Saarland University
Faculty of Mathematics and Computer Science

Master's Thesis

Mechanising Set Theory in Coq

The Generalised Continuum Hypothesis and the Axiom of Choice

Author
Felix Rech

Advisor
Dominik Kirst

Supervisor
Prof. Dr. Gert Smolka

Reviewers
Prof. Dr. Gert Smolka
Prof. Dr. Holger Hermanns

11th June 2020

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

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.

Einverständniserklärung

Ich bin damit einverstanden, dass meine (bestandene) Arbeit in beiden Versionen in die Bibliothek der Informatik aufgenommen und damit veröffentlicht wird.

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 Computer Science Department.

Acknowledgements

I want to thank Dominik Kirst, the advisor of my thesis. He always had time for me, offering valuable feedback. He gave me the freedom to develop my own ideas. I also want to thank Professor Smolka and Professor Hermanns who agreed to review my thesis.

Abstract

This thesis presents formalisations of results about the *axiom of choice* and the *generalised continuum hypothesis* in two different versions of Zermelo-Fraenkel set theory, all within the language of the Coq proof assistant. The work is divided into three major parts.

The first part describes the formalisation of *Sierpiński's theorem* which states that the generalised continuum hypothesis implies the axiom of choice. We consider this theorem in the *second-order* version of set theory ZF_2 that is intended to make internal constructions and proofs easy. It contains versions of the axiom of separation and the axiom of replacement which, in contrast to the standard first-order axiom schemes, accept arbitrary Coq-definable relations instead of just first-order formulas.

In the second part, we introduce the weaker axiomatisation of set theory ZF' that gives us more freedom for model construction. This axiomatisation is equivalent to the typical one in first-order logic, except that we require well-foundedness of sets in the external sense. The only fundamental difference to ZF_2 lies in the axioms of separation and replacement which assume predicates and functions expressed as first-order formulas. To make formulations of complex statements in first-order logic feasible, we introduce a compositional reification framework that covers a large subset of the Coq language, including higher-order functions and dependent types. This allows us to formulate statements in the way that seems most natural in Coq and construct their first-order representation later. In many cases, the construction of those representations is routine and can be automated.

In the third part, we present a proof of the relative consistency of the axiom of choice over ZF'. Assuming any model of ZF', we define an internal model L that is known as the *constructible universe*. We then show that L satisfies the axiom of choice. To this end, we need to formalise first-order logic in ZF', a strong case study for the reification framework. The proof is not possible in ZF_2 since we would not be able to show that L is a model of ZF_2. The formalisation of the consistency result is still work-in-progress.

Contents

1 Introduction

In 1891, Georg Cantor showed that there is a set that is greater than the set $\mathbb{N}$ of all natural numbers [4]. This leads to the question if there is a smallest set greater than $\mathbb{N}$. One candidate would be the power set $\mathcal{P}(\mathbb{N})$ that contains all sets of natural numbers. It seems difficult to construct a set of size strictly between $\mathbb{N}$ and $\mathcal{P}(\mathbb{N})$, but it seems equally difficult to disprove the existence of such a set. This observation led to the formulation of the continuum hypothesis which states that $\mathcal{P}(\mathbb{N})$ is the smallest set greater that $\mathbb{N}$. This question is also known as Hilbert's first problem [15]. But the observation is not really specific to $\mathbb{N}$. An obvious generalisation of the problem is the statement that for all infinite sets A, the power set $\mathcal{P}(A)$ is the smallest set greater than A. This is called the **generalised continuum hypothesis** (GCH).

Another difficult question of basic set theory is the validity of the so-called **axiom of choice** (AC). The axiom states that for every collection of non-empty sets, it is possible to choose one element from each at the same time. If the collection is finite, then this is easy because we can just go through all the sets and choose elements one at a time. If the collection is infinite then additional properties can help us make a choice. For example, following Russell [16], from infinitely many pairs of shoes, we can simply choose all the left ones. But it gets difficult when we have no way to distinguish elements. How do we choose elements for infinitely many pairs of socks?

The axiom of choice is of great interest as a necessary ingredient for many important theorems. Without AC, there might be surjections without a right-inverse. There might be sets that are not comparable in their size. For example, there might be a set that is neither smaller nor greater or equal to the set of natural numbers. This is important because it explains why the proofs in Section 4.2 are difficult at all. But on the other hand, AC also has some counter-intuitive consequences. For example, it allows us to define a well-order on the set of real numbers or to decompose a ball into finitely many pieces and reassemble them to obtain two balls of the same size [1].

The validity of GCH and AC depends very much on their exact formalisation. To answer them, we need to make more precise what sets there are, when one set is greater than another and what it means to choose elements from a collection of sets. That is where **Zermelo-Fraenkel set theory** (ZF) [26, 34] comes in. ZF is a collection of statements called axioms that tell us how we can create sets and what properties they have. For example, one of the axioms states that there is an empty set and another one states that sets are equal if they have the same elements.

Typically, ZF is treated in the formal language called *first-order logic* that goes a step further and makes precise what a statement is and what it means for a statement to be true. In this thesis, we use the more versatile language of the Coq proof assistant instead. We work in two different versions of set theory. The first is a stronger second-

order formulation ZF_2 [17] that is naturally embedded in Coq and more convenient to use. The second version is ZF', a compromise between ZF_2 and the typical first-order axiomatisation. Both share a common core but differ in the formulation of two axioms. The first difference is in the axiom of separation, also called specification. It allows us to build sets that we usually write in the form $\{x : A \mid P(x)\}$. In ZF', the predicate P has to be given by a first-order formula with a free variable for x. In ZF_2, we allow arbitrary predicates. The second difference is in the axiom of replacement which we use to construct sets of the form $\{f(x) \mid x : A\}$. In ZF', the function f has to be given by a first-order formula with two free variables that describe the relation between the input and output of f. In ZF_2, we can use an arbitrary function.

From a metatheoretical perspective, ZF_2 is a conservative extension of first-order ZF. That means that they share the same theorems in first-order logic [21]. Hence, we can easily prove such theorems in the more convenient ZF_2 and know that they hold for first-order ZF as well. ZF', on the other hand, is useful when we build models because it imposes fewer constraints. ZF_2 is categorical in every cardinality, that is, equipotent models are always isomorphic [17]. This makes it unlikely that we can at all construct models that differ significantly.

Throughout this thesis, we assume four global axioms in addition to the standard rules of Coq. We assume the law of excluded middle that tells us that every proposition is either true or false. This assumption is standard in classical set theory. We also assume the principle of definite description which states that, whenever a predicate has a unique solution, we can actually work with this solution and use it for further definitions and statements. This is important because ZF only asserts the existence of sets with certain properties but doesn't give us those sets explicitly, which is inconvenient. Moreover, the axiom of definite description is our only way to use sets for the construction of many other kinds of objects, for example, to construct a natural number from its set encoding. The other axioms are the axioms of function extensionality and proposition extensionality. They tell us that functions are equal if they are point-wise equal and propositions are equal if they are equivalent. One motivation for those axioms is that they will give us the right equality on classes of sets that are given by predicates.

