \to_{\beta,1} \dots$ then $tx \to_{\beta,1} t_1x \to_{\beta,1} t_2x \to_{\beta,1} \dots$.
For (ii) assume $t \in C(\sigma \to \tau)$ and $t \to_{\beta,1} t'$. To show that $t' \in C(\sigma \to \tau)$, 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 $ts \to_\beta t's$, we conclude that $t's \in C(\tau)$.
For part (iii), let $t : \sigma \to \tau$ satisfy $(*)$ and not be a λ-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, $ts \to_{\beta,1} t's$ where $t \to_{\beta,1} t'$ or secondly, $ts \to_{\beta,1} ts'$ where $s \to_{\beta,1} s'$. For the first option we know $t' \in C(\sigma \to \tau)$ because t satisfies $(*)$. By definition, $t's \in C(\tau)$. For the second option, if $s \to_{\beta,1} s'$, then by the induction hypothesis of the nested induction and $h(s') < h(s)$, we know $ts' \in C(\tau)$. This finishes (iii). □

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

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

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

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 $(\lambda x : \sigma.t)s \to_{\beta,1} r$. We need to show $r \in C(\tau)$. There are three cases:

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

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

(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 $t[s/x] \to_\beta t'[s/x]$ we have $t'[s/x] \in C(\tau)$.

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

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

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) $\tau = (\sigma' \to \tau')$, $x' : \tau'$ and $t' : \tau'$ for some σ' and τ'. We want to show $(\lambda x'.t')[\vec{s}/\vec{x}] = \lambda x'.(t'[\vec{s}/\vec{x}]) \in C(\sigma' \to \tau')$. 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 σ such that $t' : \sigma \to \tau$ 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 $t'[\vec{s}/\vec{x}] \in C(\sigma \to \tau)$ and $s'[\vec{s}/\vec{x}] \in C(\sigma)$. By definition, $t[\vec{s}/\vec{x}] \in C(\tau)$. □

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 $k = 0$. □

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 $\lambda_{\to}^{\text{Church}}$ the type constants

(i) I ("**individuals**")

(ii) Prop

and the term constants

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

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

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

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

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) $\bot := \forall P : \text{Prop}.P$

(ii) $\top := \bot \to \bot$

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

(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 $\Delta \vdash_\Gamma A$** (or $\Delta \vdash A$ for short) where Δ 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) $\dfrac{}{\Delta, A \vdash A}$

(ii) ($\Rightarrow$ I) $\dfrac{\Delta, A \vdash B}{\Delta \vdash A \Rightarrow B}$

(iii) ($\Rightarrow$ E) $\dfrac{\Delta \vdash A \Rightarrow B \qquad \Delta \vdash A}{\Delta \vdash B}$

(iv) ($\forall$ I) $\dfrac{\Delta \vdash A \qquad x \notin \text{fv}(\Delta) \qquad \Gamma \vdash x : \tau}{\Delta \vdash \forall x : \tau . A}$

(v) ($\forall$ E) $\dfrac{\Delta \vdash \forall x : \tau . A \qquad \Gamma \vdash t : \tau}{\Delta \vdash A[t/x]}$

(vi) (= Refl) $\dfrac{\Gamma \vdash t : \tau}{\Delta \vdash t =_\tau t}$

(vii) (= Symm) $\dfrac{\Delta \vdash t =_\tau s}{\Delta \vdash s =_\tau t}$

(viii) (= Trans) $\dfrac{\Delta \vdash t =_\tau s \qquad \Delta \vdash s =_\tau r}{\Delta \vdash t =_\tau r}$

(ix) (App Congr) $\dfrac{\Delta \vdash t =_{\sigma \to \tau} t' \qquad \Delta \vdash s =_\sigma s'}{\Delta \vdash ts =_\tau t's'}$

(x) (β Reduce) $\dfrac{\Gamma, x : \sigma \vdash t : \tau \qquad \Gamma \vdash s : \sigma}{\Delta \vdash (\lambda x : \sigma . t)s =_\tau t[s/x]}$

(xi) (Conv) $\dfrac{\Delta \vdash A \qquad \Delta \vdash A =_{\text{Prop}} B}{\Delta \vdash B}$

(xii) (Propositional extensionality)

$$\frac{}{\Delta \vdash \forall P, Q : \text{Prop}. (P \Leftrightarrow Q) \Rightarrow P =_{\text{Prop}} Q}$$

