Universität des Saarlandes

Philosophische Fakultät
Bachelor-Studiengang "Historisch orientierte Kulturwissenschaften"
Bachelorarbeit im Kernfach "Theoretische Philosophie"

Foundations of Mathematics:

A Discussion of Sets and Types

Dominik Kirst
Matrikelnummer: 2524622
Wintersemester 2017 / 2018

Advisor
Prof. Dr. Holger Sturm
Department of Philosophy
Saarland University, Germany

Reviewers
Prof. Dr. Holger Sturm
Department of Philosophy
Saarland University, Germany

Prof. Dr. Ulrich Nortmann
Department of Philosophy
Saarland University, Germany

Submitted
11th April 2018

Dominik Kirst
Saarland University
Saarland Informatics Campus
Building E1 3, Room 523
66123 Saarbrücken
kirst@ps.uni-saarland.de

Statement in Lieu of an Oath:

I hereby confirm that I have written this thesis on my own and that I have not used any other media or materials than the ones referred to in this thesis.

Saarbrücken, 11th April 2018

Declaration of Consent:

I agree to make both versions of my thesis (with a passing grade) accessible to the public by having them added to the library of the Philosophy Department.

Saarbrücken, 11th April 2018

Acknowledgments

This project was kindly offered by Prof. Holger Sturm as a bachelor's thesis for my pursued degree in historically oriented cultural studies. He took the time for discussing the strategy underlying this text and for making me sensible for the philosophical questions concerning the foundations often neglected in mathematical practice. I also owe my gratitude to Prof. Ulrich Nortmann for initially arousing my interest in the philosophy of mathematics and for agreeing to act as the second reviewer. Furthermore, I want to thank the numerous people with whom I had fruitful discussions and who contributed their helpful feedback, most notably my colleagues at the Programming Systems Lab and my PhD supervisor Prof. Gert Smolka.

Abstract

The aim of this thesis is to provide an introductory discussion of the two most common modern foundations of mathematics: axiomatic set theory and dependent type theory. To provide a basis, we begin by explaining the main concepts of and motivations for the respective systems in their standard formulation. This includes a brief outline of first-order logic and ZF set theory on the one side, and of simply typed lambda calculus and Martin-Löf type theory on the other side. Subsequently sketching an ongoing debate, both approaches are compared with respect to general criteria for practicality, their respective philosophical justification, and their approximate consistency strength. We argue that, despite being similarly expressive for the purposes of formal mathematics, dependent type theory is based on the more convincing constructivist view and advantageous for computer-assisted proving. To illustrate its potential for modern applications, some parts of the mathematical development are formally verified using the Coq proof assistant.

Contents

Chapter 1
Introduction

After logical inconsistencies had been found in Cantor's naive set theory and in Frege's foundations of arithmetic, the mathematical community of the early 20th century was left aghast. Most famously, these discoveries were due to Cantor himself [Can91], Burali-Forti [BF97], and Russell [Rus02]. Beforehand, it was generally supposed that all mathematical insights accomplished so far could in principle be derived from a compact collection of initial assumptions, embodying the so-called axiomatic method going back to Euclid's Elements. However, after pioneers like Cantor and Frege had run into the apparent problems, the whole enterprise suddenly appeared unstable and the community became aware of a fundamental problem in their current practice of mathematical reasoning. This uncertain period is nowadays dubbed the **foundational crisis of mathematics** and paved the way for an highly productive generation of researchers looking for ways to wipe out the antinomies.

Cantor's naive set theory was based on a very general abstraction principle, allowing to consider every somehow graspable property as to form an actual whole of objects, a so-called set [Can95]. The problem produced by this principle is that the sets corresponding to self-referential properties may give an contradictory answer to the question of which objects they contain as elements. Another common intuition is that the conception of a set as actual whole can only accommodate properties of small extent, leaving big collections like the universe of all objects without reasonable corresponding set. A direct solution to these problems is to restrict the abstraction principle to only guarantee the existence of sets for very specific properties, which is the path leading to **axiomatic set theory**. Among its founders were Zermelo, who formulated a concrete list of axioms in his second proof of the well-ordering theorem [Zer08], and Fraenkel [Fra25], who extended this list by the replacement and foundation axioms. The so created standard system is now called ZF set theory, immortalising its founders.

Another approach was chosen by Russell, solving self-referentiality by introducing the conception of types [Rus08]. Types group mathematical objects into semantic categories such as numbers, functions, and relations. Moreover, functions and relations are again distinguished by the types they operate on, e.g. separating a relation on numbers from a relation on functions. In this approach, a property simply cannot refer to itself as it operates on a type it does not belong to. Together with his colleague Whitehead, Russell created the epochal Principia Mathematica [WR10], a formal development of elementary mathematics based on the theory of types. Although this system appeared highly complicated and was therefore eventually outrun by the arguably simpler ZF set theory, types became integral in the emerging field of theoretical computer science. Extending previous work by Church [Chu40], Girard [Gir72], and Reynolds [Rey74] on typed function calculi, **dependent type theory** as proposed by Martin-Löf [ML85] evolved into a credible foundational alternative to axiomatic set theory.

1.1 What is a Foundation of Mathematics?

To enable a meaningful discussion in this thesis, we first attempt to characterise its key notion. A foundation of mathematics is a formal system in which the body of all mathematical constructions and theorems can be derived from first principles. Such systems come with a precise language for mathematical statements as well as a formal and verifiable notion of proof. Moreover, the underlying logical framework should be rich enough to capture all usual forms of mathematical deduction and avoid contradictions. It is worth explaining these defining criteria in more detail:

- **Universality.** A foundation should really be a basis for *all* of mathematics, meaning that its language is general and flexible enough to accommodate all concepts of all specific branches of mathematics and no area is conceptually preferred. Although not all mathematical objects need to be linguistic primitives, there are clear and elegant encodings with only reasonable technical overhead.

- **Precision.** Statements and derivations can be written down in a purely formal way, pinpointing the intended thought unambiguously. Thus any reader can reconstruct communicated content without misunderstandings, and formal proofs without gaps or flaws can be presented. Furthermore, it is possible to work out and state any additional assumptions a concrete theorem relies on.

- **Effectiveness.** It is algorithmically decidable whether or not given definitions and theorems are well-formed and hence meaningful at all. Most importantly, a formal proof must be algorithmically verifiable such that the corresponding theorem can be used without concern.

- **Consistency.** There should be some evidence suggesting that the logical system is free of contradictions. Although a formal consistency proof is normally not achievable due to Gödel's second incompleteness theorem [Gö31], it should at least be taken care to exclude the obvious logical antinomies. This means that consistency remains an empirical property, making more established foundational systems in principle more trustworthy than niche alternatives.

As it will be sketched in this thesis, both axiomatic set theory and dependent type theory are foundations of mathematics in this formal sense. Thus capable solutions to the foundational crisis could indeed be found and the main debate may focus on identifying satisfactory solutions.

1.2 What is a Satisfactory Foundation of Mathematics?

While the previously discussed criteria describe the very definition of a foundational system, there are further aspects that make such a system **practical** for realistic use. Some of these can be formulated as follows:

- **Accessibility.** It should be possible to teach a foundational system in reasonable time and with reasonable effort. This presupposes a not too complicated and rather intuitive presentation, making the initial hurdle not too discouraging. In general, the more the formal language and deduction system resemble the natural mathematical habits, the easier it is for a working mathematician to adopt to a specific foundation. This includes aspects like the compactness of formal statements and derivations, the legibility of notations, and a basically small distance between informal and formal reasoning.

- **Mechanisability.** In an age dominated by the increasing opportunities of high-performance computers, it is a key advantage of a foundational system to be mechanisable for at least three reasons. First, increasingly complex mathematical problems demand increasingly complex solutions, which may be cumbersome and hard to verify even for experts. Having computers perform **automated proof checking** speeds up the mathematical review process and enhances the trustworthiness of new results. Secondly, **interactive proof assistants** are helpful during proof development, as these programs take care of the formal bookkeeping and hence allow the user to focus on the main ideas. Thirdly, the ideal vision is of course to have a program able to find proofs on its own, having the user only give guidance during the search. Although this vision is naturally hard to fulfil in general, state-of-the-art **proof automation** is on a level where the more calculable goals can be left to the computer.

- **Community.** It is desirable that a foundation of mathematics is not only supported by an active community of both theoretically and practically working researchers but also accepted as standard by the general majority. The former ensures that the system is continuously improved, simplified, and adjusted for the intendedly wide range of applications. The latter guarantees that mathematicians from diverse backgrounds can always discuss formally in a standardised lingua franca without first having to clarify their jargon and notation.

Most importantly, a satisfactory foundation of mathematics rests upon a convincing **philosophical basis** which provides plausible answers to the ontological and epistemic questions concerning mathematical entities and deductions. These answers should include clear conceptions of objects like numbers or functions, their possible existence and infinite extent, as well as logical inference - hence allowing for a meaningful exploration of the mathematical sciences. Particularly, the accepted rules of logical inference and the order of expressible objects have a dramatic impact on the flavour of mathematical practice.

1.3 The Plan of this Thesis

This thesis is supposed to provide an introductory comparison of axiomatic set theory and dependent type theory as the two common approaches to foundations of mathematics. In order to let this comparison penetrate the surface, we begin by explaining the basics of both approaches in Chapter 2 and Chapter 3, respectively. These rather technical chapters are written for an audience with a rough experience in mathematical logic and are presented in a mildly formal language. It is intended that the more experienced reader is able to reconstruct full formal detail, while the less experienced reader can still understand the main ideas. In fact, the tutorial chapters already constitute an evaluation of the accessibility of both systems, in that both explanations are given roughly the same space, allowing for a potentially differently deep understanding.

Subsequently, in Chapter 4, axiomatic set theory and dependent type theory are compared with respect to their practicality, their philosophical basis, and their approximate consistency strength. First, concerning practicality, we evaluate both systems by means of the three aspects we have introduced above. Secondly, we mainly compare the classical first-order logic underlying axiomatic set theory to the intuitionistic higher-order logic expressed by dependent type theory. Thirdly, we show how to interpret set theory in type theory and vice versa, showing that one system in principal proves the consistency of the other.

We end with a few conclusive remarks in Chapter 5 summarising the previous discussion and coming to the result that its general usability, suitability for mechanisation, and constructive philosophy make dependent type theory the more satisfactory foundation of mathematics.

Chapter 2
Axiomatic Set Theory

Axiomatic set theory is usually formulated as a first-order theory, meaning that the set-theoretic axioms are expressed as statements in first-order predicate logic extended by the symbols of set-theoretic language. This approach separates the logical fragment dealing with formulas, their semantics and correct deductions on the one hand from the mathematical component expressing the particular theory. Mirroring this separation, this chapter first introduces predicate logic with its most important concepts and then establishes the basics of ZF set theory as a particular first-order theory. There are lots of introductory textbooks concerning set theory, we refer to [Kun14], [HJ99], [Sup60], and [SF10] for a selection. Also, see [Moo88] and [Kan96] for a historical sketch of first-order logic and set theory.

2.1 First-Order Predicate Logic

2.1.1 Syntax

Predicate logic is a formal language capturing much of the logical structure of natural language. For example, consider the following English sentence:

Every dog has a human owner whose mother has no red hair.

Formalising such a sentence means to replace individuals, functions, and predicates by symbols as well as using the common notation for the logical connectives and quantifiers. Following these steps, an unambiguous formal version could be

$$\forall x.\, d(x) \supset \exists y.\, h(y) \wedge o(y, x) \wedge \neg r(m(y))$$

where x and y are variables for individuals, m denotes the function mapping every individual to their mother, and d, h, o, and r stand for the predicates of being a dog, being a human, ownership, and having red hair. The function and predicate symbols are called **non-logical** symbols as they express semantic concepts rather than purely logical structure. Note that they each come with a concrete **arity**, i.e. number of expected arguments: in our example, almost all are unary, only o is binary.

By fixing a collection S of non-logical symbols, a so-called **signature**, one can adjust the language to the needs of the specific topic of interest. Then formally, the formulas of (first-order) predicate logic are generated inductively in four stages relative to S:

- **Variables.** We simply assume a sufficient supply of variable symbols such as x, y, z, x', x_1, etc. Actually, starting from a single symbol x and employing a rule generating variables x', x'', x''', etc. is enough.

- **Terms.** Every variable is a term. If $t_1, \ldots, t_k$ are terms and f is a k-ary function symbol in S, then $f(t_1, \ldots, t_k)$ is a term.

- **Atomic formulas.** If t_1 and t_2 are terms, then $t_1 = t_2$ is an atomic formula. Moreover, if $t_1, \ldots, t_k$ are terms and p is a k-ary predicate symbol in S, then $p(t_1, \ldots, t_k)$ is an atomic formula.

- **Formulas.** Every atomic formula is a formula. If ϕ and ψ are formulas, then $\phi \wedge \psi$, $\phi \vee \psi$, $\phi \supset \psi$, and $\neg\phi$ are formulas. Moreover, if x is a variable, then $\forall x.\phi$ and $\exists x.\phi$ are formulas.

The example formula from above, let's call it ϕ_E, can indeed be generated from these syntax rules. In fact, it is an easy problem to decide whether or not an arbitrary string of symbols is a well-formed formula or not.

In the forthcoming development, we freely adopt the common conventions to express the tree-like structure of formulas using parentheses regarding associativity and operator precedence. Moreover, if not indicated otherwise by parentheses, the quantifier scope is maximal.

Note that the given syntactic rules impose strong restrictions on the available formulas. Functions and predicates operate only on terms and only produce terms and atomic formulas, respectively. Moreover, quantification is restricted to the solely available individual variables. These restrictions distinguish first-order logic as sketched here from higher-order logics, e.g. allowing for applying functions to functions and for quantifying over relations. Although first-order logic hence surely expresses not all of the logical structure hidden in natural language, however, (combined with set-theoretic assumptions) it is still sufficient to capture most of the statements needed in ordinary mathematics.

2.1.2 Semantics

Having our example sentence reduced to the purely syntactic version ϕ_E, its status is completely unclear if we happen to forget the intended use of the non-logical symbols. In the original meaning, the statement is certainly false as not every dog even does have an owner. However, if we alter d to denote being a human as well and o to express friendship, then the English version becomes

Every human has a human friend whose mother has no red hair.

which might actually be true. Also, we have not at all qualified which domain the individuals belong to, but this knowledge is again crucial for the meaning of ϕ_E. So far, the intended domain could have been the collection of all living creatures. However, we could as well restrict quantification to humans only, then ϕ_E holds vacuously since no human is a dog and hence all humans who are dogs have human owners whose mothers have no red hair.