We begin this thesis by the introduction of the common axioms and constructions of ZF' and ZF_2. Then, we introduce full ZF_2 which we use to prove our first main result, Sierpiński's theorem. It states that GCH implies AC. We loosely follow a presentation of the proof by Gillman [8]. As an intermediate step, we prove that GCH implies the well-ordering theorem which states that every set has a so-called well-order [33]. To prove AC from the well-ordering theorem, is not difficult. For the other part of the proof, we develop some cardinal arithmetic and introduce ordinal numbers which are special sets that have well-orderings on them. To sketch this part, assume that GCH holds and fix a set A for which we need a well-ordering. A notable step in our proof is the construction of the Hartogs number of A. The Hartogs number is the least ordinal number that is not smaller or equal to A. This property does not immediately imply that the Hartogs number is greater than A; they could be incomparable. But we show that, in the context of GCH, A is in fact greater. This allows us to transport the well-ordering from the Hartogs number to A and we are finished.

In the second major part, we define first-order logic and introduce the remaining axioms of ZF'. The axiom of separation in ZF' requires us to represent predicates by first order-formulas. The axiom of replacement requires us to represent functions by first-order formulas. Constructing those representations by hand is in many cases not feasible. For that reason, we introduce a framework to construct such representations in a compositional way. We generalise the notion of representations to other objects than predicates and functions. Those include higher-order functions and dependent functions. But the framework is also open for extension to other types. Representations can typically be composed of representations of subterms. This approach makes automation easy. One can automatically construct a representation of a term if one has representations for all atomic subterms.

In the final part, we work towards the relative consistency of AC over ZF'. We loosely follow a presentation of the proof by Smullyan and Fitting [26]. First, we assume that we have any model of ZF'. We then define a class L, the *constructible universe*, in this model [10]. We show that L forms itself a model of ZF'. The goal of the definition of L is to keep L small and its structure mostly independent from the outer model. We achieve this by an iterative process: In every step, we include only sets that can be constructed by application of the axioms of ZF' to sets that are already contained. We repeat this step transfinitely often, to saturate L in the sense that every constructible set is already included and, hence, L is closed under the set-construction axioms of ZF'. In the next step, we show that the version of L *inside* of L as a model contains all sets. This is called the axiom of constructibility. It gives us a concrete description of the collection of all sets. From this description, we can put all sets in a well-order, which allows us to deduce the axiom of choice.

Contribution. Our most novel contribution is the compositional and extensible reification framework for first-order logic. It allows us to reify a vast fragment of the Coq language and can be adapted to other theories than first-order logic. The second contribution is the formalisation of Sierpiński's theorem over ZF_2. Finally, we demonstrate how the set theory ZF', its embedding in Coq, and the reification framework can help with the relative consistency proof of AC.

Outline. In Chapter 2, we introduce basic notations, and definitions related to cardinality and orderings. In Chapter 3, we define the set theory ZF_2 and many basic constructions based on that theory. Towards the end of the chapter, we introduce ordinals and prove all their general properties that we need. In Chapter 4, we construct the Hartogs number and prove results immediately leading to Sierpiński's theorem. In Chapter 5, we define the set theory ZF', referring to axioms from Chapter 3. We develop the reification framework in and present the result in Section 5.2. Then, we apply the framework to construct representations of all the basic constructions from Section 5.3. In Chapter 6, we present the full proof of the relative consistency of AC. We end the thesis with remarks about the formalisation(Chapter 7), related work (Chapter 8) an a conclusion (Chapter 9).

2 Preliminaries

The implicit formal foundation for this whole thesis is the Calculus of Inductive Constructions, the formal language of the Coq proof assistant [28]. By $\mathbb{P}$, we denote the type of propositions and by Type_i the type universe of level i but we typically leave the universe level implicit.

For any function $f : A \to A$ and natural number n, we denote by f^n the n-fold iteration of f. We let 2^A stand for the type of predicates $A \to \mathbb{P}$ and call the elements of 2^A **subtypes** of A. Sometimes, we use notations of the form $(x_i : A_i)_{i:I}$ to mean $\lambda i : I.\, x_i : A_i$ when we want to take the emphasis from the argument i. We switch implicitly between the curried and the uncurried style of function application when convenient. We write some arguments as subscripts and leave some implicit for a cleaner notation.

2.1 Cardinality

2.1.1 Definition. A **bijection** from a type A to a type B is a function $f : A \xrightarrow{\sim} B$ for which there is an **inverse**, that is, a function $f^{-1} : B \to A$ satisfying $f(f^{-1}(y)) = y$ for all $y : B$ and $f^{-1}(f(x)) = x$ for all $x : A$.

If the type of bijections from A to B is inhabited then we say that A and B have the same **cardinality** or that they are **equipotent**, abbreviated as $A \sim B$.

2.1.2 Fact. *Bijections satisfy three basic properties:*

1. *The identity function on any type is a bijection.*
2. *The composition of two bijections is a bijection.*
3. *Every bijection has an inverse bijection.*

This implies that equipotency is an equivalence relation.

2.1.3 Lemma. $2^{A+B} \sim 2^A \times 2^B$ *for all types* A *and* B.

2.1.4 Definition. An **injection** from a type A to a type B is a function $f : A \to B$ such that for all $x, x' : A$ with $f(x) = f(x')$, it follows that $x = x'$. We denote the type of injections from A to B by $A \hookrightarrow B$.

If the type $A \hookrightarrow B$ is inhabited then we say that A has smaller or equal cardinality or just that it is smaller or equal to B, abbreviated as $A \leq B$.

2.1.5 Fact. *Basic properties of injections are:*

1. *The identity function on any type is an injection.*
2. *The composition of two injections is an injection.*

2.1.6 Lemma. *Every bijection is also an injection.*

We will talk about infinite types. There are different possible definitions of infinity that are not equivalent without further axioms. The most common definition states that a type is infinite if we cannot list its members. For our purpose, the following is more convenient.

2.1.7 Definition. A type A is **Dedekind-infinite** or just **infinite** if $\mathbb{N} \leq A$.

We also define the concept of *surjections* with the exclusive goal to formulate a variant of *Cantor's theorem* that we will need in the proof of Sierpiński's theorem.

2.1.8 Definition. A function f from a type A to a type B is a **surjection** if for all $y : B$ there is an $x : A$ such that $f(x) = y$.

2.1.9 Cantor's Theorem. *There is no surjection from any type into its power type.*

2.2 Orders

We define the notions of orders, order isomorphisms, well-foundedness and well-orders and state some lemmas about those.

2.2.1 Definition. A **strict total order**, or just **order**, on a type A is a binary relation $\_ < \_ : A \to A \to \mathbb{P}$, that satisfies three properties:

Transitivity: If $x < y$ and $y < z$ then $x < z$ for all $x, y, z : A$.

Trichotomy: All $x, y : A$ satisfy exactly one of $x < y$, $x = y$ or $x > y$.

An **ordered type** is a type that has an order.

2.2.2 Definition. An **order isomorphism** from an ordered type A to an ordered type B is a bijection $f : A \xrightarrow{\sim} B$ that preserves the order in both directions:

$$x < y \leftrightarrow f(x) < f(y).$$

If the type of such isomorphisms is inhabited, we say that A as ordered type is **order isomorphic** to B, abbreviated as $A \simeq B$.

2.2.3 Fact. *There are three basic laws of order isomorphisms:*

1. *The identity function on any ordered type is an order isomorphism.*
2. *The composition of any two order isomorphisms is an order isomorphism.*
3. *Every order isomorphism has an inverse order isomorphism.*

This implies that isomorphism between ordered types is an equivalence relation.