(xiii) (Functional extensionality)

$$\frac{}{\Delta \vdash \forall f, g : \sigma \to \tau. (\forall x : \sigma . fx =_\tau gx) \Rightarrow f =_{\sigma \to \tau} g}$$

(xiv) (Infinity)

$$\frac{}{\Delta \vdash \exists f : I \to I. (\forall x, x' : I . fx =_I fx' \Rightarrow x =_I x') \wedge \exists y : I . \forall x : I . \neg (fx =_I y)}$$

(xv) (Hilbert ε_τ function / global choice operator)

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

Remark 4.4. Interpret ε as a choice operator. $\varepsilon_\tau P$ for $P : \tau \to \text{Prop}$ 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*

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

Remark 4.6.

(i) All types are non-empty. Consider $\varepsilon_\tau(\lambda x : \tau.\bot)$ which has type τ in any context.

(ii) Given a type τ, we can define the collection of subsets of τ as $(\text{Set}\,\tau) \coloneqq (\tau \to \text{Prop})$. We view $P : \tau \to \text{Prop}$ 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 τ and $P : \tau \to \text{Prop}$, if P is not always false, then you can define the subtype of τ where P holds.

Example 4.7. We can define $\alpha \times \beta$ as the subtype of $\tau = (\alpha \to \beta \to \text{Prop})$ where $P : \tau \to \text{Prop}$ 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 ε from Definition 4.3, you get a constructive version of simple type theory.

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

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

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

$$U \coloneqq \lambda A : \text{Prop}.(A =_{\text{Prop}} \top) \vee P : \text{Prop} \to \text{Prop}$$

and

$$V \coloneqq \lambda A : \text{Prop}.(A =_{\text{Prop}} \bot) \vee P : \text{Prop} \to \text{Prop}$$

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

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

Notice that $U\top$ is provable, so by the choice axiom $U(\varepsilon U)$ is provable, so $(\varepsilon U = \top) \vee P$ holds. Similarly, $V\bot$ holds, so $(\varepsilon V = \bot) \vee P$ 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 $\varepsilon V = \bot$. 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 $\neg P$. □

Remark 4.11.

(i) $\dfrac{\Delta, P \vdash \bot}{\Delta \vdash \neg P}$ is constructively valid.

(ii) $\dfrac{\Delta, \neg P \vdash \bot}{\Delta \vdash P}$ is not always constructively valid.

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

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

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 : Prop. Set $B :\equiv (\forall x : \tau.A \Rightarrow P)$. Then we can prove that claim by the following proof tree

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Delta, B \vdash B}\,\mathrm{Ass} \qquad \Gamma \vdash t : \tau}{\Delta, B \vdash A[t/x] \Rightarrow P}\,\forall\,\mathrm{E} \qquad \mathrm{Weak}\,\dfrac{\Delta \vdash A[t/x]}{\Delta, B \vdash A[t/x]}}{\Delta, B \vdash P}\,\Rightarrow\mathrm{E}}{\Delta \vdash (\forall x : \tau.A \Rightarrow P) \Rightarrow P}\,\Rightarrow\mathrm{I}}{\Delta \vdash \exists x : \tau.A}\,\forall\,\mathrm{I}$$

Example 4.13. We can define τ having a group structure by considering

$$\mathrm{GroupStruct}\,\alpha :\equiv (\alpha \to \alpha \to \alpha) \times \alpha \times (\alpha \to \alpha)$$

and a predicate $\mathrm{GroupStruct}\,\alpha \to \mathrm{Prop}$ 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.

5 Dependent Type Theory

The rules of simple type theory (when ignoring the terms) are precisely the rules of implicational logic. So, instead of having a type specifically for propositions, we could view types as propositions and terms as proofs. This is called **Curry-Howard correspondence** or the **propositions-as-types interpretation**. We could add other type formers for connectives, e.g. $\sigma \times \tau$ corresponding to conjunction and $\sigma + \tau$ corresponding to disjunction.

How do we represent quantifiers? We will add the the **dependent product type**, or **pi type**, $\prod x : \sigma.\tau$ or $\prod_{x:\sigma} \tau$, corresponding to $\forall x : \sigma.\tau$ and where τ can depend on x. A term $t : \prod x : \sigma.\tau$ is a dependent function. For $s : \sigma$, we get $ts : \tau[s/x]$.