This means that a formula is only meaningful in a fixed domain D of individuals together with a symbol assignment A mapping the k-ary function and predicate symbols to k-ary functions and predicates on D, respectively. Both ingredients together form an **interpretation** $M = (D, A)$. Then following Tarski [Tar43], the predicate $M \models \phi$ expressing that M satisfies ϕ is defined inductively:

- **Terms.** Variable-free terms can be evaluated to concrete individuals. Specifically, if terms $t_1, \ldots, t_k$ have values $d_1, \ldots, d_k$ and f is a k-ary function symbol, then $f(t_1, \ldots, t_k)$ evaluates to $A_f(d_1, \ldots, d_k)$. Here A_f denotes the k-ary function corresponding to the symbol f under the assignment A.
- **Equality.** $M \models t_1 = t_2$ if t_1 and t_2 have the same value d of D.
- **Predication.** If terms $t_1, \ldots, t_k$ have values $d_1, \ldots, d_k$ and p is a k-ary predicate symbol, then $M \models p(t_1, \ldots, t_k)$ if A_p holds for $(d_1, \ldots, d_k)$. Here A_p denotes the k-ary predicate corresponding to the symbol p under the assignment A.
- **Connectives.** The meaning of the logical connectives is defined according to the usual truth tables. For instance, $M \models \phi \wedge \psi$ if $M \models \phi$ and $M \models \psi$.
- **Quantification.** $M \models \exists x.\phi$ if there is an individual d in the domain D such that $M \models \phi[c_d/x]$. Here $\phi[c_d/x]$ denotes the formula ϕ with all occurrences of the variable x replaced by a new 0-ary function symbol c_d corresponding to d. Similarly, $M \models \forall x.\phi$ if $M \models \phi[c_d/x]$ for all individuals d.

Note that this formal definition confirms the informal interpretations of ϕ_E above. If an interpretation satisfies a formula, we call it a **model** and formulas that are satisfied by every interpretation are called **valid** and express logical tautologies. Then ϕ_E is not valid as its truth really depends on the chosen interpretation.

However, consider the following two formulas in the same signature as ϕ_E:

$$\phi_A \coloneqq \forall x.d(x) \supset \exists y.h(y) \wedge o(y,x)$$
$$\phi_B \coloneqq \forall x.h(x) \supset \neg r(m(y))$$

As they formulate two conditions implying ϕ_E, one can show that the formula $\phi_A \wedge \phi_B \supset \phi_E$ is in fact valid by following the above definition for an arbitrary interpretation M. This means that we can think of ϕ_A and ϕ_B as axioms constituting a theory that has ϕ_E as a theorem.

2.1.3 Deduction

As purely semantical reasoning for the validity of formulas such as $\phi_A \wedge \phi_B \supset \phi_E$ is done in natural language and hence subject to the usual ambiguities and communication flaws, it is a main concern of formal logic to provide for a purely syntactical deduction system. We here outline the idea of a system in the flavour of natural deduction with introduction and elimination rules for all logical symbols as introduced by Gentzen [Gen35]. Consider the following rules for conjunction and implication:

$$\frac{\Gamma \vdash \phi \quad \Gamma \vdash \psi}{\Gamma \vdash \phi \wedge \psi}\ (\wedge_I) \qquad \frac{\Gamma \vdash \phi \wedge \psi}{\Gamma \vdash \phi}\ (\wedge_{E_1}) \qquad \frac{\Gamma, \phi \vdash \psi}{\Gamma \vdash \phi \supset \psi}\ (\supset_I) \qquad \frac{\Gamma \vdash \phi \supset \psi \quad \Gamma \vdash \phi}{\Gamma \vdash \psi}\ (\supset_E)$$

Such inference rules list on top of the bar the necessary assumptions to conclude the statement below the bar. The statements in natural deduction are of the form $\Gamma \vdash \phi$, indicating that from a list Γ of assumed formulas a formula ϕ is provable. The rule $(\wedge_I)$ states that a conjunctive formula is provable if both its subformulas are provable. Conversely, if a proof of a conjunction could be derived, the rule $(\wedge_{E_1})$ can be used to obtain a proof of the left subformula (and likewise there is a rule $(\wedge_{E_2})$ for the right subformula). By $(\supset_I)$, an implication $\phi \supset \psi$ is provable whenever ψ is provable if ϕ is added to the assumptions, and $(\supset_E)$ transforms evidence for $\phi \supset \psi$ and ϕ into evidence for ψ.

A proof of a composite formula is a derivation tree using the rules of natural deduction. We do not give the full system here but encounter a few more rules considering a (simplified) derivation tree for $\phi_A \wedge \phi_B \supset \phi_E$ from no assumptions:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\phi_A \wedge \phi_B, d(x_0) \vdash \phi_A \wedge \phi_B}\,(\Gamma)}{\phi_A \wedge \phi_B, d(x_0) \vdash \forall x.d(x) \supset \exists y.h(y) \wedge o(y,x)}\,(\wedge_{E_1})}{\phi_A \wedge \phi_B, d(x_0) \vdash d(x_0) \supset \exists y.h(y) \wedge o(y,x_0)}\,(\forall_E) \quad \dfrac{}{\phi_A \wedge \phi_B, d(x_0) \vdash d(x_0)}\,(\Gamma)}{\phi_A \wedge \phi_B, d(x_0) \vdash \exists y.h(y) \wedge o(y,x_0)}\,(\supset_E)}{\phi_A \wedge \phi_B, d(x_0) \vdash h(y_0) \wedge o(y_0,x_0)}\,(\exists_E)}{\vdots}}{\phi_A \wedge \phi_B, d(x_0) \vdash h(y_0) \wedge o(y_0,x_0) \wedge \neg r(m(y_0))}\,(\exists_I)}{\phi_A \wedge \phi_B, d(x_0) \vdash \exists y.h(y) \wedge o(y,x_0) \wedge \neg r(m(y))}\,(\supset_I)}{\phi_A \wedge \phi_B \vdash d(x_0) \supset \exists y.h(y) \wedge o(y,x_0) \wedge \neg r(m(y))}\,(\forall_I)}{\phi_A \wedge \phi_B \vdash \phi_E}\,(\supset_I)}{\vdash \phi_A \wedge \phi_B \supset \phi_E}$$

From bottom to top, the target formula is first decomposed using several introduction rules. The universal quantification can be introduced if the respective formula holds for arbitrary x_0 and the existential quantification holds if there is a specific witness y_0. In the omitted part one basically derives that $\neg r(m(y_0))$ follows from $h(y_0)$ using the assumption ϕ_B. Above, the witness y_0 is obtained by eliminating an existential quantifier and the general statement for x_0 is obtained by eliminating a universal quantification. The leafs end the derivation as the respective statements occur in the list of assumptions.

Along those lines, a **sound** and **complete** deduction system for first-order predicate logic can be given, meaning that it proves only valid formulas and that every valid formula is provable, respectively. The more involved completeness theorem is due to Gödel [Göd30] and implies further useful properties of first-order logic such as the compactness and model existence theorems.

2.2 ZF Set Theory

2.2.1 Axioms

Having the logical framework all set, we can now develop axiomatic set theory as a specific first-order theory. As outlined above, the first step is to fix a signature listing the non-logical symbols the language should comprise. It might come as a surprise that requiring a single binary predicate symbol $\in$, usually written infix, suffices. The other set-theoretic symbols such as $\subseteq, \emptyset, \cup$, or $\mathcal{P}$ can all be defined within the theory, for instance $x \subseteq y$ is an abbreviation for $\forall z.z \in x \supset z \in y$.

The second step is to pose the axioms that constitute the deductive base of the theory. ZF set theory as developed by Zermelo [Zer08] and Fraenkel [Fra25] (in principle) consists of nine axioms.

Extensionality: $\forall xy.(\forall z.z \in x \leftrightarrow z \in y) \supset x = y$

This states that sets are extensional in the sense that they are uniquely determined by their elements. The next few axioms state the existence of some specific sets:

Empty Set: $\exists x.\forall y.y \notin x$

Pairing: $\forall x, y.\exists z.\forall u.u \in z \leftrightarrow u = x \vee u = y$

Union: $\forall x.\exists z.\forall u.u \in z \leftrightarrow \exists y \in x.u \in y$

Power Set: $\forall x.\exists z.\forall u.u \in z \leftrightarrow u \subseteq x$

Although it is no formal 0-ary function symbol in our signature, we use $\emptyset$ in formulas to refer to the set guaranteed by the empty set axiom. This is justified since every formula containing $\emptyset$ can be translated into an equivalent formula referring to the defining property instead. Similarly, we use the symbols $\{x, y\}$ for the unorderd pair, $\bigcup x$ for the union, and $\mathcal{P}(x)$ for the power set given by the respective axioms. We can even introduce further symbols such as $\{x\}$ for $\{x, x\}$, $x \cup y$ for $\bigcup \{x, y\}$, and $\bigcap x$ for $\{\, z \in \bigcup x \mid \forall y \in x.\, z \in y \,\}$.

The operations so far only allow for constructing finite sets such as $\{\emptyset\}$ or $\{\{\emptyset\}, \emptyset\}$. Since this is surely not enough to express a lot of mathematics, we explicitly assume an infinite set as a starting point for larger sets:

Infinity: $\exists x.\emptyset \in x \wedge (\forall y \in x.\, y \cup \{y\} \in x)$

The next two axioms are in fact schemes with one instance for every formula ϕ.

Separation: $\forall x.\exists y.\forall u.u \in y \leftrightarrow u \in x \wedge \phi(u)$

Replacement: $\psi \supset \forall x.\exists y.\forall v.v \in y \leftrightarrow \exists u \in x.\phi(u, v)$

Separation and replacement clarify the existence of subsets $\{\, y \in x \mid \phi(y) \,\}$ and image sets $\{\, z \mid \exists y \in x.\, \phi(y, z) \,\}$, respectively. In the statement of the replacement axiom, the formula ψ expresses that ϕ is functional, which can be expressed formally by $(\forall u, v, v'.\phi(u, v) \wedge \phi(u, v') \rightarrow v = v')$. Including these two schemes, ZF actually consists of infinitely many axioms, however, given their particular structure it is still an easy problem to decide whether a given formula is an axiom.

Finally, it is custom for ZF-like axiomatisations to exclude non-well-founded membership as in $x \in x$, $x \in y \in x$, or any other infinitely decreasing chain of membership judgements. A first-order statement excluding such behaviour can be given as follows:

Regularity. $\forall x.x \neq \emptyset \supset \exists y \in x.\forall z \in y.z \notin x$

2.2.2 Relations and Functions

As the only objects available in ZF set theory are sets, the ordinary mathematical objects such as functions and numbers need to be encoded. The base for the former is the **cartesian product** containing **ordered pairs**. The standard encoding of ordered pairs is due to Kuratowski [Kur21] and sets $(x, y) \coloneqq \{\{x\}, \{x, y\}\}$. Then the cartesian product $X \times Y$ is the set containing all ordered pairs (x, y) with $x \in X$ and $y \in Y$. Formally, it can be defined by

$$X \times Y \coloneqq \bigcup_{x \in X} \{\, p \mid \exists y \in Y.p = (x, y) \,\}$$

The set tight to the union is an instance of the replacement scheme as $p = (x, y)$ is a functional property and forms all pairs with first component x. Then the indexed union is an abbreviation for taking the union after replacing every $x \in X$ with its corresponding set of pairs, hence obtaining all possible combinations.

Moving on, a **set-theoretic relation** R on sets X and Y is nothing but a subset of the cartesian product $X \times Y$. Moreover, a **set-theoretic function** is a relation $f \subseteq X \times Y$ with a special property, namely that for every $x \in X$ there is a unique $y \in Y$ with $(x, y) \in f$. The latter is normally written $f(x) = y$ and the set of functions from X to Y is denoted by $X \to Y$. One can in fact give a formal definition of $f(x)$ in terms of the set operations by setting $f(x) \coloneqq \bigcup \{\, y \mid \exists p \in f.p = (x, y) \,\}$.

As examples, we define the projections $\pi_1 \in X \times Y \to X$ and $\pi_2 \in X \times Y \to Y$:

$$\pi_1(p) \coloneqq \bigcup \bigcap p \qquad \pi_2(p) \coloneqq \bigcup \{\, z \in \bigcup p \mid \forall y, y' \in p.\, y \neq y' \to z \notin y \vee y \notin y' \,\}$$

Here we used a point-wise pattern for defining a function $f \in X \to Y$ by just specifying a term $t(x)$ and setting $f \coloneqq \{\, p \in X \times Y \mid \exists x \in X.p = (x, t(x)) \,\}$. The projections then satisfy the properties $\pi_1((x, y)) = x$, $\pi_2((x, y)) = y$, and $p = (\pi_1(p), \pi_2(p))$ if p is an ordered pair. In usual practice, these equations can be shown by simply plugging in the respective definitions and arguing by equational reasoning, but it is important to note that one could also give fully formal proofs in the language of the previously discussed natural deduction system. This is in fact the practice we adopt in the following section on natural numbers by giving informal statements and arguments which, in principle, always have formal counterparts.

2.2.3 Natural Numbers

The standard way to encode the numbers in set theory is due to von Neumann [vN23], setting $0 \coloneqq \emptyset$ and $\sigma(n) \coloneqq n \cup \{n\}$. That some set containing all natural numbers exists is guaranteed by the infinity axiom and one defines the set ω of natural numbers to be the smallest such set. Formally, if ω' is the set provided by the infinity axiom and $\psi(x)$ is the formula asserting that x contains 0 and is closed under σ, then one sets $\omega \coloneqq \bigcap \{\, x \in \mathcal{P}(\omega') \mid \psi(x) \,\}$. In particular, given that the set of natural numbers exists, the successor operation defines an actual function $\sigma \in \omega \to \omega$. In order to verify that this encoding of numbers is meaningful, we justify the Peano axioms:

- **Disjointness.** We first prove that the successors are disjoint from zero, meaning that there is no $n \in \omega$ with $\sigma(n) = 0$. So let $\sigma(n) = 0$ hold for some $n \in \omega$. By the construction of σ we have $n \in \sigma(n) = n \cup \{n\}$ as $n \in \{n\}$. but then $n \in 0 = \emptyset$, which contradicts the empty set axiom.

- **Injectivity.** We next prove that the successor function is injective, that is $\sigma(n) = \sigma(n')$ implies $n = n'$ for all $n, n' \in \omega$. By the extensionality axiom, we have to prove two inclusions. For $n \subseteq n'$, we suppose $x \in n$ and show $x \in n'$. Since $n \subseteq \sigma(n)$ it follows that $x \in \sigma(n)$ and hence $x \in \sigma(n')$. This leaves us with two cases, either the wished $x \in n'$ or $x \in \{n'\}$ and hence $x = n'$. The latter leads to a contradiction of the regularity axiom and hence we can conclude $n \subseteq n'$. Analogously we obtain the other inclusion $n \subseteq n'$ and thus $n = n'$.