2.2.4 Definition. A relation $R : A \to A \to \mathbb{P}$ on a type A is **well-founded** if it satisfies the principle of **well-founded induction**: To prove a property $P : A \to \mathbb{P}$ on all members of A, it suffices to show that an arbitrary but fixed $x : A$ satisfies P under the additional assumption that all $y : A$ with $y\, R\, x$ satisfy P. In symbolic notation:

$$\begin{aligned}&\forall P : A \to \mathbb{P}.\\ &\quad \Big(\forall x : A.\, (\forall y : A.\, y\, R\, x \Rightarrow P(y)) \Rightarrow P(x)\Big)\\ &\quad \Rightarrow \forall x : A.\, P(x).\end{aligned}$$

Next we state the principle of well-founded recursion which is just a constructive version of well-founded induction.

2.2.5 Lemma (Well-founded recursion)**.** *Fix a well-founded relation $R : A \to A \to \mathbb{P}$. There is a function*

$$\mathsf{rec}_R : \prod_{B:A\to\mathsf{Type}} (\prod_{x:A}(\prod_{y:A} y\, R\, x \to B(y)) \to B(x)) \to \prod_{x:A} B(x).$$

This function satisfies that $\mathsf{rec}_R(B, f, x) = f(x, (\lambda y.\, \lambda\_.\, \mathsf{rec}_R(B, f, y)))$ for all B, f and x.

Proof. The fundamental ingredient of this proof is the axiom of definite description. We define inductively the concept of accessibility witnesses: An accessibility witness of a member $x : A$ is function that maps every member $y : A$ with $y\, R\, x$ to an accessibility witness of y. We can prove by well-founded induction that every member of A has a unique accessibility witness and we can obtain that witness by definite description. It is then easy to define rec_R by recursion on the accessibility witnesses. □

2.2.6 Definition. A **well-order** is an order that is well-founded (Definition 2.2.4).

2.2.7 Lemma. *Every non-empty subtype of a well-ordered type has a least element.*

Proof. Fix a non-empty subtype A of a well-ordered type and a member $x : A$. We apply well-founded induction on x. Either x is the least element or there is a smaller element $y < x$. In that case, we can apply the inductive hypothesis on y and are done. □

3 Second-order set theory

This chapter introduces ZF_2, the first of our two set theories. We will pose the axioms that characterise models of this theory. Those axioms are divided into two groups. The first group contains core axioms which are shared by the set theory ZF' which we will introduce in Chapter 5. Most of those axioms are trivially equivalent to a corresponding axiom from the typical formulation of Zermelo-Fraenkel set theory in first-order logic. But two of them are slightly stronger, as we will see in Section 5.1. The second group consists of the so-called axioms of separation and replacement which allow us to construct sets from predicates or functions respectively. Those axioms cannot be formulated in first-order logic since functions and predicates are not treated as objects there. In first-order logic, one does typically use axiom schemes instead, that is, infinite sets of axioms that follow a common pattern. But even those infinite axiom schemes are not as strong as the second-order formulation of separation and replacement that we use here.

We start by definition of the basic logical structure that our axioms will talk about.

3.0.1 Definition. A **ZF structure** is a type with a binary relation. We represent this relation by the infix operator $\in$. We call the members of a ZF structure **sets** and think of them as collections consisting of other members, where $x \in y$ means that x is an **element** of y.

For the rest of the chapter, we fix a ZF structure $\mathscr{S}$. If we work with definitions or notations that depend on a ZF structure and it is not clear which structure we mean, then we indicate this as a superscript as in $\in^{\mathscr{S}}$.

3.0.2 Definition. A **class** is a collection of sets given by a predicate $P : \mathscr{S} \to \mathbb{P}$. We denote such a class by $\{x \mid P(x)\}$.

We identify every set x with its corresponding class $\{y \mid y \in x\}$ and generalise the notation $x \in A$ to mean that the set x is an element of the *class* A. A class **is a set** if there is a set with the same elements. Moreover, we let every class stand for the type of its elements. For example, if A and B are classes then $A \to B$ is the type of functions that take an element of class A and return an element of class B. If A is a class and $P : A \to \mathbb{P}$ is a predicate then the statement $\forall x : A.\, P(x)$ talks about all elements of the class A.

By $\{f(x) \mid x : A\}$, we denote the class given as the image of a function $f : A \to \mathscr{S}$. More generally, by $\{g(x_1, \dots, x_n) \mid x_1 : A_1, \dots, x_n : A_n\}$, we denote the image of a function $g : \prod_{x_1 : A_1} \cdots \prod_{x_n : A_n} \mathscr{S}$ where $x_1, \dots, x_i$ may occur as free variables in A_{i+1}.

We will implicitly treat every class A as a ZF structure equipped with the element relation $\in|_A : A \to A \to \mathbb{P}$. In natural language, we will use formulations like 'the element relation *over* A' or 'classes over *over* A'.

3.1 Core axioms

Most of the ZF_2 axioms describe ways to construct sets. Our first two axioms are the only exceptions. Those tell us something about the structure of sets.

3.1.1 Axiom of Extensionality. *Two sets are equal if they contain the same elements.*

Intuitively, this means that there is nothing more to a set but the collection of its elements.

3.1.2 Axiom of Foundation. *The element relation on the type of sets is well-founded (Definition 2.2.4).*

In other words, if we want to prove a property P on an arbitrary but fixed set then we may additionally assume that P holds on all its elements. This has the following simple but important consequence.

3.1.3 Lemma. *No set contains itself.*

Proof. We prove the statement by well-founded induction on the element relation. Fix a set x and assume that $x \in x$. This allows us to apply the inductive hypothesis on x, which tells us exactly $x \notin x$. □

Now, we start introducing different ways to construct sets. For each axiom, we define a certain class and the corresponding axiom will provide a condition under which this class is a set.

3.1.4 Definition. By $\emptyset$, we denote the empty class.

3.1.5 Axiom of Empty Set. *The empty class is a set.*

3.1.6 Definition. By $\{x_1, \ldots, x_n\}$, we denote the class that contains exactly the elements $x_1, \ldots, x_n$. In the special case of only one element x, we call $\{x\}$ the **singleton** with element x. In the case of two elements x and y, we call $\{x, y\}$ the **unordered pair** of x and y.

3.1.7 Axiom of Pairing. *The unordered pair of any two sets is a set.*

This implies that every singleton is also a set since $\{x\} = \{x, x\}$ by the axiom of extensionality.

3.1.8 Definition. For a class A, we define the **union** of A as

$$\bigcup A \coloneqq \{x \mid \exists B \in A, x \in B\}.$$

3.1.9 Axiom of Union. *The union of any set is a set.*

3.1.10 Definition. We say that a class A is a **subclass** of a class B if all elements of A are also in B. In symbolic notation, we write this as $A \subseteq B$. If A is a set then we call it a **subset**.

We define the **power class** of a class A as the collection of all subsets

$$\mathcal{P}(A) \coloneqq \{x \mid x \subseteq A\}.$$

3.1.11 Axiom of Power Set. *The power class of any set is also a set. We call it the* ***power set*** *of A.*

So far, none of the axioms allow us to construct infinite sets. All available constructions preserve finiteness. The next axiom is intended to change that. But having an infinite set alone is not enough. We want more information about the elements of that set. We will fix a specific injection from $\mathbb{N}$ to $\mathscr{S}$ and assume that the image of this injection is a set. We can then use this injection as an encoding of $\mathbb{N}$.