Example 5.1. We could then define $\tau \coloneqq \mathbb{R}^n$ where $n : \mathbb{N}$. Then f, defined to send $n : \mathbb{N}$ to $\underbrace{(0, \dots, 0)}_{\text{length } n}$ would have type $\prod_{n:\mathbb{N}} \mathbb{R}^n$.

5.1 Pure Type Systems (PTSs)

Definition 5.2. A **pure type system (PTS)** is determined by $(\mathcal{S}, \mathcal{A}, \mathcal{R})$, where

(i) $\mathcal{S}$ is a set of **sorts** (or **universes**)

(ii) $\mathcal{A} \subseteq \mathcal{S} \times \mathcal{S}$ is a set of **axioms**

(iii) $\mathcal{R} \subseteq \mathcal{S} \times \mathcal{S} \times \mathcal{S}$ is a set of **relations**

We additionally have an infinite set of variables. We can define a lot of things we have seen before again:

(iv) A PTS has **preterms**

$$A, B, M, N ::= x \mid s \mid (MN) \mid (\lambda x : A.M) \mid \prod x : A, B$$

where x is a variable and s a sort. If B does not depend on x, we write $\prod x : A.B$ as $\mathbf{A \to B}$.

(v) **Contexts** are lists of the form $\Gamma \coloneqq x_1 : A_1, x_2 : A_2, \dots, x_n : A_n$. Here, the order of the context matters. We set $\mathrm{dom}(\mathbf{\Gamma}) \coloneqq \{x_1, \dots, x_n\}$.

(vi) We identify terms up to **α-equivalence**, so e.g. $\lambda x : \tau.x \equiv_\alpha \lambda y : \tau.y$ and $\prod x : \tau.B(x) \equiv_\alpha \prod y : \tau.B(y)$.

(vii) We define **β-reduction** as before.

(viii) The **types** are precisely the terms that have a sort as its type.

We write s for sorts, A, B for types and M, N for terms (that could be types). The typing rules are:

(ix) (ax) $\dfrac{}{\vdash s_1 : s_2}$ for $(s_1, s_2) \in \mathcal{A}$

(x) (var) $\dfrac{\Gamma \vdash A : s}{\Gamma, x : A \vdash x : A}$ where $x \notin \mathrm{dom}(\Gamma)$

(xi) (weak) $\dfrac{\Gamma \vdash M : A \qquad \Gamma \vdash B : s}{\Gamma, x : B \vdash M : A}$ where $x \notin \mathrm{dom}(\Gamma)$

(xii) (prod) $\dfrac{\Gamma \vdash A : s_1 \qquad \Gamma, x : A \vdash B : s_2}{\Gamma \vdash \prod x : A.B : s_3}$ for $(s_1, s_2, s_3) \in \mathcal{R}$

(xiii) (abs) $\dfrac{\Gamma, x : A \vdash M : B \qquad \Gamma \vdash \prod x : A.B : s}{\Gamma \vdash \lambda x : A.M : \prod x : A.B}$

(xiv) (app) $\dfrac{\Gamma \vdash M : \prod x : A.B \qquad \Gamma \vdash N : A}{\Gamma \vdash MN : B[N/x]}$

(xv) (conv) $\dfrac{\Gamma \vdash M : A \qquad \Gamma \vdash A' : s}{\Gamma \vdash M : A'}$ for $A \equiv_\beta A'$

Example 5.3. If $\tau : s$, then $(\lambda x : s.x)\tau \equiv_\beta \tau$. It depends on the PTS whether $(\lambda x : s.x)\tau$ is a well-formed type.

Remark 5.4. A rule (s_1, s_2, s_2) is abbreviated as (s_1, s_2) and axioms (s_1, s_2) are denoted $s_1 : s_2$. $\forall x : A.B$ is another way to write $\prod x : A.B$.

Definition 5.5. A pure type system is called **strongly** resp. **weakly normalizing** if β-reduction on well-typed terms (including sorts) is strongly resp. weakly normalizing.

Example 5.6.