- **Induction.** Finally, the set-theoretic induction principle for natural numbers basically expresses that ω is the smallest set containing all numbers, which was the very construction of ω. Formally, we have to show for every formula ϕ satisfying $\phi(0)$ and $\forall n.\phi(n) \supset \phi(n+1)$ it holds that $\phi(n)$ for every $n \in \omega$. For such a formula ϕ, it clearly holds that $\{\, n \in \omega' \mid \phi(n) \,\}$ has the property ψ from above, so $\omega \subseteq \{\, n \in \omega' \mid \phi(n) \,\}$ and hence $\phi(n)$ for every $n \in \omega$.

  There is also the slightly stronger **complete induction principle**, stating that $\phi(n)$ holds for all numbers n if $\phi(n)$ can be proved whenever $\phi(n')$ holds for all $n' \in n$. Although we omit the rather technical formal proof at this point, we take the freedom to employ this form of induction where necessary.

To reach the point where we can express all of number arithmetic, it remains to define the less-than ordering and functions like addition and multiplication. Given the particular encoding, a number is exactly the set of all previous numbers. Hence $m < n$ can be simply defined as $m \in n$, more formally, we define a relation $< \subseteq \omega \times \omega$ by $< \coloneqq \{\, p \in \omega \times \omega \mid \pi_1(p) \in \pi_2(p) \,\}$. Moreover, we define the non-strict counterpart $\leq \coloneqq \{\, p \in \omega \times \omega \mid \pi_1(p) \subseteq \pi_2(p) \,\}$. In order to justify these definitions, we prove that $\in$ restricted to ω is a **well-ordering**.

- **Asymmetry.** If $m \in n$ it cannot be that $n \in m$ as this contradicts regularity.
- **Transitivity.** We first show by induction that $x \subseteq n$ for all $x \in n$ and $n \in \omega$. The claim holds trivially for 0 since $\emptyset$ has no elements. So suppose n is transitive and $x \in \sigma(n)$. Then either $x \in n$ or $x = n$. If $x \in n$ we know that $x \subseteq n$ since n is transitive and if $x = n$ then $x \subseteq n$ holds trivially. Now given that every number is transitive, supposing $n_1 \in n_2 \in n_3$ implies $n_2 \subseteq n_3$ and hence $n_1 \in n_3$.
- **Trichotomy.** The statement that for any two numbers m and n one of $m \in n$, $m = n$, or $n \in m$ holds follows by a nested induction on both m and n. Since this is rather technical, we omit the formal proof here.
- **Well-Foundedness.** Let x be a non-empty subset of ω, so assume $n_0 \in x$. We have to find a least element $n \in x$, meaning that $n \subseteq n'$ for all $n' \in x$. By applying the complete induction principle, we may assume that x has a least element if we find some $n \in x$ with $n \in n_0$. Now we distinguish two cases. Either n_0 is already the least element of x, then we are done. Otherwise, there must be $n \in x$ with $n_0 \not\subseteq n$. Then by the trichotomy $n \in n_0$ must hold and the inductive assumption applies.

Addition and Multiplication are usually defined by giving recursive equations:

$$\begin{aligned} m + 0 &\coloneqq m & \qquad m \cdot 0 &\coloneqq 0 \\ m + \sigma(n) &\coloneqq \sigma(m + n) & \qquad m \cdot \sigma(n) &\coloneqq m + m \cdot n \end{aligned}$$

It takes some effort to show that one can really define functions $+$ and $\cdot$ from $\omega \times \omega$ to ω satisfying these equations. The solution is given by the following **recursion theorem**:

Theorem (Recursion). *Assume a base value $n_0 \in \omega$ and a step function $s \in \omega \to \omega$. There exists a unique function $f \in \omega \to \omega$ such that $f(0) = n_0$ and $f(\sigma(n)) = s(f(n))$.*

Before we prove the recursion theorem, we show how it is applied to define addition and multiplication. We fix a number m and pick the base value $n_0 \coloneqq m$ and the step function $s \coloneqq \sigma$. Then the recursion theorem yields a function f_m with $f_m(0) \coloneqq m$ and $f_m(\sigma(n)) = \sigma(f_m(n))$. So on any input n, f_m adds m to n. Hence to obtain the full binary addition function, we simply set $m + n \coloneqq f_m(n)$. Formally, we actually define a function $f \in \omega \to (\omega \to \omega)$ where $f(m) \coloneqq f_m$ and set $m + n \coloneqq f(m)(n)$. Multiplication is obtained similarly for base value $n_0 \coloneqq 0$ and step function $s(n) \coloneqq m + n$. Moreover, one could continue and define exponentiation for base value $n_0 \coloneqq 1$ and step function $s(n) \coloneqq m \cdot n$.

We end our development of the set-theoretic approach to the foundations of mathematics with a proof of the recursion theorem.

Proof of the recursion theorem. We have to prove two things, namely existence and uniqueness of a function $f \in \omega \to \omega$ satisfying the recursive equations stated in the theorem. Beginning with the uniqueness part, assume two such functions f and g. As functions encoded as sets of pairs $(x, f(x))$ are extensional, we just have to prove that $f(n) = g(n)$ for every $n \in \omega$. We apply the induction principle, so we have to show $f(0) = g(0)$ and $f(\sigma(n)) = g(\sigma(n))$ whenever $f(n) = g(n)$. Both follow from the specification as f and g are solutions to the recursion equations: the former since $f(0) = n_0 = g(0)$, the latter since $f(\sigma(n)) = s(f(n)) = s(g(n)) = g(\sigma(n))$.

We now prove the existence of f. To this end, we say that a **computation** on n is a function $c \in \sigma(n) \to \omega$ satisfying the recursive equations from above, so $c(0) = n_0$ and $c(\sigma(n')) = s(c(n'))$ for all $n' \in n$. Note that by setting $\sigma(n)$ as domain for c, we again exploit that numbers are the set of all previous numbers. Now we first prove by induction that for every n there is a computation on n. In the base case, we simply set $c \coloneqq \{(0, n_0)\}$. It is easy to see that this defines a function $c : 1 \to \omega$ as the single element 0 of the domain 1 is mapped to a unique value n_0. Moreover, c clearly satisfies the recursive equations, so c defines a computation on 0. Now suppose that c' is a computation on n. Then we set $c \coloneqq c' \cup \{(\sigma(n), s(c'(n))\}$ which defines a function $c \in \sigma(\sigma(n)) \to \omega$ that agrees with c' on all $n' \leq n$ and additionally maps $\sigma(n)$ to $s(c'(n))$. It follows that c is a computation on $\sigma(n)$.

Now by an analogous argument as above, for every number n there can only be a single computation on n. Hence we can refer to this computation via c_n or, more formally, have a function $C \in \omega \to (\omega \to \omega)$ with $C(n) \coloneqq c_n$ that maps every number to its respective computation. Then we can set $f(n) \coloneqq C(n)(n)$ to define a function $f \in \omega \to \omega$ and establish the recursive equations. Indeed $f(0) = C(0)(0) = n_0$ since $C(0)$ is a computation on 0. Moreover $f(\sigma(n)) = C(\sigma(n))(\sigma(n)) = s(C(n)(n)) = s(f(n))$ using that $C(\sigma(n))$ is a computation on $\sigma(n)$ and that computations on n are unique. □

Chapter 3
Dependent Type Theory

The type theoretic approach to foundations of mathematics is based on rather different primitives than axiomatic set theory. At its core lies the simply typed lambda calculus, an abstract theory of functions and computation, which can be thought of as a simple programming language. In this language, terms are associated with particular types, distinguishing their behaviour as specific sorts of functions or individuals, and only well-typed expressions are admitted. Instead of a clear separation of logical formulas and mathematical objects, logical language arises as an internal concept expressed by the typing rules of the calculus. By gradually introducing further constructions on types and terms in the style of Martin-Löf type theory, the internal logic amounts to higher-order predicate logic and the basic mathematical objects and properties become expressible. We refer to [BDS13], [GTL89], [ML85], and [Luo18] for introductory material.

3.1 Simply Typed Lambda Calculus

3.1.1 Syntax

The primitive notion of the lambda calculus is that of a syntactic function describing an algorithmic procedure. As a motivating example, we consider the identity function I on the natural numbers. This function comes with a clear procedural intuition: it simply returns any given input. However, in set-theoretic language, I is encoded as the set of all pairs (n, n) of numbers $n \in \mathbb{N}$ and the more informative point-wise definition $I(n) \coloneqq n$ is a mere abbreviation for the formal encoding. Moreover, applying the obtained function I to the concrete number 3 by writing $I(3)$ is again an abbreviation for a set-theoretic operation not at all capturing the procedural behaviour.

The lambda calculus starts from the exactly opposite perspective, being centred around the very meaning of a procedure. Primitive to the syntax are exactly the point-wise function definitions and applications, that were indirect in set theory. For instance, the identity function is defined by $I \coloneqq \lambda x.x$, where the λ introduces an argument variable x that is used in the trivial procedure described in the subterm. Then applying the function I to another term s is again expressed by a primitive term, normally written $I\,s$, usually without the use of parentheses. Formally, the terms of the lambda calculus are defined by three syntax rules:

- **Variables.** Every variable $x, y, z, x', x_1, \ldots$ is a term.
- **Applications.** Ifs s and t are terms, then $s\,t$ is a term.
- **Abstractions.** If s is a term and x is a variable, then $\lambda x.s$ is a term.

As we did with the quantifiers in logical formulas, we group together prefixes of lambdas by writing $\lambda xy.s$ for $\lambda x.\lambda y.s$. We consider two further terms:

$$K \coloneqq \lambda xy.x \qquad\qquad S \coloneqq \lambda xyz.x\,z\,(y\,z)$$

The term K describes a procedure taking two arguments and returning the first one. A bit more interesting, the procedure defined in S takes another procedure x as argument and applies it to the third argument z as well as to the result of applying the second argument y to z.

3.1.2 Computation

The meaning of a lambda term is given by its computational behaviour. For instance, consider the application $I\,s = (\lambda x.x)\,s$ for some term s. As I is meant to express the identity function, we expect its application to s just to return s. Formally, this is captured by replacing the single occurrence of the argument variable x in the body of I by the argument s, written by $I\,s \succ x[s/x] = s$. The notation $[s/x]$ abbreviates the substitution of x by s as before and $\succ$ denotes a **reduction step**. The general definition of reductions $s \succ t$ is given by the following inductive rules:

$$\frac{}{(\lambda x.s)t \succ s[t/x]} \qquad \frac{s \succ s'}{s\,t \succ s'\,t} \qquad \frac{t \succ t'}{s\,t \succ s\,t'} \qquad \frac{s \succ s'}{\lambda x.s \succ \lambda x.s'}$$

Note that the example $I\,s \succ s$ was an application of the first rule. The other rules simply assert that reduction works in the context of composite terms. Also consider the following more interesting example of a reduction sequence

$$S\,K\,K \succ^* \lambda z.K\,z\,(K\,z) \succ^* \lambda z.z$$

where $\succ^*$ is an abbreviation for several reduction steps. From the left to the middle, we took two steps for simply plugging in the two arguments into S. From the middle to the right, we then took two steps below the outer lambda, evaluating the application of K to z and $(K\,z)$. The obtained term $\lambda z.z$ is the same as I up to a renaming of the variables. In fact, it is common practice to identify such terms, so we in total obtain that $S\,K\,K \succ^* I$. Moreover, since I cannot reduce any further, we say that I is a **normal form** and consequently that $S\,K\,K$ has the normal form I. Not every term necessarily has a normal form, the standard example is $\Omega \coloneqq (\lambda x.x\,x)(\lambda x.x\,x)$ where $\Omega \succ \Omega$.

The most important property of the obtained reduction system is **confluence**: whenever $s \succ^* t$ and $s \succ^* t'$ there is a term u with $t \succ^* u$ and $t' \succ^* u$. A corollary is the uniqueness of normal forms, meaning that $t = t'$ whenever t and t' are normal forms of a term s. This shows that the computation expressed by term reductions is well-behaved, as it always yields the same value independent of the chosen evaluation order.

Seen as a computational model, the lambda calculus is in fact as expressible as any other programming language. In fact, it can shown to be **Turing complete**, meaning that every effectively computable algorithm can be implemented as lambda term. One proof of this statement proceeds by encoding natural numbers, booleans and unbounded recursion directly as terms, allowing for computing all general recursive functions. Making this idea precise is beyond the scope of this thesis and we refer to the relevant literature, for instance the famous article by Turing [Tur36]. We study an alternative type-theoretic construction of the natural numbers in Section 3.2.

3.1.3 Types

In the introductory example for this section, we actually introduced I as the identity function on natural numbers. However, after giving the formal syntax, the term I lost this semantic aspect and now operates on arbitrary terms. In the remainder of this section, we recover the idea of functions for specific arguments by introducing types. To this end, we first extend the syntax to accommodate type constructions and annotations.

- **Base Types.** We assume some base types a, b, c, etc.
- **Function Types.** If A and B are types, then so is the function type $A \to B$.
- **Type Annotations.** Abstractions now have the form $\lambda x : A.s$ for a type A.

Types are a means to categorise the terms into semantic classes such as the coarse distinction into functions and individuals. For instance, we could now assume some encoding of the natural numbers as a type $\mathbb{N}$ and consider the specific term $I_{\mathbb{N}} \coloneqq \lambda n : \mathbb{N}.n$. Then $I_{\mathbb{N}}$ should then be associated with the function type $\mathbb{N} \to \mathbb{N}$, making it applicable only to terms of type $\mathbb{N}$ and producing results of type $\mathbb{N}$.

The fact that a term s has type A is formalised by a judgement $\Gamma \vdash s : A$, where Γ is a context assigning some variables to types. The typing judgements are defined inductively by rules following the term syntax:

$$\frac{\Gamma, (x : A) \vdash s : B}{\Gamma \vdash (\lambda x : A.s) : A \to B}\ (\lambda) \qquad \frac{\Gamma \vdash s : A \to B \quad \Gamma \vdash t : A}{\Gamma \vdash s\,t : B}\ (a) \qquad \frac{(x : A) \in \Gamma}{\Gamma \vdash x : A}\ (v)$$

By the rule (λ), an abstraction $\lambda x : A.s$ has the function type $A \to B$ whenever binding the variable x to type A yields type B for the body s. Conversely, an application $s\,t$ has type B whenever s has a function type $A \to B$ and t the matching argument type A by the rule (a). Finally, the rule (v) yields a type A for a variable x if a corresponding assignment is in the context.