3.1.12 Definition. An **encoding** of a type A consists of a class $\bar{A}$ and a bijection $\mathsf{encode}_A : A \xrightarrow{\sim} \bar{A}$. We denote the inverse of this bijection by decode_A.

We say that a type A is **set-like** if it has an encoding and $\bar{A}$ is a set.

Of course, it suffices to provide an injection $\mathsf{encode}_A : A \hookrightarrow \mathscr{S}$. Then we can define $\bar{A}$ as the image of this injection and encode_A will trivially be a bijection into $\bar{A}$. That us usually the most convenient way to define an encoding.

Note that every class is class-like by this definition and every set is set-like. In both cases, the identity function can be used as the encoding.

Encodings will become useful in combination with the axiom of replacement. This axiom takes a set A and a function $f : A \to \mathscr{S}$ as input and states that the image $\{f(x) \mid x \in A\}$ is also a set. But we can easily show that the same holds if A is any set-like type.

In the first-order axiomatisation, encodings will be even more important. First-order logic allows us to quantify over classes, as long as we can represent them in first-order logic. Quantification over arbitrary types is not possible but we can make it work when the types are class-like. Abstractly speaking, it is helpful to make encodings explicit because this allows us connect all the definitions and lemmas that we already have (on $\mathbb{N}$ for example) with the world of set. There is no need to redefine or reprove them.

3.1.13 Definition. We define an encoding of the natural numbers by

$$\begin{aligned} \mathsf{encode}_{\mathbb{N}}(0) &\coloneqq \emptyset \\ \mathsf{encode}_{\mathbb{N}}(n+1) &\coloneqq \mathsf{encode}_{\mathbb{N}}(n)^+, \end{aligned}$$

where $x^+ \coloneqq \bigcup\{x, \{x\}\}$ is the **successor** of x.

To see that this is an injection, we use the fact that for two different natural numbers n and k, we have either $n < k$ or $n > k$. Those inequalities imply by induction and definition of the successor that $\mathsf{encode}(n) \in \mathsf{encode}(k)$ or vice versa. We conclude that n and k have different encodings because no set contains itself (Lemma 3.1.3).

The idea behind this definition is due to von Neumann [30] and can be generalised to ordinals which we introduce in Section 3.5.

3.1.14 Axiom of Infinity. *The encoding $\bar{\mathbb{N}}$ is a set.*

This axiom implies that $\mathbb{N}$ is set-like and the set $\bar{\mathbb{N}}$ must be infinite.

3.2 Exclusive ZF_2 axioms

Now, we introduce two set-construction axioms that do not only take sets as input but additionally predicates and functions respectively. Those axioms are not suitable for some of the proofs in chapter 6. The reason is that they introduce to much of the metalogic (the language of Coq) into the set theory. That makes it more difficult to understand which kinds of sets exist and which don't. We would need to answer this question about predicates and functions in the metalogic. And to answer those is difficult. That is the reason why we use ZF_2 only for the next chapter and introduce the weaker set theory ZF' later.

We work with ZF_2 at all because it is simpler and expresses the ideas in a purer way. ZF' is closer to the standard foundation of mathematics, but ZF_2 seems faithful to Zermelo's informal view of set theory [34].

3.2.1 Axiom of Separation. *Given a set A and a property $P : A \to \mathbb{P}$, the subclass*

$$\{x \in A \mid P(x)\}$$

is a set.

In other words, this states that every subclass of a set is also a set.

3.2.2 Axiom of Replacement. *Given a set A and a function $f : A \to S$, the image*

$$\{f(x) \mid x : A\}$$

is a set.

In cases where we already know that the image of f is bounded by a set B, this is redundant to the instance of separation on the class $\{y \in B \mid \exists x : A.\, f(x) = y\}$. The axiom of replacement is only necessary in more exotic cases.

There are versions of this axiom that use functional relations instead of functions (for example those by Kirst [17] or Barras [3]). Those versions imply the axiom of definite description on sets. But we already assume the general axiom of definite description. It allows us to take a functional relation and produce the corresponding function. Hence, both version of the axiom of replacement are equivalent in our setting.

To end this section, we combine all the axioms into the definition of ZF_2.

3.2.3 Definition. The ZF-Structure $\mathscr{S}$ is a **ZF_2-Model** if it satisfies all axioms in this chapter.

For the remaining sections of this chapter, we will assume that $\mathscr{S}$ is such a ZF_2-Model.

3.3 Basic constructions

The axioms already introduced some of the most widely used set constructors. In the same spirit, we derive some further useful ways to construct sets. We start with a generalisation of the axiom of replacement that allows us to range over arbitrary set-like types, not just sets.

3.3.1 Lemma. *Given a set-like type A and a function $f : A \to \mathscr{S}$, the image $\{f(x) \mid x : A\}$ is a set.*

Proof. By the axiom of extensionality, we have that $\{f(x) \mid x : A\} = \{f(\mathsf{decode}(x)) \mid x : \bar{A}\}$. The claim follows since the right side is a set by the axiom of replacement. □

We continue with binary and indexed versions of union and intersection

3.3.2 Definition. We define the **binary union** of two classes A and B as

$$A \cup B \coloneqq \{x \mid x \in A \vee x \in B\}.$$

3.3.3 Lemma. *Binary union of two sets is a set.*

Proof. The claim follows by the equality $A \cup B = \bigcup\{A, B\}$. □

3.3.4 Definition. We define the **binary intersection** of two classes A and B as

$$A \cap B \coloneqq \{x \mid x \in A \wedge x \in B\}.$$

3.3.5 Lemma. *The binary intersection of two sets is a set.*

Proof. By the equality $A \cap B = \{x \in A \cup B \mid x \in A \wedge x \in B\}$ since the right side is a set by Lemma 3.3.3 and the axiom of separation. □

In the same style as the binary union and intersection, we describe indexed versions of those operations.

3.3.6 Definition. Given a type I and a family of classes $A : I \to \mathsf{Class}$, we define

$$\bigcup_{i:I} A(i) \coloneqq \{x \mid \exists i : I.\, x \in A(i)\}.$$

We call this operation the **indexed union**.

3.3.7 Lemma. *Given a set-like type I and a family of sets $A : I \to \mathscr{S}$, the indexed union $\bigcup_{i:I} A(i)$ is a set.*

Proof. This follows from the equality $\bigcup_{i:I} A(i) = \bigcup\{A(i) \mid i : I\}$. □

3.3.8 Definition. Given a class I and a family of classes $A : I \to \mathsf{Class}$, we define

$$\bigcap_{i:I} A(i) \coloneqq \{x \mid \forall i : I.\, x \in A(i)\}.$$

We call this operation the **indexed intersection**.

3.3.9 Lemma. *Given a* non-empty *set I and a family of sets $A : I \to \mathscr{S}$, the indexed intersection $\bigcap_{i:I} A(i)$ is a set.*

Proof. The claim follows by the equality $\bigcap_{i:I} A(i) = \{x : \bigcup_{i:I} A(i) \mid \forall i : I.\, x \in A(i)\}$. □

Note that we needed the additional condition of non-emptiness. Otherwise, the intersection becomes the class of all sets which cannot be a set (Lemma 3.1.3). The proof would not work because the right side of the equality would be the empty class.