(i) Let $\mathcal{S} = \{*, \Box\}$, $\mathcal{A} = \{* : \Box\}$ and $\mathcal{R} = \{(*, *)\}$. The only term of type $\Box$ is $*$. In this system, types cannot depend on terms. This system corresponds to simply-typed λ-calculus. An example of a derivation in this system is:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{* : \square}\,\text{ax}}{\sigma : * \vdash \sigma : *}\,\text{var}}{\sigma : *, \tau : * \vdash \sigma : *}\,\text{weak} \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{\vdash * : \square}\,\text{ax}}{\sigma : * \vdash * : \square}\,\text{weak}}{\sigma : *, \tau : * \vdash \tau : *}\,\text{var}}{\sigma : *, \tau : *, x : \sigma \vdash \tau : *}\,\text{weak}}{\sigma : *, \tau : * \vdash \sigma \to \tau : *}\,\text{prod}$$

We can also proof things like

$$\sigma : *, \tau : * \vdash (\sigma \to \sigma) \to \tau \to \sigma : *$$

and

$$\tau : *, \sigma : *, f : \sigma \to \tau, g : \tau \to \tau, x : \tau \vdash g(g(f(x))) : \tau.$$

(ii) Consider $\mathcal{S} = \{*, \square\}$, $\mathcal{A} = \{* : \square\}$ and $\mathcal{R} = \{(*, *), (\square, *)\}$. This is called **system F**. We can now derive $\vdash \prod \alpha : *.\alpha \to \alpha : *$. The last step of that derivation is

$$\dfrac{\dfrac{\vdots}{\vdash * : \square} \qquad \dfrac{\vdots}{\alpha : * \vdash \alpha \to \alpha : *}}{\vdash \prod \alpha : *, \alpha \to \alpha : *}\,\text{prod}$$

This is a **polymorphic type**. An inhabitant of this type is $\lambda\alpha : *.\lambda x : \alpha.x$. This is called the **polymorphic identity function**. The word "**polymorphic**" means "for all types at the same time".

We can give impredicative encodings of connectives. For example, $\bot \coloneqq \prod_{\alpha:*} \alpha$ and $A \wedge B \coloneqq \prod_{\alpha:*}(A \to B \to \alpha) \to \alpha$.

(iii) Let $\mathcal{S} = \{*, \square\}$, $\mathcal{A} = \{* : \square\}$ and $\mathcal{R} = \{(*, *), (*, \square)\}$. This is called **$\lambda$P**. In this system $\alpha : * \vdash \alpha \to * : \square$. This system is where real dependent types start to show up. See this partial derivation:

$$\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{\alpha : * \vdash \alpha : *} \qquad \dfrac{\vdots}{\alpha : * \vdash * : \square}}{\alpha : * \vdash \alpha \to * : \square}\,\text{prod}}{\alpha : *, \beta : \alpha \to * \vdash \beta : \alpha \to *}\,\text{var}}{\alpha : *, \beta : \alpha \to * \vdash \prod x : \alpha.\beta x : *}$$

We can also show

$$\alpha : *, \beta : \alpha \to *, \gamma : \alpha \to *, f : \prod_{x:\alpha} \beta x, g : \prod_{x:\alpha} \beta x \to \gamma x \vdash \lambda x : \alpha.gx(fx) : \prod_{x:\alpha} \gamma x.$$

(iv) Consider $\mathcal{S} = \{*, \Box\}$, $\mathcal{A} = \{* : \Box\}$ and $\mathcal{R} = \{(*,*), (\Box, \Box)\}$. This is called $\lambda\underline{\omega}$. In this system we can derive $\vdash * \to * : \Box$ and $\vdash \lambda\alpha : *.\alpha \to \alpha : *$. If we add $(\Box, *)$ to $\mathcal{R}$, we can derive

$$\vdash \lambda A : *.\lambda B : *.A \wedge B : * \to * \to *$$

(v) Let $\mathcal{S} = \{*, \Box\}$ and $\mathcal{A} = \{* : \Box\}$. **$\lambda$C** or the **calculus of constructions** has all 4 rules from the previous examples, i.e. $\mathcal{R} = \{(*,*), (*,\Box), (\Box,*), (\Box,\Box)\}$.

(vi) Lean's type theory has the following rules for sorts and function types:

$$\mathcal{S} = \{\text{Prop}\} \cup \{\text{Type } u \mid u \in \mathbb{N}\}$$
$$\mathcal{A} = \{\text{Prop} : \text{Type } 0\} \cup \{\text{Type } u : \text{Type } (u+1) \mid u \in \mathbb{N}\}$$
$$\mathcal{R} = \{(\text{Type } u, \text{Type } v, \text{Type } (\max u\ v)) \mid u, v \in \mathbb{N}\} \cup \{(s, \text{Prop}, \text{Prop}) \mid s \in \mathcal{S}\}$$

If $A : \text{Type } u$ and $B : \text{Type } v$, then $A \to B : \text{Type } (\max u\ v)$. If $P : A \to \text{Prop}$, then $\forall x : A.Px : \text{Prop}$. If $A : \text{Type } u$ and $C : A \to \text{Type } v$, then $\prod_{x:A} Cx : \text{Type } (\max u\ v)$. Prop is called and **impredicative universe**.