Now by single applications of (λ) and (v) we can derive the typing judgement $\vdash I_{\mathbb{N}} : \mathbb{N} \to \mathbb{N}$ as desired. To see a more involved example, we derive a type for an annotated version of S:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma \vdash x : A \to B \to C}\,(v) \quad \dfrac{}{\Gamma \vdash z : A}\,(v)}{\Gamma \vdash x\,z : B \to C}\,(a) \quad \dfrac{\dfrac{}{\Gamma \vdash y : A \to B}\,(v) \quad \dfrac{}{\Gamma \vdash z : A}\,(v)}{\Gamma \vdash y\,z : B}\,(a)}{\Gamma \vdash x\,z\,(y\,z) : C}\,(a)}{\vdash S_{A,B,C} : (A \to B \to C) \to (A \to B) \to A \to C}\,(\lambda)$$

Here $S_{A,B,C} \coloneqq \lambda(x : A \to B \to C)(y : A \to B)(z : A).x\,z\,(y\,z)$ and Γ consists of the pairs $(x : A \to B \to C)$, $(y : A \to B)$, and $(z : A)$.

The simply typed lambda calculus consists of all terms that are typeable using the given rules. It adds further remarkable properties to the confluent reduction behaviour we have already encountered in the untyped case:

- **Unique Types.** If $\Gamma \vdash s : A$ and $\Gamma \vdash s : A'$, then $A = A'$. This means that any term can only be associated to a single type and is due to the annotations at the argument variables. Hence, $I_{\mathbb{N}}$ solely operates as the identity function on the natural numbers.
- **Subject Reduction.** If $\Gamma \vdash s : A$ and $s \succ t$, then also $\Gamma \vdash t : A$. This means that the type of a term is preserved under reduction. Hence, applying a numeric function of type $\mathbb{N} \to \mathbb{N}$ such as $I_{\mathbb{N}}$ to a term of type $\mathbb{N}$ reduces only to terms of type $\mathbb{N}$.
- **Strong Normalisation.** Every reduction sequence terminates. Hence, it is guaranteed that every term evaluates to a normal form, independent of the reduction order.

The latter property implies that non-terminating terms like Ω are not typeable. In fact, all terms necessary to express unbounded recursion are not typeable, meaning that the simply typed lambda calculus is not Turing-complete.

3.1.4 Internal Propositional Logic

Given the well-behaved interaction of typing and computation, the simply typed lambda calculus qualifies as a meaningful framework for the theory of functions. That it can also be seen as a logical system may come as a surprise and is the basic insight underlying all foundational type theories. Comparing the typing rules for abstraction and application to introduction and elimination of implication in natural deduction unveils the analogy:

$$\frac{\Gamma, (x : A) \vdash s : B}{\Gamma \vdash (\lambda x : A.s) : A \to B}\ (\lambda) \qquad \frac{\Gamma, \phi \vdash \psi}{\Gamma \vdash \phi \supset \psi}\ (\supset_I)$$

$$\frac{\Gamma \vdash s : A \to B \quad \Gamma \vdash t : A}{\Gamma \vdash s\, t : B}\ (a) \qquad \frac{\Gamma \vdash \phi \supset \psi \quad \Gamma \vdash \phi}{\Gamma \vdash \psi}\ (\supset_E)$$

Indeed, these rules are identical if we interpret implications as function types and leave out the terms. The former is known as the **propositions-as-types** or **Curry-Howard correspondence** [Cur34, Cur58, How80, Wad15]. Under this interpretation, a proposition is seen as the type of its proofs, implying that a proposition is provable whenever its corresponding type is inhabited.

Considering the terms, we have just encountered a first instance of the so-called **proofs-as-programs** or **Brouwer-Heyting-Kolmogorov interpretation** [Hey34, Kol32]. If we interpret A as a data type, then the term I_A expresses the now very familiar identity procedure of type $A \to A$. On the other hand, interpreting A as a logical proposition, I_A can be seen as a proof of the tautology $A \to A$ since the typing derivation for $\vdash I_A : A \to A$ is isomorphic to the natural deduction proof of $\vdash \phi \supset \phi$ for some formula ϕ. More generally, following the proofs-as-programs interpretation, a proof of any implication $\phi \supset \psi$ is a function transforming a proof of ϕ into a proof of ψ.

As it solely comes with base and function types, the simple type system studied so far only covers the implicative fragment of propositional logic. In the following section, we will enrich the system with more complex types and terms such as sums and products. We will see that this likewise enriches the internal logic to the extent of higher-order predicate logic.

3.2 Martin-Löf Type Theory

3.2.1 Products and Sums

Martin-Löf considered his intuitionistic type theory an open system, where additional types with respective terms and rules could be introduced by need. We outline some of the standard constructions following the general pattern consisting of type formation, term formation, introduction and elimination rules.

Given types A and B we form a new type $A \times B$ called the **product**. The corresponding terms are pairs (s, t) of terms s and t. The eliminator for products is an operation $E_\times$ lifting functions $f : A \to B \to C$ to functions $E_\times\, f : A \times B \to C$. Introduction and elimination are formalised by respective typing rules:

$$\frac{\Gamma \vdash s : A \quad \Gamma \vdash t : B}{\Gamma \vdash (s, t) : A \times B} \qquad \frac{\Gamma \vdash f : A \to B \to C}{\Gamma \vdash E_\times\, f : A \times B \to C}$$

The introduction rule on the left expresses the conditions one would expect, demanding each component of the pair to have the corresponding type. The elimination rule on the right clarifies how the mentioned lifting works. On top of the additional syntax and typing rules, new types may come with reduction rules clarifying the intended behaviour of elimination on normal forms. For products, one adds the rule $E_\times\, f\, (s, t) \succ f\, s\, t$ asserting that elimination just decomposes pairs and feeds the components to the given function. From the general elimination scheme, one can easily reconstruct the more familiar projection functions:

$$\pi_1 \coloneqq E_\times\, (\lambda(x : A)\, (y : B).x) \qquad \pi_2 \coloneqq E_\times\, (\lambda(x : A)\, (y : B).y)$$

By simple type derivations we obtain $\vdash \pi_1 : A \times B \to A$ and $\vdash \pi_2 : A \times B \to B$. Moreover, from the additional reduction rule it follows that $\pi_1\, (s, t) \succ s$ and $\pi_2\, (s, t) \succ s$, so the projections behave as expected.

A second additional type former is the **sum** $A + B$ for types A and B. The terms have either the form $i_1\, s$ for a term $s : A$ or $i_2\, t$ for a term $t : B$. The eliminator E_+ takes two functions $f_1 : A \to C$ and $f_2 : B \to C$ and applies the one matching to the particular member of $A + B$. Formally, the sum type comes with typing rules

$$\frac{\Gamma \vdash s : A}{\Gamma \vdash i_1\, s : A + B} \qquad \frac{\Gamma \vdash t : B}{\Gamma \vdash i_2\, t : A + B} \qquad \frac{\Gamma \vdash f_1 : A \to C \quad \Gamma \vdash f_2 : B \to C}{\Gamma \vdash E_+\, f_1\, f_2 : A + B \to C}$$

as well as the reduction rules $E_+\, f_1\, f_2\, (i_1\, s) \succ f_1\, s$ and $E_+\, f_1\, f_2\, (i_2\, t) \succ f_2\, t$.

Next, we define the **empty type** $\bot$ and the **unit type** $\top$. As it is supposed to have no members, we simply do not provide any introduction rule for $\bot$ but assume an eliminator $E_A : \bot \to A$ for every type A. Contrarily, the unit type introduces a particular term $T : \top$ and comes with no meaningful eliminator.

3.2.2 Dependent Types

So far, we have considered types and terms as completely separate entities. However, it is fruitful to also accommodate types that depend on values, hence called **dependent types**. For instance, once we have defined a type $\mathbb{N}$ of natural numbers in Section 3.2.4, it might be interesting to consider types $\mathbb{N}_n$ for some $n : \mathbb{N}$ which contains all $m < n$. In order to express such types, we at least have to allow types to contain variables. Then for a type B containing a variable x we can define the **dependent product** $\Pi x : A.B$ and the **dependent sum** $\Sigma x : A.B$.

The former describes dependent functions for which the return type might depend on the argument. Hence the introduction and elimination rule for dependent products resemble the rules (v) and (a) for normal function types:

$$\frac{\Gamma, (x : A) \vdash s : B}{\Gamma \vdash (\lambda x : A.s) : (\Pi x : A.B)} \qquad \frac{\Gamma \vdash s : (\Pi x : A.B) \quad \Gamma \vdash t : A}{\Gamma \vdash s\, t : B[t/x]}$$

The latter contains dependent pairs for which the type of the second component might depend on the first component. Hence the typing rules for dependent sums are a generalised version of the rules for normal products:

$$\frac{\Gamma \vdash s : A \quad \Gamma, (x : A) \vdash t : B}{\Gamma \vdash (s, t) : (\Sigma x : A.B)} \qquad \frac{\Gamma \vdash f : (\Pi x : A.B) \to C}{\Gamma \vdash E_\Sigma\, f : (\Sigma x : A.B) \to C}$$

Note that, although there is the obvious similarity to products, the type $\Sigma x : A.B$ is called dependent sum as it can also be seen as a big sum over B indexed by A. For similar reasons, instead the type $\Pi x : A.B$ is called dependent product.

Now that we admit variables in types, it is a natural idea to also allow abstraction on type level. That is, we now also consider terms $\lambda x : A.B$ for types A and B which describe functions taking a term s of a type A and yielding a type $B[s/x]$. In order to assign types to such abstractions, we need to be able to assign a type to the resulting $B[s/x]$. Hence we simply assume a universal type U which we assign to every type we have considered so far. Then the rule for type abstraction is in fact captured by the rule for normal abstractions:

$$\frac{\Gamma, (x : A) \vdash B : U}{\Gamma \vdash \lambda x : A.\, B : A \to U}$$

As types now may contain programs, and additional **conversion rule** has to assert that $\Gamma \vdash s : B$ whenever $\Gamma \vdash s : A$ and A and B have the same normal form. Members s of the type $A \to U$ can be seen as **type families**, as they yield a type $s\, t$ for every $t : A$. Also, we can now simply interpret all types as special terms and use them as arguments for functions and dependent types. For instance, we can express the type constructor for products as a function of type $U \to U \to U$ by $\lambda(A : U)(B : U).A \times B$ and the corresponding term constructor of type $\Pi(A : U)(B : U).A \to B \to A \times B$ by $\lambda(A : U)(B : U)(x : A)(y : B).(x, y)$.

3.2.3 Internal Predicate Logic

At this point we have arrived at a very powerful type system that allows for expressing constructions on various levels in a uniform way. All operations were introduced with clear procedural intuitions and we now turn to exploring their logical interpretation. As mentioned in Section 3.1.4, the basis for using dependent type theory as a foundation of mathematics is to interpret logical propositions as types. We have already discussed that functions are the core of dependent type theory. Now considering U as the type of propositions, functions $P : A \to U$ can be seen as predicates since they return a proposition for every member of A. Then the rest of predicate logic can be interpreted as summarised in Table 3.1.

Table 3.1: Summary of propositions-as-types and proofs-as-programs.

Logical concept	Interpretation	Proof term
Truth	$\top$	the canonical proof T
Falsity	$\bot$	none
Conjunction	$A \times B$	pair (s, t) of respective proofs of A and B
Disjunction	$A + B$	$i_1\, s$ or $i_1\, s$ for a respective proof s of A or B
Implication	$A \to B$	function turning proofs of A to proofs of B
Negation	$A \to \bot$	function turning proofs of A to absurdity
$\forall$-quantification	$\Pi(x : A).P\, x$	function yielding proofs of $P\, s$ for s in A
$\exists$-quantification	$\Sigma(x : A).P\, x$	pair of a witness s in A and a proof of $P\, s$

We have already seen that the interpretation of implication as function types is justified by the similarity of the natural deduction and typing rules, respectively. The same holds for the other logical operations, we do not discuss the correspondence here in detail. Note that the internal logic is higher order in that it can express higher-order functions and predicates and allows quantification over all typeable terms, which includes functions and predicates.

The only symbol of predicate logic we have not interpreted yet is $=$ for equality. As equations are formulas, we also want to accommodate equality as a type. Hence for every type A and terms s and t we define a type $s =_A t$ and with rules:

$$\frac{\Gamma \vdash s : A}{\Gamma \vdash e\, s : s =_A s} \qquad \frac{\Gamma \vdash H : P\, s \quad \Gamma \vdash P : A \to U}{\Gamma \vdash E_{=}\, H : s =_A t \to P\, t}$$

The introduction rule internalises the reduction behaviour of terms in that the canonical equality proof $e\, s$ proves equations $t =_A u$ if the terms t, u, and s all have the same normal form. Reading the elimination rule in the propositional interpretation, it states that if a predicate P holds for a term s, then it also holds for all terms t equal to s. Hence $=_A$ expresses **Leibniz equality** in the sense that there is no property distinguishing equal objects.

3.2.4 Natural Numbers

As we did in our development of set theory, we again choose the natural numbers as a case study. Since we are interested in entities such as the collection $\mathbb{N}$ of numbers itself as well as functions from $\mathbb{N}$ to $\mathbb{N}$, we want to interpret $\mathbb{N}$ as a type. It is an immediate implementation of the inductive conception underlying the Peano axioms to define $\mathbb{N}$ with two term constructors 0 and S:

$$\frac{}{\Gamma \vdash 0 : \mathbb{N}} \qquad\qquad \frac{\Gamma \vdash s : \mathbb{N}}{\Gamma \vdash S\, s : \mathbb{N}}$$

Note that these rules do not assign a type to S itself but we can capture its behaviour by the function $\lambda n : \mathbb{N}.S\, n$ of type $\mathbb{N} \to \mathbb{N}$. Concerning the eliminator for $\mathbb{N}$, we first consider a simplified form

$$\frac{\Gamma \vdash s : A \qquad \Gamma \vdash f : A \to A}{\Gamma \vdash E_{\mathbb{N}}\, s\, f : \mathbb{N} \to A}$$

with reduction rules $E_{\mathbb{N}}\, s\, f\, 0 \succ s$ and $E_{\mathbb{N}}\, s\, f\, (S\, n) \succ f\, (E_{\mathbb{N}}\, s\, f\, n)$. So $E_{\mathbb{N}}\, s\, f$ is a function returning the start value s in input 0 and the step function f applied to the result for n on input $S\, n$. This means that $E_{\mathbb{N}}$ exactly implements primitive recursion, making addition and multiplication immediately definable by setting

$$m + n \coloneqq E_{\mathbb{N}}\, m\, (\lambda n'.S\, n')\, n \qquad \text{and} \qquad m * n \coloneqq E_{\mathbb{N}}\, 0\, (\lambda n'.m + n')\, n$$

where we introduce the convention to leave out clear type annotations and to use the pattern $s\, x \coloneqq t$ as short form of $s \coloneqq \lambda x.t$. The infix notations $m + n$ and $m * n$ hence define functions $+$ and $*$ of type $\mathbb{N} \to \mathbb{N} \to \mathbb{N}$. We can use the eliminator $E_\times$ for products to obtain cartesian versions $E_\times + : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ of addition and $E_\times * : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ of multiplication.