3.3.10 Definition. We define the difference of two classes A and B as

$$A \setminus B \coloneqq \{x \in A \mid x \notin B\}.$$

3.3.11 Lemma. *The difference $A \setminus B$ of a set A and a class B is a set.*

Proof. This follows immediately by the axiom of separation. □

Finally, we use the indexed union to treat the general version of replacement with an arbitrary number of arguments.

3.3.12 Lemma. *Fix a function $f : \prod_{x_1:A_1} \dots \prod_{x_n:A_n} \mathscr{S}$, where every A_{i+1} stands for a set-like type and may contain $x_1, \dots, x_i$ as free variables. The image*

$$\{f(x_1, \dots, x_n) \mid x_1 : A_1, \dots, x_n : A_n\}$$

is a set.

Proof. The claim follows by the equality

$$\{f(x_1, \dots, x_n) \mid x_1 : A_1, \dots, x_n : A_n\} = \bigcup_{x_1:A_1} \dots \bigcup_{x_n:A_n} \{f(x_1, \dots, x_n)\}.$$

□

3.4 Important encodings

In this section, we take a look at essential type constructors of type theory like the sum and product on types and provide conditions under which the resulting types are class-like or even set-like. We start with the product. The first step is an encoding of ordered pairs of *sets*.

3.4.1 Definition. We define the **pairing function** as

$$\langle \_, \_ \rangle : \mathscr{S} \to \mathscr{S} \to \mathscr{S}$$
$$\langle x, y \rangle \coloneqq \{\{x\}, \{x, y\}\}.$$

This way to represent pairs was introduced by Kuratowski [20]. Hence, the sets $\langle x, y \rangle$ are known as Kuratowski pairs. The fundamental property of the pairing function is its injectivity.

3.4.2 Lemma. *The pairing function is injective.*

Proof. Fix sets x, y, x' and y' and assume that $\langle x, y \rangle = \langle x', y' \rangle$ which is the same as $\{\{x\}, \{x, y\}\} = \{\{x'\}, \{x', y'\}\}$. Generally, an equality between unordered pairs falls into one of two possible cases:

- Assume that $\{x\} = \{x'\}$ and $\{x, y\} = \{x', y'\}$. Then we get $x = x'$ from the first equation which turns the second one into $\{x, y\} = \{x, y'\}$. This also implies $y = y'$.
- Assume that $\{x\} = \{x', y'\}$ and $\{x, y\} = \{x'\}$. The first equation implies $x' = x = y'$ which turns the second one into $\{x, y\} = \{y'\}$ which implies $y = y'$. □

From the pairing function, we can define encodings of arbitrary pairs, as long as we can encode both components:

3.4.3 Lemma. *A sigma type $\sum_{x:A} B(x)$ is class-like if A and all $B(x)$ are class-like.*

Proof. We define an encoding by $\mathsf{encode}((x, y)) \coloneqq \left\langle \mathsf{encode}_A(x), \mathsf{encode}_{B(x)}(y) \right\rangle$. Injectivity follows from injectivity of encode_A, $\mathsf{encode}_{B(x)}$ and the pairing function. □

3.4.4 Lemma. *A sigma type $\sum_{x:A} B(x)$ is set-like if A and all $B(x)$ are set-like.*

Proof. The corresponding class is $\{\langle x, y\rangle \mid x : A, y : B(x)\}$. This class is a set by Lemma 3.3.12. □

Of course, the same results apply to simple product types of the form $A \times B$, since those are just a special case of sigma types.

The specific encoding of pairs makes it easy to prove the next lemma. The lemma may not seem meaningful here, but it will be important in the next chapter, specifically for Fact 4.1.2.

3.4.5 Lemma. *Given a class A, we have the inequality $A \times A \leq \mathcal{P}^2(A)$.*

Proof. The inequality is given by the encoding which maps $(x, y) : A \times A$ to $\{\{x\}, \{x, y\}\} \in \mathcal{P}^2(A)$. □

We continue with the sum of types, also known as disjoint union.

3.4.6 Lemma. *The sum $A + B$ of any two class-like types A and B is also class-like.*

Proof. We define an encoding by

$$\mathsf{encode}_{A+B}(\mathsf{inj}_1(x)) \coloneqq \mathsf{encode}_{\mathbb{N}\times A}((0, x))$$
$$\mathsf{encode}_{A+B}(\mathsf{inj}_2(y)) \coloneqq \mathsf{encode}_{\mathbb{N}\times B}((1, y)).$$

To prove injectivity, fix two members $u, v : A + B$ and assume that $\mathsf{encode}(u) = \mathsf{encode}(v)$. There are four cases to distinguish. We only consider two of them, the other two are analogous.

- Assume that $u = \mathsf{inj}_1(x)$ and $v = \mathsf{inj}_1(x')$ for some $x, x' : A$. Then $\mathsf{encode}((0, x)) = \mathsf{encode}((0, x'))$ and, hence, $x = x'$.
- Assume that $u = \mathsf{inj}_1(x)$ and $v = \mathsf{inj}_2(y)$ for some $x : A$ and $y : B$. Then $\mathsf{encode}((0, x)) = \mathsf{encode}((1, y))$ and, hence, $0 = 1$, a contradiction. □

3.4.7 Lemma. *The direct sum of any two set-like types is also set-like.*

Proof. The encoding class is a subclass of the set $\overline{\mathbb{N} \times A} \cup \overline{\mathbb{N} \times B}$. This is a set by Lemma 3.4.4. Hence, the encoding class is a set by the axiom of separation. □

Next, we consider function types.

3.4.8 Lemma. *A dependent function type $\prod_{x:A} B(x)$ is class-like if A is set-like and all $B(x)$ are class-like.*

Proof. We define an encoding that maps every function to its so-called graph, the set of all input-output pairs: $\mathsf{encode}(f) \coloneqq \{\mathsf{encode}_{\sum_{x:A} B(x)}((x, f(x)) \mid x : A\}$. This encoding function produces sets by the axiom of replacement. That holds only because we required that A is set-like. The encoding function is also injective because the assumption that $\{\mathsf{encode}((x, f(x)) \mid x : A\} = \{\mathsf{encode}((x, g(x)) \mid x : A\}$ for any two functions f and g implies that f and g are pointwise equal, hence, by function extensionality, $f = g$. □

3.4.9 Lemma. *A dependent function type $\prod_{x:A} B(x)$ is set-like if A and all $B(x)$ are set-like.*

Proof. The encoding class is a subclass of the set $\mathcal{P}(\overline{\sum_{x:A} B(x)})$ and, hence, set-like by the axiom of separation. □

Finally, we come to a class of types with a very simple encoding, namely propositions. In our very classical setting, their members (which are proofs) do not carry any information. Nonetheless, it will be convenient to know that propositions are class like since it allows us to use them for replacement, as in $\{\ldots \mid n : \mathbb{N}, n \text{ is even}\}$.

