

Logic of Proof Assistants

Prof. Floris van Doorn

University of Bonn

24th April, 2025

Contents

1	Introduction	2
1.1	Inductive Definitions	3
2	First-Order Logic	4
2.1	Provability	6

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, rezk

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. $\pi: \mathbb{R} \rightarrow \mathbb{N}$, $x \mapsto |\{p \leq x \mid p \text{ prime}\}|$. 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**. $c: U^n \rightarrow U$ is called an **n -ary function**.

- (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 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 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 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** $S := \mathcal{F} \cup \mathcal{V} \cup \{ "(", ")", " ", "," \}$ 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 $c_f: (S^*)^n \rightarrow S^*$,
 $c_f(t_1, \dots, t_n) := f("t_1", \dots, "t_n")$

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 **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 **fv**(φ) and **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) $y[t/x] := \begin{cases} t & \text{if } y = x \\ y & \text{otherwise} \end{cases}$

(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 φ **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) $(\forall y.\varphi) := \begin{cases} \forall y.\varphi & \text{if } y = x \\ \forall z.\varphi[z/y][t/x] & \text{otherwise} \end{cases}$

(vii) $(\exists y.\varphi) := \begin{cases} \exists y.\varphi & \text{if } y = x \\ \exists z.\varphi[z/y][t/x] & \text{otherwise} \end{cases}$

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) $(\varphi_1 \rightarrow \varphi_2) \equiv_\alpha (\psi_1 \rightarrow \psi_2)$ 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 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 $\Gamma \vdash \varphi$ (read : “ Γ proves φ ”) is defined inductively by

- (i) $\frac{}{\Gamma, \varphi \vdash \varphi}$ (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 $x \in \text{fv}(\Gamma, \psi)$ (\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 $\varphi \equiv_{\alpha} \psi$ (α -equivalence)