The simplified elimination rule from above employs a constant return type A. This can be generalised to a type family $P : \mathbb{N} \to U$ such that eliminating a value n yields a result of type $P\, n$:

$$\frac{\Gamma \vdash P : \mathbb{N} \to U \qquad \Gamma \vdash s : P\, 0 \qquad \Gamma \vdash f : \Pi n : \mathbb{N}.P\, n \to P\, (S\, n)}{\Gamma \vdash E_{\mathbb{N}}\, s\, f : (\Pi n : \mathbb{N}.P\, n)}$$

Given the extra argument n in f specifying the argument type $P\, n$ and return type $P\, (S\, n)$, the second reduction rule now reads $E_{\mathbb{N}}\, s\, f\, (S\, n) \succ f\, n\, (E_{\mathbb{N}}\, s\, f\, n)$. The simplified elimination is a special case of of the general elimination as we can always set $P \coloneqq \lambda n : \mathbb{N}.A$ to be the type family being constantly A and let the step function ignore its additional argument. Also, we can easily define an expression c performing a case distinction such that $c\, s\, t\, 0 \succ s$ and $c\, s\, t\, (S\, n) \succ t$ by setting $c\, s\, t\, n \coloneqq E_{\mathbb{N}}\, s\, (\lambda n\, x.t)\, n$.

We prove the Peano axioms for our type $\mathbb{N}$ to illustrate that the chosen definition was indeed meaningful. Recall that this means to give types expressing the axioms and to construct terms inhabiting these types.

- **Disjointness.** This axiom states that zero is no successor and a corresponding type is $\Pi(n : \mathbb{N}).\neg 0 = S\,n$. A term of this type is a function taking a number $n : \mathbb{N}$ and a proof of $H : 0 = S\,n$ as arguments and returning a proof of $\bot$. Recall that the eliminator $E_=$ can transform H and a proof of $P\,0$ for some predicate P into a proof of $P\,(S\,n)$. So we just have to define a predicate P that is easily provable for 0, for instance $P\,0 = \top$, but with $P\,(S\,n) = \bot$. This can be done via case distinction by $P\,n \coloneqq c\,\top\,\bot\,n$.
- **Injectivity.** That S defines an injection can be formulated as the type $\Pi mn.S\,m = S\,n \to m = n$. A term of this type is function taking arguments m, n, and a proof that $S\,m = S\,n$, yielding a proof that $m = n$. We define the predecessor function $p \coloneqq E_{\mathbb{N}}\,0\,(\lambda m\,r.\,m)$. By the reduction behaviour of $E_{\mathbb{N}}$ we have $p\,(S\,n') \succ^* n'$ for every n', so in particular we have proofs $p\,(S\,n) = n$ and $p\,(S\,m) = m$. Then applying $E_=$ yields a proof of $m = n$.
- **Induction.** Interpreting the general elimination rule for $E_{\mathbb{N}}$ logically, it exactly expresses natural induction: the term $E_{\mathbb{N}}\,s\,f$ is a proof that for all numbers a predicate P holds whenever s is a proof that P holds for 0 and f is a proof that $P\,n$ implies $P\,(S\,n)$ for all n. So in the propositions-as-types interpretation, induction and recursion are the very same thing. Formally, the induction principle can be expressed by the dependent function type $\Pi(P : \mathbb{N} \to U).P\,0 \to (\Pi n.P\,n \to P\,(S\,n))) \to (\Pi n.P\,n)$, which is inhabited by the term $\lambda(P : \mathbb{N} \to U)(s : P\,0)(f : \Pi n.P\,n \to P\,(S\,n)).\,E_{\mathbb{N}}\,s\,f$.

The final type construction we consider is a type $s \leq t$ for natural numbers:

$$\frac{\Gamma \vdash s : \mathbb{N}}{\Gamma \vdash l_1\,s : s \leq s} \qquad \frac{\Gamma \vdash s : \mathbb{N} \qquad \Gamma \vdash t : \mathbb{N} \qquad \Gamma \vdash H : s \leq t}{\Gamma \vdash l_2\,s\,t\,H : s \leq S\,t}$$

The proof constructor l_1 justifies that $s \leq s$ for every term of type $\mathbb{N}$. The second constructor l_2 allows to increase the term on the right. Elimination is based on

$$\frac{\Gamma \vdash P : \mathbb{N} \to U \qquad \Gamma \vdash s : P\,n \qquad \Gamma \vdash f : \Pi m.n \leq m \to P\,m \to P\,(S\,m)}{\Gamma \vdash E_{\leq}\,n\,P\,s\,f : \Pi m.n \leq m \to P\,m}$$

which, seen as an induction principle, yields a proof of $P\,m$ for all m with $n \leq m$ if there is a proof of $P\,n$ and a transformation of evidence for $P\,m$ to evidence for $P\,(S\,m)$ for every such m with $n \leq m$. We end this case study by establishing that $\leq$ is a well-ordering of $\mathbb{N}$. This time we exemplify the common practice in not giving explicit proposition types and proof terms but providing an informal mathematical justification.

- **Reflexivity.** This is exactly the first introduction rule of $\leq$.
- **Transitivity.** Assume $n \leq m$ and $m \leq k$. We prove $n \leq k$ by induction on the derivation of $m \leq k$. This means, we apply $E_{\leq}$ for $P\,l \coloneqq n \leq l$ and have to prove $P\,m$ and that $P\,l$ implies $P\,(S\,l)$ if $m \leq l$. For the former, $P\,m$ is exactly the assumption $n \leq m$. So now let $m \leq l$ and $P\,l$, so $n \leq l$, we want to prove $P\,(S\,l)$, so $n \leq S\,l$. This follows from the second introduction rule of $\leq$.
- **Antisymmetry.** We show that $n \leq m$ and $m \leq n$ together imply $n = m$ by induction on n and a case distinction for m. This leaves four cases. In the first case, all numbers are 0 and the claim follows trivially. The next two cases contain absurd assumptions of the form $S\,n \leq 0$. In the last case we assume $S\,n \leq S\,m$ and $S\,m \leq S\,n$ and want to prove $S\,n = S\,m$. The inductive assumption reads $n = m$ if $n \leq m$ and $m \leq n$, where the conditions follow from the respective assumptions for the successors by case distinction and transitivity. Hence $n = m$ and thus $S\,n = S\,m$.
- **Linearity.** We show that either $n \leq m$ or $m \leq n$ by the same induction and case distinction that we applied for antisymmetry. This time the three cases where one of the number is 0 are trivial since we can always prove $0 \leq n$. In the final case we know that either $n \leq m$ or $m \leq n$ by induction and want to derive a decision for the successors. This decision is straight-forward as $n \leq m$ implies $S\,n \leq S\,m$ by a simple induction on $n \leq m$.
- **Foundation.** We define the strict ordering $n < m \coloneqq S\,n \leq m$. A way to express that $\leq$ and $<$ are well-founded is to demand that every downward chain like $\cdots < n < m$ eventually stops. That a number n does not admit an infinite downward chain can be expressed as a type $W\,n$:

  $$\frac{\Gamma \vdash n : \mathbb{N} \qquad \Gamma \vdash H : \Pi m.m < n \to W\,m}{\Gamma \vdash w\,H : W\,n}$$

  This expresses the insight that n does not admit an infinite chain if no $m < n$ does. A proof that $<$ is well-founded hence amounts to a proof that $W\,n$ for all n. We apply natural induction, there are two cases. Clearly $W\,0$ since there simply is no $n < 0$. So now we assume $W\,n$ and want to prove $W\,(S\,n)$, hence we let $m < S\,n$ and show $W\,m$. Case distinction on $m < S\,n$ admits two cases. In the former, we have $m = n$ and the claim follows by assumption. In the latter, we have $m < n$ and obtain $W\,m$ by applying $W\,n$.

To provide a demo of mechanised proving, we have implemented the constructions and proofs of this section in a commented Coq script available online.[1]

[1] `https://www.ps.uni-saarland.de/~kirst/hok/numbers.html`

Chapter 4
Aspects of a Comparison

There is no real need to argue in depth that both axiomatic set theory and dependent type theory are formal foundations of mathematics as characterised in Section 1.1. As shown in the two previous chapters, both systems offer an equally universal and precise language for mathematical statements and their respective notion of deduction is effectively verifiable following the given inference rules. Concerning the aspects of practicality given in Section 1.2, however, it turns out that both approaches can be judged differently. Although similarly accessible, depent type theory seems more suitable for implementation on computers whereas axiomatic set theory is currently accepted by the larger community. After adressing these aspects in Section 4.1, we contrast the philosophical background of the classical first-order formulation of axiomatic set theory with the intuitionistic higher-order logic internal to dependent type theory in Section 4.2. Finally, in Section 4.3, we show how one system can be modelled by the other, giving a partial answer to the consistency question.

4.1 Practicality

4.1.1 Accessibility

In the two previous chapters we have introduced the basic ideas of axiomatic set theory and dependent type theory on roughly ten pages each. In our intention, these should provide enough information for a formally experienced reader to be able to consult the literature we referred to and work out the remaining details on their own. Hence we suppose that both systems are in principle similarly well-suited for teaching and that there are no major obstacles making one of the systems considerably hard to access.

Moreover, working on a reasonably detailed level, both languages do not differ very much from an informal approach to mathematics. As no one would ever give a full natural deduction tree or proof term for a realistic theorem by hand, one anyway resorts to a less formal language where the outlined reasoning appears almost identical to common practice. The only importance is that it is possible to reconstruct a fully formal proof if need be, for instance to dispel any doubts. Concerning the statements themselves, the type-theoretic language with judgements $n : \mathbb{N}$ and type constructors $\times$ or Π might be less familiar to the working mathematician but of course one can always employ a more common notation using $n \in \mathbb{N}$, $\wedge$, and $\forall$.

One frequent objection to dependent type-theory is its apparent relative complexity in comparison to an axiomatisation of set theory. Not only is there a lot of syntax to digest but also the amount of typing rules may seem discouraging at first glance. In contrast, one could say, ZF set theory only comes with nine axioms and no syntax but a symbol for membership. It is clear that this argument can mostly be dismantled since a formal development of ZF presupposes the whole machinery of first-order logic including the syntax and semantics of formulas as well as the rules of a deduction system. Quite the contrary, the additional syntax type-theory provides to include proof terms has a major advantage. Proof terms are a very compact and natural representation of derivations, allowing for a precise notion of proof simplification via β-reduction and a correspondence of strong normalisation with consistency.

The remaining technical overhead produced by a fully-fledged type theory with conversion and universes is compensated by the very concept of types, capturing semantic intuition and ruling out meaningless statements. For a novice to axiomatic set theory it may come as a surprise that the only objects considered are sets and other concepts like functions and numbers are encoded as sets. From an intuitive perspective, one is more likely to think that functions and numbers are distinct entities and that it is meaningless to apply a number to itself or a function on numbers to some other function. Whereas all these terms are perfectly well-formed in an axiomatic set theory, they are excluded by dependent type theory as they cannot be assigned meaningful types.

The apparent complexity of dependent type theory also comes with the benefit that many mathematical concepts are built-in, causing no additional need for indiscriminate encodings. This includes the notions of ordered pairs, functions, and predicates, which are all primitive to dependent type theory. Moreover, as seen in the type-theoretic treatment of Peano arithmetic (Section 3.2.4), the intuitive conception of inductively generated numbers is captured perfectly by the introduction rules of the type $\mathbb{N}$ and the recursive definitions of addition and multiplication are immediately expressed by the internal computational system. Axiomatic set theory lacks a direct treatment of these elementary concepts.

4.1.2 Mechanisability

Dependent type theory is well-suited for direct implementation on a computer. Similar to other functional programming languages, such an implementation mainly consists of algorithms for type checking and the computation of normal forms. The features of mature proof assistants based on dependent type theory may include a helpful tactic language, which allows for interactively constructing proof terms following informal reasoning, and code extraction, translating internally verified programs into other executable languages. Proof assistants based on axiomatic set theory are harder to implement and less suited for program verification.

Among the currently most prominent proof assistants, only **Mizar** [NK09] is based on axiomatic set theory. Developed by Trybulec in 1973, to date the system provides the largest library of formalised mathematics. Statements and proofs can be written in a pseudo-natural language, making Mizar accessible for a broad community. Interestingly, the language allows to define a weak type structure such that some of the advantages of a typed language are accommodated.

On the other side, there are many proof assistants based on various dialects of type theory, each with their own active community and field of applications. To only mention a few representatives, there is **Agda** [Nor08], which is a close implementation of Martin-Löf type theory as outlined in Section 3.2, **Coq** [Coq18], which is based on a close derivative called calculus of constructions proposed by Coquand and Huet [CH88], and **Isabelle/HOL** [NWP02], which implements a version of classical higher-order logic without dependent types.

Implementations of type theory already play a key role in establishing new mathematical results. As an example, in the mid 19th century it was conjectured that any (possibly fictional) map could be coloured using only four colours such that no two adjacent countries have same colour. The **four colour theorem** remained without accepted proof until 1976, when Appel and Haken used computers to perform the necessary large case distinctions [AH76]. Since the employed software was not formally verified, the status of the theorem was left uncertain until Gonthier published a fully formalised Coq proof in 2006 [Gon08].

The **Kepler conjecture**, stating that the densest possible way to pack oranges into a box is the so-called cubic close packing, presents a similar example. Suggested by Kepler in the early 17th century, the conjecture was only proved by Hales in 2005 [Hal05]. Again, this first proof employed untrusted software and only after a version formalised in HOL Light and Isabelle had been given in 2017 [HAB+17], the theorem was accepted without doubt.

There are many more examples of extensive formalisation projects (e.g. [GAA+13] and [Ler09]) and it is to assume that computer-assisted formal verification will play an increasingly important role in mathematical research. Type theory seems to provide a reasonable basis for this purpose. To emphasise this point, Sections 3.2.4 and 4.3.2 are accompanied by formal Coq proof scripts.