3.4.10 Lemma. *All propositions are set-like.*

Proof. Fix a proposition P. We use the trivial encoding that maps everything to $\emptyset$. Depending on whether P is true, we get $\bar{P} = \emptyset$ or $\bar{P} = \{\emptyset\}$. Both are sets. □

3.5 Ordinals

Now, we introduce *ordinals*. Ordinals are sets that serve two purposes. First, ordinals are well-ordered by the element-relation and represent equivalence classes of well-ordered sets: For every well-ordered set, there is exactly one isomorphic ordinal. Second, we can regard ordinals as a generalisation of natural numbers that allows us to count beyond infinities: There is a zero and a successor function and, additionally, every *set* of ordinals has a least upper bound.

There are many possible definitions of ordinals but it seems difficult to find one that expresses those properties precisely. We consider a definition that can only be formulated in a higher-order logic but appears most convenient for our purposes. For the formulation, we use the notion of transitive sets.

3.5.1 Definition. We call a set A **transitive** if for all sets x and y, whenever $x \in A$ and $y \in x$ then $y \in A$. In other words, a set is transitive if every element is also a subset.

This is a specialised version of the general notion transitivity of relations where we fix the set A.

3.5.2 Definition. We define the class Ord of ordinals inductively: An **ordinal** is a transitive set of ordinals.

The transitivity condition is exactly what makes the element-relation on the class of ordinals transitive. As a side remark, even if we didn't have the axiom of foundation, this would imply that the element relation on ordinals is well-founded.

3.5.1 Ordinals for counting

In this subsection, we consider ordinals in one of their roles. Like on the natural numbers, we provide constructors, an induction principle and a recursor. Finally, we will see that the class of ordinals is well-ordered by the element relation.

We start by establishing the constructors.

3.5.3 Lemma.

1. *The empty set is an ordinal.*
2. *The successor α^+ of an ordinal α is an ordinal.*
3. *If A is a set of ordinals then $\bigcup A$ is an ordinal.*

Proof.

1. Both conditions are trivial since there are no elements.
2. Assume that α is an ordinal. We need to show that every element of α^+ is a subset of α^+ and an ordinal. Fix such an element x. By definition of the successor, $x = \alpha$ or $x \in \alpha$. The first case is trivial. In the second case, $x \subseteq \alpha \subseteq \alpha^+$ by transitivity of α and definition of the successor. Moreover, as an element of an ordinal, x is also an ordinal.
3. Fix a set of ordinals A. We need to show that every element of $\bigcup A$ is a subset of $\bigcup A$ and an ordinal. Fix such an element x. By definition of the union, there is an ordinal $\alpha \in A$, such that $x \in \alpha$. Then $x \subseteq \alpha \subseteq \bigcup A$ by transitivity of α and definition of the union. Moreover, as an element of an ordinal, x is also an ordinal. □

Since the class of ordinals contains the empty set and is closed under the successor function, we can see by induction that it contains the encodings of all natural numbers.

The constructors that we provided, are not disjoint. To formulate a useful induction principle and obtain a useful recursor, we distinguish the ordinals that can only be constructed by the third constructor.

3.5.4 Definition. A **limit ordinal** is an ordinal that is neither the empty set nor the successor of another ordinal. We use λ as an identifier that implicitly ranges over limit ordinals.

There are versions of this definition that include the empty set as a limit ordinal. But it is sometimes more natural to treat the empty set separately.

When we work with limit ordinals then it is often helpful to know that they all have a certain form:

3.5.5 Lemma. *Every limit ordinal λ satisfies $\lambda = \bigcup_{\alpha\in\lambda} \alpha$.*

Proof. $\bigcup_{\alpha\in\lambda} \alpha \subseteq \lambda$ follows immediately from transitivity of ordinals. For the other direction, we fix a $\beta \in \lambda$. It suffices to show that $\beta^+ \in \lambda$ since this implies $\beta \in \beta^+ \in \bigcup_{\alpha\in\lambda} \alpha$. We assume that $\beta^+ \notin \lambda$ and derive a contradiction. By trichotomy, we have $\beta^+ = \lambda$ or $\lambda \in \beta^+$. The first case contradicts the fact that λ is a limit. But in the second case, we have $\lambda \in \beta^+ = \beta \cup \{\beta\}$. Either $\lambda \in \beta$ or $\lambda = \beta$ which both contradicts $\beta \in \lambda$. □

3.5.6 Lemma (Transfinite induction)**.** *Fix a predicate $P : \mathsf{Ord} \to \mathbb{P}$ with the following properties.*

- *The empty set satisfies P.*
- *If any ordinal α satisfies P then the successor α^+ satisfies P.*
- *If all elements of a limit ordinal λ satisfy P then λ satisfies P.*

Then every ordinal satisfies P.

Proof. By well-founded induction on the element relation and the fact that every ordinal is either empty, a successor or a limit. □

Following the same structure as the induction principle, we introduce a recursor.

3.5.7 Lemma (Transfinite recursion)**.** *Fix a family of types $A : \mathsf{Ord} \to \mathsf{Type}$ and*

- *an element $x : A(\emptyset)$,*
- *a function $f : \Pi_\alpha A(\alpha) \to A(\alpha^+)$ and*
- *a function $g : \Pi_\lambda(\Pi_{\alpha:\lambda} A(\alpha)) \to A(\lambda)$ where λ ranges over limit ordinals.*

There is a function $\mathsf{rec}_{\mathsf{Ord}}(A, x, f, g) : \Pi_\alpha A(\alpha)$ that satisfies the equations

- $\mathsf{rec}_{\mathsf{Ord}}(A, x, f, g)(\emptyset) = x$,
- $\mathsf{rec}_{\mathsf{Ord}}(A, x, f, g)(\alpha^+) = f_\alpha(\mathsf{rec}_{\mathsf{Ord}}(A, x, f, g)(\alpha))$ *for all ordinals α and*
- $\mathsf{rec}_{\mathsf{Ord}}(A, x, f, g)(\lambda) = g_\lambda(\lambda\alpha.\, \mathsf{rec}_{\mathsf{Ord}}(A, x, f, g)(\alpha))$ *for all limit ordinals λ.*

Proof. The construction of the recursor can be done by well-founded recursion (Lemma 2.2.5) and is analogous to the proof of the transfinite induction principle. □

Usually, we will not give x, f and g explicitly when we apply transfinite recursion. Instead, we will just specify the equations that we want.

3.5.8 Lemma. *The class of ordinals is well-ordered by the element relation.*

Proof. Transitivity follows from transitivity of ordinals as sets.

To show trichotomy, we fix two ordinals α and β. First, note that $a \in \beta$, $\alpha = \beta$ and $\beta \in \alpha$ are mutually exclusive by the axiom of foundation. Hence, suffices to show that one of them actually holds. To this end, we apply well-founded induction on both, α and β. If $\alpha = \beta$, we are done. Otherwise, we know that $\alpha \nsubseteq \beta$ or $\beta \nsubseteq \alpha$. Assume without loss of generality that the first holds and there is a $\xi \in \alpha$ that is not in β. With the inductive hypothesis on α, we know that $\beta = \xi$ or $\beta \in \xi$. In the first case, $\beta \in \alpha$ holds trivially. In the second case, it follows that $\beta \in x \subseteq \alpha$ with transitivity of α.

Well-foundedness of the element-relation on the class of ordinals follows from the axiom of foundation. □

An interesting property of ordinals is that the element-relation and strict inclusion coincide:

3.5.9 Lemma. *Fix ordinals α and β. We have $\alpha \in \beta \Leftrightarrow \alpha \subsetneq \beta$.*

Proof. Assume that $\alpha \in \beta$. Then $\alpha \subseteq \beta$ by transitivity of ordinals and $\alpha \neq \beta$ since $\alpha = \beta$ would imply the contradiction $\alpha \in \alpha$.