(vii) Consider $\mathcal{S} = \{*\}$, $\mathcal{A} = \{* : *\}$, $\mathcal{R} = \{(*,*)\}$. We can construct a term of type $\prod_{\alpha:*} \alpha$. This system is **inconsistent**, i.e. every type is inhabited. In this system, weak normalization fails.

Theorem 5.7. *In Example 5.6, the PTSs (i)-(vi) are strongly normalizing.*

Conjecture 5.8. *If a PTS is weakly normalizing, then it is strongly normalizing.*

Proposition 5.9. *In any PTS*

(i) If $\Gamma \vdash M : A$, then $\Gamma \vdash A : s$ or $A \to_\beta s$.

(ii) (Substitution) If $\Gamma, x : A, \Delta \vdash M : B$ and $\Gamma \vdash N : A$, then $\Gamma, \Delta[N/x] \vdash M[N/x] : B[N/x]$.

(iii) (Subject reduction) If $\Gamma \vdash M : A$ and $M \to_\beta M'$, then $\Gamma \vdash M' : A$.

Remark 5.10. $\vdash * : \Box$ shows that we need to consider $A \to_\beta s$ in Proposition 5.9 (i).

Definition 5.11. A PTS is **functional** if

(i) If $(s_1, s_2), (s_1, s_2') \in \mathcal{A}$, then $s_2 = s_2'$.

(ii) If $(r_1, r_2, r_3), (r_1, r_2, r_3') \in \mathcal{R}$, then $r_3 = r_3'$.

Theorem 5.12 (Unique typing)**.** *In a functional PTS, if* $\Gamma \vdash M : A$ *and* $\Gamma \vdash M : A'$ *then* $A \equiv_\beta A'$.

Remark 5.13. In Lean, $\mathbb{N}, \mathbb{R}, \mathbb{R}^{\mathbb{R}} : \text{Type } 0$. The category of groups in Type u has type Type $(u+1)$.

5.2 Inductive types

For concreteness we will stick to Lean's type theory. Instead of providing the very general and complicated definition of an inductive type, we will be explaining them mostly by example.

Remark 5.14. To make talking about Prop and Type easier, we define Sort as Sort $0 :=$ Prop and Sort $(u+1) :=$ Type u.

Definition 5.15. New type formers can be specified using constants and several rules:

(i) **formation rule**: specifying when a type is well-formed

(ii) **introduction rules**: specifying how to form an element of the type

(iii) **elimination rules**: specifying how to combine the type with others

(iv) **computation rules**: specifying how combinations of other rules simplify

(v) **uniqueness principle**: specifying which elements of the type agree

Example 5.16. The dependent product type can be viewed as an inductive type: (prod) is the formation rule, (abs) the introduction rule, (app) an elimination rule, β-reduction is a computation rule, and η-reduction $\lambda x : A.fx \to_\eta f$ is a uniqueness principle.

Definition 5.17. **Cartesian products** can now be defined inductively:

(i) formation rule: $$\frac{\Gamma \vdash A : \text{Type } u \qquad \Gamma \vdash B : \text{Type } v}{\Gamma \vdash A \times B : \text{Type } (\max u\ v)}$$

(ii) introduction rule: $$\frac{\Gamma \vdash a : A \qquad \Gamma \vdash b : B}{\Gamma \vdash (a, b) : A \times B}$$

(iii) elimination rules: $\dfrac{\Gamma \vdash v : A \times B}{\Gamma \vdash \pi_1(v) : A}$ and $\dfrac{\Gamma \vdash A \times B}{\Gamma \vdash \pi_2(v) : B}$

(iv) computation rules: $\pi_1(a, b) \to_\beta a$ and $\pi_2(a, b) \to_\beta b$

(v) uniqueness principle: $(\pi_1(v), \pi_2(v)) \to_\eta v$

Definition 5.18. We can define **Σ-types** which are also called **dependent sum types** using:

(i) formation rule: $\dfrac{\Gamma \vdash A : \text{Type } u \qquad \Gamma \vdash B : A \to \text{Type } v}{\Gamma \vdash \sum_{x:A} B(x) : \text{Type } (\max u\ v)}$