4.1.3 Community

To date, ZF set theory is the widely accepted and fully canonised standard system for foundations of mathematics. Most mathematical results in research articles, textbooks, as well as lectures are presented in a set-theoretic language and it is generally agreed on a concrete formulation of the underlying logical system and axioms. Moreover, having been developed for over one hundred years by now, the theory is well-understood and there are only few discussed extensions such as the axiom of choice playing a major role for the working mathematician. Other arguable additions such as the continuum hypothesis or large cardinal assumptions only manifest in rather specific situations. Thus, axiomatic set theory clearly enjoys the general prevalence one expects of a foundational system to be reasonable in practice.

Dependent type theory, on the other side, only plays a marginal role outside of a few areas within mathematics and theoretical computer science. For one reason, dependent type theory in its modern formulation is considerably younger than axiomatic set theory and hence now faces a more mature competitor. Moreover, as yet there is no standard type system similar to ZF set theory everyone agrees on, rather there are several equally investigated alternatives. Although most of them share the common structure embodied in Martin-Löf type theory, some of these alternatives differ in fundamental aspects of their internal logic and younger disciples such as homotopy type theory [Uni13] still involve some open questions.

Of course, such an evaluation is always a current snapshot that might be subject to change. For instance, as it will be discussed in the next section, dependent type theory naturally implements an intuitionistic logic which is a good fit for questions comprising algorithmic aspects. So although dependent type theory might perhaps not supersede axiomatic set theory as a general purpose foundation of mathematics, it probably will become more prominent in some areas native to theoretical computer science.

4.2 Philosophical Background

4.2.1 Conceptual Origin

The historical path leading to axiomatic set theory is roughly based on a viewpoint dubbed **platonism**. In this view, mathematical objects are considered to exist as abstract entities independent of the human mind. Concretely, objects such as specific numbers or geometrical shapes, that appear in numerous instantiations within the physical world, belong to a single abstract essence inhabiting a non-physical universe of conceptions. From this perspective, the underlying logical system and axioms of set theory are attempts to formalise the self-evident knowledge about this conceptual universe we can gain from studying its inklings in the physical world. Taking this idea literal, the platonic reading of the logical quantifiers really refers to existence and universality of corresponding abstract ideas, and proving mathematical theorems means to explore their eternal truths.

The explorative flavour underlying set theory shows in the way how open questions are faced. For instance, as established by results of Gödel and Cohen, the continuum hypothesis (CH) determining the cardinality of the reals can neither be proved nor disproved from the ZF axioms. However, a prominent impression is that CH in fact *has* a determined status in the conceptual universe which is yet to discover by finding the right extensions of the axiomatisation. This singles out the intended structure described by ZF as the *real* mathematical universe from the interpretations of alternative but still formally consistent axiomatisations. Facing the growing variety of alternatives, modern ideas such as the set-theoretic multiverse proposed by Hamkins [Ham12] and the search for maximality principles outlined by Incurvati [Inc17] express weaker forms of platonism.

Dependent type theory, on the other hand, has its roots in **constructivism**. Instead of resorting to a universe of abstract ideas, the constructivist or intuitionistic view supposes the mathematical objects to only exist as concepts in the human mind. For instance, encountering various triangular shapes in the physical world gives rise to the abstract concept of a triangle that we can grasp and communicate. Similarly, we can abstract out the amount of any collections of objects and moreover, since we can imagine empty collections and the procedure of extending collections by single objects, we can develop and share an intuitive understanding of the natural numbers.

As a convinced constructivist, Martin-Löf developed his formulations of dependent type theory following the paradigm that all objects either are canonical terms only built from primitive constructors, or procedures that compute such normal forms. That is, terms that have the type of natural numbers are either iterated applications of the successor function starting on the canonical zero term or concrete procedures that eventually compute canonical numbers. Consequently, existence means constructibility and as there is no presupposed universe of mathematical truth, truth simply means provability by the logical principles wired in the brain.

4.2.2 Classical vs. Intuitionistic Logic

The different philosophical origins of axiomatic set theory and dependent type theory have a governing influence on the respectively accepted logical principles. Most importantly, the first-order predicate logic at the base of axiomatic set theory is classical in assuming the law of **excluded middle** (XM), stating that for every formula ϕ the disjunction $\phi \vee \neg\phi$ holds. This is equivalent to the double negation principle, stating that from $\neg\neg\phi$ one can always derive ϕ, and has no constructive interpretation. This and related differences of the two foundational systems as well as their consequences are outlined in this section.

Classical reasoning appears in many ways during a traditional development of axiomatic set theory and stems from the belief that every logical statement has a definite truth value in the platonic universe. Several elementary set-theoretic statements rely on, or are in fact equivalent to XM, for instance that subsets of finite sets are finite, that non-empty sets are inhabited, and the like. Moreover, common axiomatic additions like the axiom of choice (AC) or a generalised form of the continuum hypothesis (GCH), determining the cardinality of the power sets of arbitrary sets, both imply XM. Concerning the former, assuming the axiom of choice yields a general way to choose elements from arbitrary families of non-empty sets and is, among others, equivalent to the well-ordering theorem and Zorn's lemma. As shown by Diaconescu [Dia75], AC implies XM and hence can only be assumed in a classical setting. Concerning the latter, that GCH in turn implies AC was shown by Sierpiński [Sie47]. Another feature of axiomatic set theory is that it admits **impredicative** definitions, where an object may be constructed using a seemingly circular definition that may refer to the very object under consideration. For instance, when proving the Knaster-Tarski fixed-point theorem for monotone functions, one obtains the least fixed point as the intersection of all pre-fixed points, which includes this intersection itself.

Not only is it the case that many mathematical results rely on classical reasoning, also the induced technique of proof by contradiction potentially allows to shortcut elaborate constructive proofs. For instance, consider existential formulas of the form $\exists x.\, \phi(x)$. It may well be that for concrete properties ϕ there is an explicit construction of a witnessing object. However, in a classical setting, such statements can be proved indirectly by showing that assuming $\neg\exists x.\, \phi(x)$ leads to a contradiction and hence $\exists x.\, \phi(x)$ must hold as the only remaining option. A similar argument is possible to prove disjunctions $\phi \vee \psi$ without having to provide a concrete decision for either ϕ or ψ. So in particular, classical reasoning establishes De Morgan's laws which judge $\exists x.\, \phi(x)$ and $\neg\forall x.\, \neg\phi(x)$ as well as $\phi \vee \psi$ and $\neg\phi \wedge \neg\psi$ to be logically equivalent, respectively. Note by the way that a proof by contradiction must not be confused with a proof of negation. A formula $\neg\phi$ may be proved by assuming ϕ and then deriving absurdity, which is totally fine with the constructive interpretation of negation. In general, whether or not a short classical proof or an explicit constructive proof is to be preferred mainly depends on the personal taste and philosophical stance one is willing to take.

It is clear that a committed constructivist, on the other hand, has to reject XM. In the interpretation of truth as provability witnessed by mental constructions, there is no place for underspecified objects as derived from indirect proofs. Consequently, neither AC nor GCH can be assumed in a meaningful constructive mindset. Note that this also gives rise to various degrees of weak constructivism, for instance one could still soundly accept XM but refrain from accepting AC or GCH. Intuitionistic logic allows for developing the main body of standard mathematical results as illustrated by Bishop [BB85] and others. Such a project requires some effort in finding constructive proofs where possible or to reformulate definitions and statement in a constructively suitable and classically equivalent way. Contrarily, the stronger position of finitism even restricts the logical language to consider finite objects only, a view severely shrinking the fragment of representable mathematics. Also not shared by all constructivists is the objection against impredicative definitions based on the paradigm of iterated constructions, stating that every object may only consist of already intuited parts.

Dependent type theory naturally internalises intuitionistic logic based on the proofs-as-programs interpretation. As we have outlined in Section 3.2, disjunctions and existential quantification are interpreted as sum types and dependent sum types, respectively. The only canonical inhabitants provided by the typing rules for sum types $A + B$ are the injections $i_1\, s$ and $i_2\, t$ for $s : A$ and $t : B$, so a type-theoretic proof of a disjunction in fact carries the information which alternative was established. Similarly, a canonical inhabitant of a dependent sum $\Sigma x : A.\, P\, x$ is a dependent pair (s, t) where $s : A$ is a concrete witness and $t : P\, s$ its corresponding correctness proof. Although by design interpreting intuitionist logic, many dialects of dependent type theory are designed such that they are still consistent with classical assumptions. For instance, in our presentation of Martin-Löf type theory, it is consistent to assume a term c of type $\Pi A : U.\, A + \neg A$, hence recovering the availability of proof by classical case distinction. Moreover, appropriate type-theoretic versions of AC and GCH can be assumed and there is also a way to carefully admit impredicative definitions.

The axiomatic freedom of constructive logic is one of its key advantages. Classical principles can be added flexibly by need and the constructive core allows for subclassical logical analyses. This means for instance that only in a logical system without a hard-wired choice principle it is possible to study statements that are equivalent to choice. As soon as one works in a theory with AC, there is no meaningful information in a proof that another statements implies AC. Analogously, classical logic admits no meaningful discussion of the equivalents to XM or weaker assumptions possibly sufficient for a given purpose. As another advantage, constructive proofs bear algorithmic content, as made explicit in a dependent type theory (cf. [Con02]). In particular, a constructive proof of a formula $\forall x.\, \exists y.\, \phi(x, y)$ yields a procedure that computes a corresponding object y for every input x. This means that intuitionistic logic is a perfect match for computability theory, since the fact that every definable function is computable erases the cumbersome need for an explicit model of computation.

4.2.3 First-Order vs. Higher-Order Logic

Orthogonal to the acceptance of non-constructive logical principles is the admitted order of objects and quantification within the logical system. Axiomatic set theory is usually formalised in first-order predicate logic with a single binary relation symbol for membership and quantification restricted to individuals. Thereby, set-theoretic operations of higher-order character like separation referring to predicates and replacement referring to functions must be presented in a somewhat artificial way using axiomatic schemes. In sharp contrast, dependent type theory naturally internalises a higher-order logic where functions may uniformly act on and return functions, and where quantification is in principle allowed on all types. We sketch in this section how the difference in logical order results in a completely distinct metatheory, which is subject to quite a controversial debate.

As it was outlined in Section 2.1, first-order logic can be equipped with a sound and complete deduction system. This means that a formula is semantically valid if and only if it can be syntacticly derived using the deduction rules. An immediate consequence is **compactness**, stating that a set of formulas has a model if and only if all its finite subsets have a model. Moreover, it follows that a set of formulas is free of derivable contradictions if and only if it has a model. These properties establish a well-behaved and in some sense desirable connection of the semantics and syntactic provability of first-order formulas. In particular, systematically enumerating all possible deductions eventually captures every semantic truth.

On the other hand, some results show a notorious weakness of first-order logic. Most notably, the Löwenheim-Skolem theorem implies that theories with some infinite model also must have models in every other infinite cardinality. Hence first-order logic is not sufficient to characterise a single intended model, a property which would be called **categoricity**, but instead always admits unintended non-standard models. This indeterminacy is further fuelled by Gödel's incompleteness theorem, stating that every formal and effective system of reasonable strength contains statements that are neither provable nor rejectable. As a consequence, the *completeness* of the deduction system applied to one of these undecided statements produces two models where one satisfies the statement and the other its negation. This procedure can be iterated and hence yields arbitrarily incompatible models.

In contrast, the additional expressive strength of higher-order logic allows for categorical axiomatisations. For instance, axiomatising the natural numbers with their usual operations in higher-order logic, a system called second-order arithmetic, singles out the intended model of the usual numbers. Hence the unwelcome distinction of standard and non-standard models of arithmetic completely vanishes when using a logic of at least second order. Similarly, set theory itself can be naturally axiomatised in a higher-order logic by replacing the axiomatic schemes by their universal closure. Concretely, in this system called second-order ZF there is a single separation axiom of the form

$$\forall x.\, \forall P.\, \exists y.\, \forall z.\, z \in y \leftrightarrow z \in x \wedge P(z)$$

stating that every set x and predicate P produces a subset $y = \{\, z \in x \mid P(z) \,\}$. That an axiomatisation in this style only admits a very regular form of models was already observed by Zermelo [Zer30], who decidedly argued in favour of the higher-order axiomatisation against the opposition led by Skolem, Gödel, and Quine (cf. [Sko22]). Zermelo's quasi-categoricity result inspired a line of recent work by Uzquiano [Uzq99], McGee [McG97], and Kirst and Smolka [KS17].

However, the strength of higher-order logic comes at the loss of deductive completeness. As the incompleteness theorem applies to second-order arithmetic, for every effective deduction system for higher-order logic there must be a deductively undecided arithmetic statement. But there is only a single model of second-order arithmetic, so (thinking classically) either the statement itself or its negation is semantically valid. Thus in either case the deduction system cannot prove a valid formula and is hence necessarily incomplete. Informally this can be understood as a transition of the incompleteness of first-order theories to the incompleteness of the deduction systems for higher-order logic.

There has been some work to establish first-order-like semantics for higher-order logic, most notably by Henkin [Hen50]. In his approach, higher-order variables for predicates and functions not necessarily range over the whole collection of possible relations and functions within the semantic domain. That is, assuming an interpretation $M = (D, A)$, the first-order variables still range over the whole of D but a variable P for a unary predicate need not range over all of $\mathcal{P}(D)$ and a variable f for a unary function need not range over all of $D \to D$. Then restricting the higher-order quantification in certain ways can maintain completeness and compactness.

Conversely, axiomatised set theory with its internal notion of predicates and functions renders higher-order logic to some extent. For instance, one can of course quantify over all predicates on the natural numbers ω, since those are elements of $\mathcal{P}(\omega)$. Only when talking about concepts on proper classes, the internal representation fails and a uniform higher-order treatment gets desirable. This is to a certain degree implemented in class-set theories such as NBG [vN25], providing at least some support for reasoning on class level.

Dependent type theory offers an elegant treatment of higher-order logic, where categoricity is attainable and the problem of incompleteness is lessened by the constructive philosophy. In this mindset, there are deductively undecidable statements everywhere, most obviously XM itself, and focussing on provability anyway reduces the importance of semantic validity. Following exemplary objections voiced by Shapiro [Sha91] and Väänänen [Vä01], it hence may seem puzzling how much higher-order logic has been fought against and that a foundational system such as ZF based on a language with unwelcome logical defects could become mainstream.

4.3 Relative Consistency

4.3.1 Types as Sets

We now describe a simple set-theoretic semantics for dependent type theory. This semantics is given by a denotation function mapping types A to sets $[\![A]\!]$ and well-typed terms $\Gamma \vdash s : A$ to elements $[\![\Gamma \vdash s : A]\!] \in [\![A]\!]$. By means of such a denotational semantics we obtain an interpretation of the type-theoretic language within set theory and, most importantly, deduce the logical consistency of the type system. In the following informal presentation we outline the standard ideas given by Coquand [Coq87], Dybjer [Dyb91], and Werner [Wer97].