For the other direction, assume that $\alpha \subsetneq \beta$ and that $\alpha \notin \beta$. By trichotomy of the element relation on ordinals, we can have $\alpha = \beta$ (which contradicts $\alpha \subsetneq \beta$) or $\beta \in \alpha$ (which leads to the contradiction $\beta \in \alpha \subseteq \beta$). □

3.5.2 Ordinals as well-orders

As announced before, ordinals represent well-orders:

3.5.10 Lemma. *Every ordinal is well-ordered by the element relation.*

Proof. The class Ord of all ordinals is well-ordered by the element-relation (Lemma 3.5.8). Every ordinal is a subclass of Ord and, hence, also well ordered by the element-relation. □

3.5.11 Lemma. *Isomorphic ordinals are equal.*

Proof. Fix two ordinals α and β with an isomorphism $f : \alpha \xrightarrow{\simeq} \beta$. We apply $\in$-induction on both. We need to show that $\alpha \subseteq \beta$ and $\beta \subseteq \alpha$. Without loss of generality, we focus on the first direction. Fix a $\xi \in \alpha$. It suffices to show that $\xi \simeq f(\xi)$ since, by the inductive hypothesis on x, this implies $\xi = f(\xi) \in \beta$.

The isomorphism is given by the restriction $f|_\xi : \xi \to \beta$. This is actually a function $\xi \to f(\xi)$: For all $x \in \xi$, we have $f(x) \in f(\xi)$ by the morphism property of f. As the inverse, we have $f^{-1}|_{f(\xi)}$. The function $f|_\xi$ is still a morphism since it is the restriction of a morphism. □

One can also show that every well-ordered set has an isomorphic ordinal, its **order type**. But we do not need that fact and will not prove it here.

3.6 GCH, AC and the well-ordering theorem

We end the chapter with the formal statements of the generalised continuum hypothesis (GCH), the axiom of choice (AC) and the well-ordering theorem.

3.6.1 Definition. The **generalised continuum hypothesis** states that, for all infinite sets A and B such that $A \leq B \leq \mathcal{P}(A)$, we have either $A \sim B$ or $B \sim \mathcal{P}(A)$.

In other words, there is no cardinality strictly between that of a set A an that of the power set $\mathcal{P}(A)$. The (specialised) continuum hypothesis, in contrast, talks only about $\mathbb{N}$ in place of arbitrary sets A.

3.6.2 Definition. The **axiom of choice** states that for every family $F : I \to \mathscr{S}$ of inhabited sets over an index set I, there is propositionally a choice function that maps every $i \in I$ to an element of $F(i)$.

3.6.3 Definition. The **well-ordering theorem** states that every set has a well-order.

We consider the well-ordering theorem because of its equivalence to AC but we need only one direction of this equivalence.

3.6.4 Theorem. *The well-ordering theorem implies the axiom of choice.*

Proof. Let $F : I \to \mathscr{S}$ be a family over a set I. Fix a well-order on $\bigcup_{i:I} F(i)$. By the Axiom of Definite Description, there is a function that maps every $i : I$ to the unique least element of $F(i)$. □

4 Sierpiński's theorem

In this chapter, we will see that the generalised continuum hypothesis implies the well-ordering theorem and, with it, the axiom of choice. The result is called Sierpiński's theorem after Wacław Sierpiński who proved it in 1947 [25]. The other direction of the implication does not hold; the axiom of choice does not generally imply the generalised continuum hypothesis. But that is outside the scope of this thesis. Our presentation of Sierpiński's theorem takes inspiration from a version by Gillman [8].

4.1 The Hartogs numbers

Our goal for this section is to assign to every set a special ordinal, its so-called *Hartogs number*, named after its inventor Friedrich Hartogs [14]. If we assumed stronger axioms, such as the generalised continuum hypothesis or just the axiom of choice, then we could show that the Hartogs number of a set is always greater than that set. In fact, that is exactly how the proof of Sierpiński's theorem in the next section works. But without further assumption, the closest thing that we can achieve is to show that the Hartogs number of a set is at least not smaller or equal in cardinality. As a by-product of the construction, we will obtain an upper bound on the Hartogs number that will be crucial to prove Sierpiński's theorem.

4.1.1 Definition. We define the **Hartogs number** of a given set A as

$$\aleph(A) \coloneqq \{\alpha \in \mathsf{Ord} \mid \alpha \leq A\}.$$

If it turns out that the Hartogs number is an ordinal then $\aleph(A) \not\leq A$ will follow immediately from this definition because otherwise, the Hartogs number would contain itself. We proceed in three steps:

1. We show that $\aleph(A) \leq \mathcal{P}^6(A)$, which implies that the Hartogs number is a set (Lemma 4.1.3).

2. We show that the Hartogs number is an ordinal.

3. We conclude that $\aleph(A) \not\leq A$.

4.1.2 Fact. *The Hartogs number of every set A satisfies the upper bound*

$$\aleph(A) \leq \mathcal{P}^6(A).$$

Proof. By the general upper bound on the Cartesian product (Lemma 3.4.5),

$$\begin{aligned}\mathcal{P}\left(\mathcal{P}(A) \times \mathcal{P}(A \times A)\right) &\leq \mathcal{P}\left(\mathcal{P}(A) \times \mathcal{P}^3(A)\right) \\ &\leq \mathcal{P}\left(\mathcal{P}^3(A) \times \mathcal{P}^3(A)\right) \\ &\leq \mathcal{P}^6(A).\end{aligned}$$

By transitivity, it suffices to define the injection

$$\begin{aligned}f : \aleph(A) &\hookrightarrow \mathcal{P}(\mathcal{P}(A) \times \mathcal{P}(A \times A)) \\ f(\alpha) &\coloneqq \{x \in \mathcal{P}(A) \times \mathcal{P}(A \times A) \mid x \simeq \alpha\},\end{aligned}$$

where we treat every $x \in \mathcal{P}(A) \times \mathcal{P}(A \times A)$ as a subset of A with a relation on it that can satisfy $x \simeq \alpha$ if the relation is an order. To see that f is injective, fix two ordinals $\alpha, \beta \in \aleph(A)$ with $f(\alpha) = f(\beta)$. By definition of the Hartogs number, there is an injection $\alpha \hookrightarrow A$. We embed the order on α along this injection to obtain an $x \in \mathcal{P}(A) \times \mathcal{P}(A \times A)$. Note that $x \simeq \alpha$. Therefore $x \in f(\alpha) = f(\beta)$ and hence, $x \simeq \beta$ by definition of f. Together, we have $\alpha \simeq x \simeq \beta$ which implies $\alpha = \beta$ since isomorphic ordinals are equal (Lemma 3.5.11). □

As mentioned before, this upper bound is very generous. We could use a different encoding of ordered subsets to get the bound down to $\mathcal{P}^3(A)$. For our purpose, however, this does not matter.

Now , we show that the Hartogs number is a set. We do this through the following lemma.