(ii) introduction rule: $\dfrac{\Gamma \vdash a : A \qquad \Gamma \vdash b : B(a)}{\Gamma \vdash (a, b) : \sum_{x:A} B(x)}$

(iii) elimination rules: $\dfrac{\Gamma \vdash v : \sum_{x:A} B(x)}{\Gamma \vdash \pi_1(v) : A}$ and $\dfrac{\Gamma \vdash v : \sum_{x:A} B(x)}{\Gamma \vdash \pi_2(v) : B(\pi_1(v))}$

(iv) computation rules: $\pi_1(a, b) \to_\beta a$ and $\pi_2(a, b) \to_\beta b$

(v) uniqueness rule: $(\pi_1(v), \pi_2(v)) \to_\eta v$

Remark 5.19. The formation rule in the above definition could equivalently be stated as

$$\frac{\Gamma \vdash A : \text{Type } u \qquad \Gamma, x : A \vdash B : \text{Type } v}{\Gamma \vdash \sum_{x:A} B : \text{Type } (\max u\ v)}$$

Remark 5.20. Notice that for $(a, b) : \sum_{a:A} B(a)$, we have $\pi_2(a, b) : B(\pi_1(a, b))$ and $b : B(a)$, i.e. the typing is only the same up to β-equivalence.

Example 5.21. A **magma** is a set (or type) with a binary operation and no axioms. We can write this using Σ-types. The type of a magma is $\sum_{A:\text{Type } u}(A \to A \to A) : \text{Type } (u+1)$. The type of a pointed magma is $\sum_{A:\text{Type } u}(A \to A \to A) \times A : \text{Type } (u+1)$.

Remark 5.22. How can we define a (dependent) function out of $A \times B$? Precisely we want to know:

(i) Given $C : \text{Sort } w$, when is $A \times B \to C$ inhabited?

(ii) Given $C : A \times B \to \text{Sort } w$, when is $\prod_{v:A\times B} C(v)$ inhabited?

For (i), we need $f : A \to B \to C$, which then gives us $g : A \times B \to C, v \mapsto f(\pi_1(v))(\pi_2(v))$. For (ii), we need $f : \prod_{a:A} \prod_{b:B} C(a, b)$ to give us $g : \prod_{v:A\times B} C(v), v \mapsto f(\pi_1(v))(\pi_2(v))$.

Thus, there is a derivable recursion principle

$$\mathrm{rec}_{A\times B} : (A \to B \to C) \to A \times B \to C$$

and a derivable induction principle

$$\mathrm{ind}_{A\times B} : \left(\prod_{a:A}\prod_{b:B} C(a,b)\right) \to \prod_{v:A\times B} C(v).$$

Remark 5.23. When we define a function writing $v \mapsto gv$ we actually mean $\lambda v.gv$.

Definition 5.24. We can define **coproducts** as follows

(i) formation rule: $\dfrac{\Gamma \vdash A : \mathrm{Type}\ u \qquad \Gamma \vdash B : \mathrm{Type}\ v}{\Gamma \vdash A + B : \mathrm{Type}\ (\max u\ v)}$

(ii) introduction rules: $\dfrac{\Gamma \vdash a : A}{\Gamma \vdash \mathrm{inl}(a) : A + B}$ and $\dfrac{\Gamma \vdash b : B}{\mathrm{inr}(b) : A + B}$

(iii) elimination rule:

$$\frac{\Gamma \vdash C : A + B \to \mathrm{Sort}\ w \qquad \Gamma \vdash f : \prod_{a:A} C(\mathrm{inl}(a)) \qquad \Gamma \vdash g : \prod_{b:B} C(\mathrm{inr}(b))}{\Gamma \vdash \mathrm{ind}_{A+B}(C,f,g) : \prod_{v:A+B} C(v)}$$

(iv) computation rules:

$\mathrm{ind}_{A+B}(C,f,g)(\mathrm{inl}\, a) \to_\iota f(a)$ and $\mathrm{ind}_{A+B}(C,f,g)(\mathrm{inr}\, b) \to_\iota g(b)$

(v) there is no uniqueness rule

Definition 5.25. We can also define the **natural numbers**:

(i) formation rule: $\dfrac{}{\Gamma \vdash \mathbb{N} : \mathrm{Type}\ 0}$

(ii) introduction rules: $\dfrac{}{\Gamma \vdash 0 : \mathbb{N}}$ and $\dfrac{\Gamma \vdash n : \mathbb{N}}{\Gamma \vdash \mathrm{s}(n) : \mathbb{N}}$

(iii) elimination rule:

$$\frac{\Gamma \vdash C : \mathbb{N} \to \mathrm{Sort}\ w \qquad \Gamma \vdash f_0 : C(0) \qquad \Gamma \vdash f_\mathrm{s} : \prod_{n:\mathbb{N}} C(n) \to C(\mathrm{s}(n))}{\Gamma \vdash \mathrm{ind}_{\mathbb{N}}(C,f_0,f_\mathrm{s}) : \prod_{n:\mathbb{N}} C(n)}$$

(iv) computation rules: $\mathrm{ind}_{\mathbb{N}}(C,f_0,f_\mathrm{s})(0) \to_\iota f_0$ and $\mathrm{ind}_{\mathbb{N}}(C,f_0,f_\mathrm{s})(\mathrm{s}(n)) \to_\iota f_\mathrm{s}(n)(\mathrm{ind}_{\mathbb{N}}(c,f_0,f_\mathrm{s})(n))$

Remark 5.26. We could also write the second introduction rule in the above definition as $\dfrac{}{\Gamma \vdash \mathrm{s} : \mathbb{N} \to \mathbb{N}}$.

Remark 5.27. To define $g : \prod_{n:\mathbb{N}} C(n)$ we need $g(0) :\equiv_\iota f_0 : C(0)$ and $g(\mathrm{s}(n)) :\equiv_\iota f_\mathrm{s}(n, g(n)) : C(\mathrm{s}(n))$. This way of defining terms is called **pattern matching**.

Definition 5.28. We can define **addition** on $\mathbb{N}$ using pattern matching: to define $n + - : \mathbb{N} \to \mathbb{N}$ we set $n + 0 \coloneqq n$ and $n + \mathrm{s}(m) = \mathrm{s}(n + m)$.

Definition 5.29. We define the **equality type** (also called **indentity type**) as the smallest reflexive relation:

(i) formation rule: $$\frac{\Gamma \vdash A : \mathrm{Sort}\ u \qquad \Gamma \vdash x : A \qquad \Gamma \vdash y : A}{\Gamma \vdash x =_A y : \mathrm{Prop}}$$

(ii) introduction rule: $$\frac{\Gamma \vdash x : A}{\Gamma \vdash \mathrm{refl}_x : x =_A x}$$

(iii) elimination rules:
$$\frac{\Gamma \vdash C : A \to \mathrm{Sort}\ w \qquad \Gamma \vdash v : C(x) \qquad \Gamma \vdash h : x =_A y}{\Gamma \vdash \mathrm{rec}_{=_A}(C, v, h) : C(y)}$$

(iv) computation rule: $\mathrm{rec}_=(C, v, \mathrm{refl}_x) \to_\iota v$

Remark 5.30. We could have also used $\dfrac{}{\Gamma \vdash \prod_{A:\mathrm{Type}\ u} \prod x : A x =_A x}$ as the introduction rule in the above definition.

Remark 5.31. Inductive types are the least/initial types generated by their introduction rules.

Remark 5.32. Lean additionally has:

(i) a general scheme for inductive types

(ii) definitional proof irrelevance: if $P : \mathrm{Prop}$ and $h_1, h_2 : P$, then $h_1 \equiv h_2$.

(iii) a conversion rule: if $x : A$, $A \equiv B$, then $x : B$, where $\equiv$ is the equivalence relation generated by proof irrelevance, (δ-, ζ-), β-, η- and ι-reduction.

(iv) the ability to make new definitions

(v) analogous inductive propositions $\wedge, \vee, \ldots$

(vi) propext : $\prod_{P,Q:\mathrm{Prop}} (P \leftrightarrow Q) \to P =_{\mathrm{Prop}} Q$

(vii) choice : $\prod_{A:\mathrm{Type}\ u} \mathrm{nonempty}(A) \to A$ where $\mathrm{nonempty}(A)$ us the inductive proposition stating that A is non-empty.

Example 5.33. δ- and ζ-reduction relate to unfolding definitions. For the definition

```
def two : ℕ := s(s(0))
```

we get `two` $\to_\delta$ `s(s(0))`. Within a definition you might write

```
let two := s(s(0)),
```

then `two` $\to_\zeta$ `s(s(0))`.