In principle, every type-theoretic concept is interpreted as its set-theoretic counterpart. Hence we naturally begin by setting $[\![\bot]\!] \coloneqq \emptyset$, so we interpret the empty type as the empty set. Similarly, we set $[\![\top]\!] \coloneqq \{\emptyset\}$, interpreting the unit type as the concrete singleton $\{\emptyset\}$. Concerning the canonical member T of the unit type we then set $[\![\Gamma \vdash T : \top]\!] \coloneqq \emptyset$, yielding $[\![\Gamma \vdash T : \top]\!] \in [\![\top]\!]$ as wished.

Similarly simple is the interpretation of binary products on types by cartesian product on sets, formally $[\![A \times B]\!] \coloneqq [\![A]\!] \times [\![B]\!]$. Then the primitive pairs inhabiting $A \times B$ are mapped to the encoded ordered pairs which are elements of $[\![A]\!] \times [\![B]\!]$:

$$[\![\Gamma \vdash (s,t) : A \times B]\!] \coloneqq ([\![\Gamma \vdash s : A]\!], [\![\Gamma \vdash t : B]\!])$$

Here, we first analyse the typing of (s,t) to obtain typings of the components s and t, respectively. Then we construct the ordered pair of the two respective denotations. The interpretation of binary sums $A + B$ is dually given by disjoint union of sets and not discussed here in detail.

The following extends the denotation to functions, application and abstraction:

$$\begin{aligned} [\![A \to B]\!] &\coloneqq [\![A]\!] \to [\![B]\!] \\ [\![\Gamma \vdash s\,t : B]\!] &\coloneqq [\![\Gamma \vdash s : A \to B]\!]([\![\Gamma \vdash t : A]\!]) \\ [\![\Gamma \vdash (\lambda x : A.s) : A \to B]\!] &\coloneqq \{\, (a, [\![\Gamma, (x : A) \vdash s : B]\!]^a_x) \mid a \in [\![A]\!] \,\} \end{aligned}$$

So function types $A \to B$ are interpreted as the set of functions from $[\![A]\!]$ to $[\![B]\!]$. Then analysing an application $\Gamma \vdash s\,t : B$, we know that s has type $A \to B$ for some A with member t. Hence the interpretation $[\![\Gamma \vdash s : A \to B]\!]$ of s is a function in $[\![A]\!] \to [\![B]\!]$ we can apply to the interpretation $[\![\Gamma \vdash t : A]\!]$ of t. Finally, a lambda abstraction $\lambda x : A.s$ is translated point-wise into a set of pairs with first component a in $[\![A]\!]$ and the image $[\![\Gamma, (x : A) \vdash s : B]\!]^a_x$ as second component. The notation $[\![s]\!]^a_x$ is the interpretation of s with x interpreted as a and can be formalised by carrying an environment for variables throughout the definition of the denotation function.

Now that we have studied the semantics of functions, we can consider the eliminators for empty type and product that were neglected before. Recall that the eliminator E_A for $\bot$ has type $\bot \to A$. Hence it must be interpreted as a function $\emptyset \to [\![A]\!]$. There is only a single such function, namely the empty function, so we set $[\![\Gamma \vdash E_A : \bot \to A]\!] \coloneqq \emptyset$. Next, recall that the eliminator $E_\times$ for binary products turns a function $f : A \to B \to C$ into a function of type $A \times B \to C$. We define this function point-wise by

$$[\![\Gamma \vdash E_\times f : A \times B \to C]\!]\, p \coloneqq [\![\Gamma \vdash f : A \to B \to C]\!](\pi_1(p))(\pi_2(p))$$

where p is an ordered pair in $[\![A]\!] \times [\![B]\!]$ and so the interpretation of f can be applied to the components $\pi_1(p)$ and $\pi_2(p)$.

Moving to dependent products and sums, they are interpreted as indexed cartesian product and indexed disjoint union, respectively. Only discussing the product case at this point, we formally set $[\![\Pi x : A.B]\!] \coloneqq \Pi_{a \in [\![A]\!]} [\![B]\!]^a_x$ with the product of all $[\![B]\!]^a_x$ indexed by $[\![A]\!]$ on the right-hand side. The elements of this indexed product are exactly functions taking an element $a \in [\![A]\!]$ and returning an element of $[\![B]\!]^a_x$, so a typing $\Gamma \vdash (\lambda x : A.s) : (\Pi x : A.B)$ is correctly translated into an element of $[\![\Pi x : A.B]\!]$.

So far we could work out the whole interpretation in ZF set theory without additional assumptions. Now considering the type universe U which is closed under all type constructions, its counterpart mus be a set $[\![U]\!]$ which is analogously closed under the set operations. Such a set is called **Grothendieck universe** [Wil69] and does not necessarily exist in ZF set theory. However, it is consistent to add an axiom asserting a Grothendieck universe and hence completing the interpretation.

Theorem (Soundness)**.** *Assuming ZF and the existence of a Grothendieck universe, the denotation function $[\![\_]\!]$ is total on both types and well-typed terms. Moreover, for every well-typed term $\Gamma \vdash s : A$ it holds that $[\![\Gamma \vdash s : A]\!] \in [\![A]\!]$.*

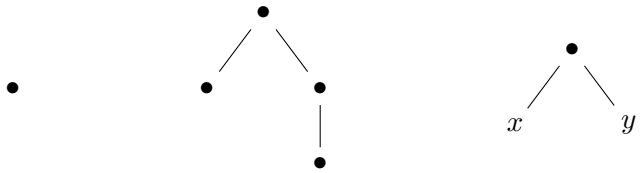
Note that, in order to maintain the soundness property, every extension of the type system needs to be accompanied by a sound interpretation. For instance, if we add a type $\mathbb{N}$ of natural numbers as in Section 3.2.4, we have to add the canonical interpretation of the type $\mathbb{N}$ as the set ω, the term 0 as $\emptyset$ and the type function $S : \mathbb{N} \to \mathbb{N}$ as the set function $\sigma \in \omega \to \omega$. In fact, a very general scheme for inductively defined types has an intuitive set-theoretic semantics [Dyb91].

Employing the soundness theorem, we immediately deduce that there is no well-typed term $s : \bot$ as this would yield an element $[\![\vdash s : \bot]\!]$ of $[\![\bot]\!] = \emptyset$. Hence we can conclude that the logical interpretation of the type system is free of contradictions.

Corollary (Consistency)**.** *Assuming ZF and the existence of a Grothendieck universe, Martin-Löf type theory with a single type universe is consistent. That is, there is no proof of absurdity, or equivalently, the empty type $\bot$ is not inhabited.*

4.3.2 Sets as Trees

A way to conversely interpret set theory in dependent type theory was given by Aczel [Acz78] and elaborated by Werner [Wer97], Barras [Bar10], and Kirst and Smolka [KS18]. In this interpretation, a set is seen as the well-founded tree depicting its membership structure in the sense of the following simple examples:



The trivial tree on the left depicts $\emptyset$ as it indicates no elements. The tree in the middle has one trivial child and one child with trivial child, hence denoting the set $\{\emptyset, \{\emptyset\}\}$. The tree on the right shows the general way to interpret an unordered pair $\{x, y\}$ as a binary tree with children for each component.

The type $\mathcal{T}$ of well-founded trees can be expressed in dependent type theory by

$$\frac{\Gamma \vdash A : U \quad \Gamma \vdash f : A \to \mathcal{T}}{\Gamma \vdash \tau\, A\, f : \mathcal{T}} \qquad \frac{\Gamma \vdash F : \Pi A f.\, (\Pi a.\, P\,(f\,a)) \to P\,(\tau\, A\, f)}{\Gamma \vdash E_{\mathcal{T}}\, F : (\Pi s : \mathcal{T}.P\, s)}$$

with a constructor τ and an eliminator $E_{\mathcal{T}}$ implementing structural recursion. The term $\tau\, A\, f$ denotes the tree with subtrees $f\, a$ for all $a : A$ as depicted by

$$\tau\, A\, f = \quad \begin{array}{ccccc} & & \bullet & & \\ & \swarrow & \downarrow & \searrow & \\ f\,a & & f\,a' & \cdots & f\,a'' \end{array}$$

which suggests to define membership $s \in (\tau\, A\, f)$ to mean that there is $a : A$ with $s = f\, a$. However, as there are structurally equivalent but syntacticly different trees, we actually first have to introduce a corresponding equivalence relation $s \equiv t$ on trees and then set $s \in (\tau\, A\, f) \coloneqq \Sigma a : A.s \equiv f\, a$ as well as $s \subseteq t \coloneqq \Pi u : \mathcal{T}.\, u \in s \to u \in t$. In fact, we directly interpret the set-theoretic equality as tree equivalence and prove that $\mathcal{T}$ satisfies type-theoretic formulations of (some of) the ZF axioms. For instance, by simple inductions it follows that $s \equiv t$ if and only if $s \subseteq t$ and $t \subseteq s$, establishing the extensionality axiom.

Now that we found a way to express membership and equality, we can start interpreting the set operations by employing their type-theoretic counterparts. To represent the empty set, we simply define the trivial tree $\emptyset \coloneqq \tau \perp E_{\mathcal{T}}$. Note that the eliminator $E_{\mathcal{T}}$ of $\perp$ indeed has the expected type $\perp \to \mathcal{T}$ and that it is straightforward to prove that $s \notin \emptyset$ for all trees s. Pairing is defined by $\{s, t\} \coloneqq \tau\, \mathbb{N}\, (c\, s\, t)$ where $c\, s\, t : \mathbb{N} \to \mathcal{T}$ performs the case distinction as defined in Section 3.2.4. It then follows that $u \in \{s, t\}$ if and only if $u \equiv s$ or $u \equiv t$, as required.

Employing the projections $p_1\,(\tau\,A\,f) \coloneqq A$ and $p_2\,(\tau\,A\,f) \coloneqq f$ we further define

$$\bigcup(\tau\,A\,f) \coloneqq \tau\,(\Sigma\,a.\,p_1\,(f\,a))\,(\lambda(a,b).\,p_2\,(f\,a)\,b)$$

and prove that $u \in \bigcup s$ if and only if there is t with $u \in t \in s$, justifying the union axiom. Now we can express the set-theoretic successor $\sigma\,s \coloneqq s \cup \{s\}$ and immediately obtain the natural numbers by $\omega \coloneqq \tau\,\mathbb{N}\,(\lambda n : \mathbb{N}.\,\sigma^n\,\emptyset)$, where $\sigma^n\,\emptyset$ denotes the iteration $E_{\mathbb{N}}\,\emptyset\,\sigma\,n$. This in particular proves the infinity axiom.

Concerning the axiom schemes for separation and replacement, we first have to clarify how to interpret the formulas $\phi(x)$ in $\{\,y \in x \mid \phi(y)\,\}$ and $\phi(y,z)$ in $\{\,z \mid \exists y \in x.\,\phi(y,z)\,\}$. As the former express properties, it is natural to use predicates $P : \mathcal{T} \to U$ and define sets $\{\,t \in \tau\,A\,f \mid P\,t\,\} \coloneqq \tau\,(\Sigma\,a.\,(f\,a) \in P)\,(f \circ \pi_1)$, where $(f \circ \pi_1)$ denotes the function applying f after the projection π_1. The latter formulas expressing functional relations can be approximated by functions $F : \mathcal{T} \to \mathcal{T}$ and considering sets $\{\,F\,t \mid t \in \tau\,A\,f\,\} \coloneqq \tau\,A\,(F \circ f)$. Both operations satisfy membership laws corresponding to higher-order versions of the respective axiom schemes of ZF. However, the full replacement axiom relying on a functional relation rather than a function does not have a direct interpretation in dependent type theory. Moreover and for related reasons, $\mathcal{T}$ satisfies only a weak form of the power set axiom, resulting in the fact that $\mathcal{T}$ interprets the constructive set theory CZF [Acz78].

Theorem. *(Interpretation) $\mathcal{T}$ satisfies the axioms of the constructive set theory CZF.*

This result implies the relative consistency of CZF in the following sense. Suppose there is a formula ϕ such that both ϕ and $\neg\phi$ are provable from the axioms of CZF. This formula has a type-theoretic counterpart A and as the axioms of CZF hold for $\mathcal{T}$, the respective proofs translate into inhabitants of A and $A \to \bot$. Hence there is an inhabitant of $\bot$, which means that such ϕ cannot exist.

We remark that extended type theories such as the calculus of constructions [CH88] with its impredicative universe of propositions and full hierarchy of predicative universes interpret more expressive versions of set theory. Werner [Wer97] has shown that the calculus of constructions with a choice assumption proves ZF (with the axiom of choice) consistent, and that there is a general correspondence of the predicative type universes to set-theoretic Grothendieck universes. This connection was also elaborated by Aczel [Acz98] and partially formalised by Kirst and Smolka [KS18]. More details and explanations concerning the constructions and proofs in this section are given in a corresponding Coq script.[2]

[2] `https://www.ps.uni-saarland.de/~kirst/hok/setsastrees.html`

Chapter 5
Conclusion

This thesis attempts to provide an initial account of the foundational discussion in mathematics. Given the restrictions of a bachelor's project, such an attempt necessarily remains incomplete, as it is noticeable throughout this text. For instance, the explanation of the two systems under consideration in Chapters 2 and 3 avoided some technical details and elementary type-theoretic concepts like general inductive types, the hierarchy of universes and impredicativity had to be left out. Furthermore, the philosophical discussion in Chapter 4 was simplified in sharpening the platonic conception of axiomatic set theory in contrast to the constructive viewpoint leading to dependent type theory. In reality, there are a lot more subtle positions and we only focused on the respective mainstream interpretations. Finally, this thesis is not comprehensive in only examining two particular foundational systems. Concerning set theory, this is fair enough since ZF is widely accepted as standard, although there are the mentioned alternatives such as constructive set theory CZF, class-set theory NBG, and their higher-order versions. Given that dependent type theory is relatively young, there is no agreement on a standard formulation yet. Already when opting for Martin-Löf type theory, one has to make decisions concerning the structure of type universes, the notion of conversion, and the concrete presentation of typing judgements, to only mention a few. Moreover, there is still active research on derivatives as witnessed by the work on the calculus of constructions and the recent rise of homotopy type theory [Uni13]. Completely neglected in this thesis is the foundational approach via category theory (cf. Section 1.8 in [Awo10]), yielding a third option next to axiomatic set theory and dependent type theory.

For the two particular systems studied in this thesis, however, we suppose it was sufficiently illustrated how they serve as formal foundations of mathematics. Both systems were outlined in adequately technical terms, making precise their concrete representation of mathematical statements as formulas and the

corresponding notion of formal proof as rule-based deduction. Subsequently, some basic Peano arithmetic was developed in order to provide a case study for the respective mathematical practice. Hereby the most remarkable difference is the need for encoding and recursion by hand in axiomatic set theory opposed to the primitive treatment of numbers and computation in dependent type theory. Moreover, the complementary Coq proof scripts are meant to exemplify the use of implemented type theories for the formal verification of mathematical results.

Concerning the studied aspects of practicality, we judge dependent type theory to be the superior foundation. The basic notion of a type as a semantic category for mathematical objects captures usual pre-formal intuitions such as a principal distinction of individuals, predicates, and functions. These distinctions are not syntactically visible in the less structured world of sets and need to be reconstructed internally. Similarly, the primitive support for inductive structures and computation in a type-theoretic framework allows to establish a formal treatment closer to mathematical ideas and is more suitable for applications in computer science. These advantages definitely even out the potentially steeper learning curve one might face when studying dependent type theory with all its syntactical subtleties. It is needless to reiterate that the biggest potential of dependent type theory lies in its applicability for mechanisation, which might shape the future of mathematical research and hence ultimately cause a replacement of axiomatic set theory as accepted standard foundation.

Besides these inner-mathematical considerations, constructivism as embodied in dependent type theory provides a very convincing philosophical base for mathematics. The constructivist interpretation of mathematical objects as intuited conceptions in the human mind answers the ontological questions which cannot be as easily solved by a realistic viewpoint. Furthermore identifying truth with provability, the hardly graspable and formally ill-behaved conception of eternal truth in the platonic universe can be disposed of and results in a pragmatic epistemology. Also, the concrete consequences of constructivism for the accepted logical system are entirely desirable. Intuitionistic proofs carry more information and enable a meaningful analysis of the computational content of formal justifications. Lessening the importance of a complete deduction system, intuitionistic logic is not tied to a first-order view on mathematics, which admit the advantages of higher-order languages such as categoricity and syntactic uniformity.

Of course, all these arguments cannot provide a definite answer to the search for foundations in mathematics. Mathematical habits change over time and the evaluation of competing foundational approaches will always be influenced by the currently predominating paradigms. From this perspective, the mathematical venture is open-ended and hence there is a need for formal flexibility and courage for constant reformation. Thus any satisfactory foundational system should be the best current solution for portraying and explaining the mathematical practice in all its numerous facets. As argued in this thesis, dependent type theory seems to be a good candidate for now.

Bibliography

[Acz78] Peter Aczel. The Type Theoretic Interpretation of Constructive Set Theory. *Studies in Logic and the Foundations of Mathematics*, 96:55–66, January 1978.

[Acz98] Peter Aczel. On Relating Type Theories and Set Theories. In *Types for Proofs and Programs*, Lecture Notes in Computer Science, pages 1–18. Springer, Berlin, Heidelberg, March 1998.

[AH76] Kenneth Appel and Wolfgang Haken. Every Planar Map is Four Colorable. *Bulletin of the American Mathematical Society*, 82(5):711–712, September 1976.

[Awo10] Steve Awodey. *Category Theory*. Oxford University Press, 2010.

[Bar10] Bruno Barras. Sets in Coq, Coq in Sets. *Journal of Formalized Reasoning*, 3(1):29–48, October 2010.

[BB85] Errett Bishop and Douglas S. Bridges. *Constructive Analysis*. Grundlehren der mathematischen Wissenschaften. Springer-Verlag, Berlin Heidelberg, 1985.

[BDS13] Hendrik P. Barendregt, Wil Dekkers, and Richard Statman. *Lambda Calculus with Types*. Perspectives in logic. Cambridge University Press, 2013.

[BF97] Cesare Burali-Forti. Una Questione sui Numeri Transfiniti. *Rendiconti del Circolo Matematico di Palermo (1884-1940)*, 11(1):154–164, December 1897.

[Can95] Georg Cantor. Beiträge zur Begründung der transfiniten Mengenlehre. *Mathematische Annalen*, 46(4):481–512, November 1895.

[Can91] Georg Cantor. A Letter to Hilbert on September 26, 1897. In Herbert Meschkowski and Winfried Nilson, editors, *Briefe*, page 408. 1991.

[CH88] Thierry Coquand and Gérard P. Huet. The Calculus of Constructions. *Information and Computation*, 76(2):95–120, February 1988.

[Chu40] Alonzo Church. A Formulation of the Simple Theory of Types. *The Journal of Symbolic Logic*, 5(2):56–68, 1940.

[Con02] Robert L. Constable. Naïve Computational Type Theory. In *Proof and System-Reliability*, NATO Science Series, pages 213–259. Springer, Dordrecht, 2002.

[Coq87] Thierry Coquand. Metamathematical Investigations of a Calculus of Constructions. January 1987.

[Coq18] Coq Proof Assistant. `http://coq.inria.fr`, 2018.

[Cur34] Haskell B. Curry. Functionality in Combinatory Logic. *Proceedings of the National Academy of Sciences of the United States of America*, 20(11):584–590, 1934.

[Cur58] Haskell B. Curry. *Combinatory Logic*. Amsterdam: North-Holland Pub. Co., 1958.

[Dia75] Radu Diaconescu. Axiom of Choice and Complementation. *Proceedings of the American Mathematical Society*, 51(1):176–178, 1975.

[Dyb91] Peter Dybjer. Inductive Sets and Families in Martin-Löf's Type Theory and Their Set-Theoretic Semantics. In *Logical Frameworks*, pages 280–306. Cambridge University Press, 1991.

[Fra25] Abraham Fraenkel. Untersuchungen über die Grundlagen der Mengenlehre. *Mathematische Zeitschrift*, 22:250–273, 1925.

[GAA+13] Georges Gonthier, Andrea Asperti, Jeremy Avigad, Yves Bertot, Cyril Cohen, Francois Garillot, Stéphane Le Roux, Assia Mahboubi, Russell O'Connor, Sidi Ould Biha, Ioana Pasca, Laurence Rideau, Alexey Solovyev, Enrico Tassi, and Laurent Théry. A Machine-Checked Proof of the Odd Order Theorem. In *Interactive Theorem Proving*, Lecture Notes in Computer Science, pages 163–179. Springer, Berlin, Heidelberg, July 2013.

[Gen35] Gerhard Gentzen. Untersuchungen über das logische Schließen. *Mathematische Zeitschrift*, 39:176–210, 1935.

[Gir72] Jean-Yves Girard. *Interprétation fonctionnelle et élimination des coupures de l'arithmétique d'ordre supérieur*. PhD thesis, Université Paris Diderot, 1972.

[Göd30] Kurt Gödel. Die Vollständigkeit der Axiome des logischen Funktionenkalküls. *Monatshefte für Mathematik und Physik*, 37(1):349–360, 1930.

[Gon08] Georges Gonthier. Formal Proof: The Four-Color Theorem. *Notices of the American Mathematical Society*, 55(11):1382–1393, 2008.

[GTL89] Jean-Yves Girard, Paul Taylor, and Yves Lafont. *Proofs and Types*. Cambridge University Press, New York, NY, USA, 1989.

[Gö31] Kurt Gödel. Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I. *Monatshefte für Mathematik*, 38(1):173–198, 1931.

[HAB+17] Thomas Hales, Mark Adams, Gertrud Bauer, Tat Dat Dang, John Harrison, Le Truong Hoang, Cezary Kaliszyk, Victor Magron, Sean Mclaughlin, Tat Thang Nguyen, Quang Truong Nguyen, Tobias Nipkow, Steven Obua, Joseph Pleso, Jason Rute, Alexey Solovyev, Thi Hoai An Ta, Nam Trung Tran, Thi Diep Trieu, Josef Urban, Ky Vu, and Roland Zumkeller. A Formal Proof of the Kepler Conjecture. *Forum of Mathematics, Pi*, 5, 2017.

[Hal05] Thomas C. Hales. A Proof of the Kepler Conjecture. *Annals of Mathematics*, 162(3):1065–1185, 2005.

[Ham12] Joel David Hamkins. The Set-Theoretic Multiverse. *The Review of Symbolic Logic*, 5(03):416–449, September 2012. arXiv: 1108.4223.

[Hen50] Leon Henkin. Completeness in the Theory of Types. *The Journal of Symbolic Logic*, 15(2):81–91, 1950.

[Hey34] Arend Heyting. *Mathematische Grundlagenforschung Intuitionismus Beweistheorie*. Ergebnisse der Mathematik und ihrer Grenzgebiete. 2. Folge. Springer-Verlag, Berlin Heidelberg, 1934.

[HJ99] Karel Hrbacek and Thomas Jech. *Introduction to Set Theory, Third Edition, Revised and Expanded*. CRC Press, June 1999.

[How80] William Howard. The Formulas-as-Types Notion of Construction. In JP Seldin and JR Hindley, editors, *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus, and Formalism*, pages 479–490. Academic Press, 1980.

[Inc17] Luca Incurvati. Maximality Principles in Set Theory. *Philosophia Mathematica*, 25(2):159–193, June 2017.

[Kan96] Akihiro Kanamori. The Mathematical Development of Set Theory from Cantor to Cohen. *Bull. Symbolic Logic*, 2(1):1–71, 03 1996.

[Kol32] Andrei N. Kolmogorov. Zur Deutung der intuitionistischen Logik. *Mathematische Zeitschrift*, 35(1):58–65, December 1932.

[KS17] Dominik Kirst and Gert Smolka. Categoricity results for second-order ZF in dependent type theory. In *ITP 2017, Brazil*, Sep 2017.

[KS18] Dominik Kirst and Gert Smolka. Large model constructions for second-order ZF in dependent type theory. In *CPP 2018, USA*, Jan 2018.

[Kun14] Kenneth Kunen. *Set Theory: An Introduction to Independence Proofs*. Elsevier, June 2014.

[Kur21] Kazimierez Kuratowski. Sur la notion de l'ordere dans la Théorie des Ensembles. *Fundamenta Mathematica*, 2(1):161–171, 1921.

[Ler09] Xavier Leroy. Formal Verification of a Realistic Compiler. *Commun. ACM*, 52(7):107–115, July 2009.

[Luo18] Zhaohui Luo. Computation and Reasoning: A Type Theory for Computer Science. March 2018.

[McG97] Vann McGee. How we Learn Mathematical Language. *The Philosophical Review*, 106(1):35–68, 1997.

[ML85] Per Martin-Löf. *Intuitionistic Type Theory: Notes by Giovanni Sambin of a Series of Lectures Given in Padua, June 1980*. Prometheus Books, Napoli, June 1985.

[Moo88] Gregory H. Moore. The Emergence of First-Order Logic. In *University of Minnesota Press, Minneapolis*, pages 95–135, 1988.

[NK09] Adam Naumowicz and Artur Kornilowicz. A Brief Overview of Mizar. In *Theorem Proving in Higher Order Logics*, Lecture Notes in Computer Science, pages 67–72. Springer, August 2009.

[Nor08] Ulf Norell. Dependently Typed Programming in Agda. In *Advanced Functional Programming*, Lecture Notes in Computer Science, pages 230–266. Springer, May 2008.

[NWP02] Tobias Nipkow, Markus Wenzel, and Lawrence C. Paulson. *Isabelle/HOL: A Proof Assistant for Higher-order Logic*. Springer, Berlin, Heidelberg, 2002.

[Rey74] John C. Reynolds. Towards a Theory of Type Structure. In *Programming Symposium, Proceedings Colloque Sur La Programmation*, pages 408–423, London, UK, UK, 1974. Springer-Verlag.

[Rus08] Bertrand Russell. Mathematical Logic as based on the Theory of Types. *American Journal of Mathematics*, 30:222–262, July 1908.

[Rus02] Bertrand Russell. A Letter to Frege on Jun 16, 1902. In Jean van Heijenoort, editor, *From Frege to Gödel: A Source Book in Mathematical Logic*, pages 124–125. 2002.

[SF10] Raymond M. Smullyan and Melvin Fitting. *Set Theory and the Continuum Problem*. Dover books on mathematics. Dover Publications, 2010.

[Sha91] Stewart Shapiro. *Foundations Without Foundationalism: A Case for Second-Order Logic*. Oxford University Press, 1991.

[Sie47] Waclaw Sierpiński. L'hypothèse généralisée du continu et l'axiome du choix. *Fundamenta Mathematicae*, 34(1):1–5, 1947.

[Sko22] Thoralf Skolem. Some Remarks on Axiomatized Set Theory. In J. van Heijenoort, editor, *From Frege to Gödel: A Sourcebook in Mathematical Logic*, pages 290–301. toExcel, Lincoln, NE, USA, 1922.

[Sup60] Patrick Suppes. *Axiomatic Set Theory*. Dover Books on Mathematics Series. Dover Publications, 1960.

[Tar43] Alfred Tarski. The Semantic Conception of Truth: and the Foundations of Semantics. *Philosophy and Phenomenological Research*, 4(3):341–376, 1943.

[Tur36] Alan M. Turing. On Computable Numbers, with an Application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, 2(42):230–265, 1936.

[Uni13] The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. `https://homotopytypetheory.org/book`, Institute for Advanced Study, 2013.

[Uzq99] Gabriel Uzquiano. Models of Second-Order Zermelo Set Theory. *The Bulletin of Symbolic Logic*, 5(3):289–302, 1999.

[vN23] John von Neumann. Zur Einführung der transfiniten Zahlen. *Acta Scientiarum Mathematicarum*, 1(4):199–208, 1923.

[vN25] John von Neumann. An Axiomatisation of Set Theory. In Jean van Heijenoort, editor, *From Frege to Gödel: A Source Book in Mathematical Logic, 1879–1931*, pages 393–413. Harvard University Press, 1925.

[Vä01] Jouko Väänänen. Second-Order Logic and Foundations of Mathematics. *The Bulletin of Symbolic Logic*, 7(4):504–520, 2001.

[Wad15] Philip Wadler. Propositions As Types. *Commun. ACM*, 58(12):75–84, November 2015.

[Wer97] Benjamin Werner. Sets in Types, Types in Sets. In *Theoretical Aspects of Computer Software*, pages 530–546. Springer, Heidelberg, September 1997.

[Wil69] Neil H. Williams. On Grothendieck Universes. *Compositio Mathematica*, 21(1):1–3, 1969.

[WR10] Alfred N. Whitehead and Bertrand Russell. *Principia Mathematica*. Cambridge University Press, 1st edition, 1910.

[Zer08] Ernst Zermelo. Neuer Beweis für die Möglichkeit einer Wohlordnung. *Mathematische Annalen*, 65:107–128, 1908.

[Zer30] Ernst Zermelo. Über Grenzzahlen und Mengenbereiche: Neue Untersuchungen über die Grundlagen der Mengenlehre. *Fundamenta Mathematicæ*, 16:29–47, 1930.