4.1.3 Lemma. *Every class that is smaller than a set is itself also a set.*

Proof. Fix an arbitrary class A and a set B that is larger or equal to A. By definition, we have an injection $f : A \hookrightarrow B$. The class A is the image of f^{-1} and thus a set by the axiom of replacement (3.2.2). □

4.1.4 Corollary. *The Hartogs number of a set is also a set.*

Proof. This follows from the upper bound $\aleph(A) \leq \mathcal{P}^6(A)$ with the previous lemma. □

4.1.5 Fact. *The Hartogs number of a set is an ordinal.*

Proof. Fix a set A. We know that the Hartogs number $\aleph(A)$ contains only ordinals by definition and that it is a set by the previous corollary. It suffices to show that the Hartogs number is transitive. Fix two ordinals α and β with $\beta \in \alpha \in \aleph(A)$. Our goal is to prove that $\beta \in \aleph(A)$. By definition of the Hartogs number, $\alpha \in \aleph(A)$ is an ordinal that satisfies $\alpha \leq A$ and we have to show those properties for β.

1. As element of an ordinal, β is one too.
2. As element of an ordinal, β is also a subset. Hence, $\beta \leq \alpha \leq \aleph(A)$. □

4.1.6 Hartogs' Theorem. *For all sets A, we have $\aleph(A) \not\leq A$.*

Proof. Assume that $\aleph(A) \leq A$. By definition, we get that $\aleph(A) \in \aleph(A)$, which is a contradiction (Lemma 3.1.3). □

4.2 Sierpiński's theorem

This section focuses on proof of our main theorem and some fact about cardinality that contribute immediately to that proof. Our approach is an adaption of that from Gillman [8] which in turn is a modified form of Sierpiński's original formulation [25].

We will provide a condition under which a set A satisfies the equation $A \sim A + A$. Under the axiom of choice, this equivalence holds for all infinite sets. But without the axiom, we can only do the proof for *power sets* of infinite sets. Actually, the result holds for types in general, not just for sets. However, the proof for sets is a little easier to write down.

As an intermediate result, we want to show that every infinite set A satisfies $A \sim 1 + A$. First, we show this property for the type of natural numbers; then, we use the fact that every infinite set is equipotent to a sum of the form $\mathbb{N} + \_$ to generalise the result.

4.2.1 Lemma. $\mathbb{N} \sim 1 + \mathbb{N}$.

Proof. We define functions in both directions by

$$\begin{aligned} f : \mathbb{N} &\to 1 + \mathbb{N} & g : 1 + \mathbb{N} &\to \mathbb{N} \\ f(0) &\coloneqq \mathsf{inj}_1(0) & g(\mathsf{inj}_1(\_)) &\coloneqq 0 \\ f(n+1) &\coloneqq \mathsf{inj}_2(n) & g(\mathsf{inj}_2(n)) &\coloneqq n + 1. \end{aligned}$$

By case analysis, we can easily show that those are mutually inverse. □

4.2.2 Lemma. *If $B \subseteq A$ then $A \sim B + (A \setminus B)$.*

Proof. The claim follows by $A \sim B \cup (A \setminus B) \sim B + (A \setminus B)$. The first step is trivial with the definition of union and set difference. The second step works since B and $A \setminus B$ are disjoint. □

4.2.3 Lemma. *If A is an infinite set then $A \sim 1 + A$.*

Proof. Let $B \subseteq A$ be the image of the injection from $\mathbb{N}$ to A. Then $B \sim \mathbb{N}$ which implies $B \sim 1 + B$ (Lemma 4.2.1). We conclude

$$\begin{aligned} A &\sim B + (A \setminus B) \\ &\sim (1 + B) + (A \setminus B) \\ &\sim 1 + (B + (A \setminus B)) \\ &\sim 1 + A \end{aligned}$$

by the previous lemma and associativity of the disjoint union with respect to cardinality. □

With this, we can prove the desired result now.

Lemma. *If A is an infinite set then $\mathcal{P}(A) \sim \mathcal{P}(A) + \mathcal{P}(A)$.*

Proof. We deduce

$$\begin{aligned}\mathcal{P}(A) &\sim 2^A \\ &\sim 2^{1+A} \\ &\sim 2^1 \times 2^A \\ &\sim 2 \times 2^A \\ &\sim 2^A + 2^A \\ &\sim \mathcal{P}(A) + \mathcal{P}(A)\end{aligned}$$

with Lemma 4.2.3 and Fact 2.1.3. □

We continue with a rather technical but powerful lemma.

4.2.4 Lemma. *All sets A and B such that $A \sim A + A$ and $A + B \sim \mathcal{P}(A)$ satisfy the inequality $\mathcal{P}(A) \leq B$ (and, hence, $\mathcal{P}(A) \sim B$).*

Proof. Without loss of generality, we assume that A and B are disjoint. From the assumptions and Fact 2.1.3, we deduce

$$A \cup B \sim A + B \sim \mathcal{P}(A) \sim 2^A \sim 2^{A+A} \sim 2^A \times 2^A.$$

Hence, there is a bijection $f : A \cup B \xrightarrow{\sim} 2^A \times 2^A$. We compose f with the first projection to obtain a function $\pi_1 \circ f : A \to 2^A$. By Cantor's theorem (2.1.9), this cannot be a surjection, that is, there is an $A_1 \in 2^A$ that is not the first component of any $f(x)$ for $x \in A$. Conversely, for all $A_2 \in 2^A$, the preimage $f^{-1}(A_1,A_2)$ must come from B. This leads us to the conclusion that the injection

$$\begin{aligned}2^A &\hookrightarrow A \cup B \\ A_2 &\mapsto f^{-1}(A_1, A_2)\end{aligned}$$

is actually an injection of type $2^A \hookrightarrow B$. Hence, $\mathcal{P}(A) \sim 2^A \leq B$. □

Under the axiom of choice, we could replace the condition that $A \sim A + A$ by the condition that A has to be infinite. Without the axiom of choice however, that does not seem to be strong enough. The lemma could be proved more generally for types and power types but, again, the special case of sets is a bit easier to write down.

Now, we have everything that we need to prove Sierpiński's theorem.

4.2.5 Sierpiński's Theorem. *The generalised continuum hypothesis implies the axiom of choice.*

Proof. We assume the generalised continuum hypothesis and show the well-ordering theorem which implies the axiom of choice (Theorem 3.6.4). Let us fix a set A that we want to well-order. It suffices to find some ordinal greater than or equal to A.

Since the Hartogs number of A is at least not smaller, it seems like a reasonable candidate. However, it will be useful to consider instead the Hartogs number of some possibly larger set B with the property that $\mathcal{P}^i(B)$ is infinite and satisfies

$$\mathcal{P}^i(B) \sim \mathcal{P}^i(B) + \mathcal{P}^i(B) \tag{4.1}$$

for all natural numbers i. By the previous lemma and the definition of infinity (2.1.7), $B \coloneqq \mathcal{P}(A + \mathbb{N})$ satisfies this property.

We know that $\mathcal{P}^6(B)$ is an upper bound on the Hartogs number of B (Fact 4.1.2). We show by induction that, in general, for all natural numbers i, the upper bound $\aleph(B) \leq \mathcal{P}^i(B)$ implies the existence of a well-order on A. The base case is simple since the assumption that $\aleph(B) \leq \mathcal{P}^0(B) = B$, contradicts Hartogs' theorem (4.1.6).

For the inductive step, we fix a natural number i and assume the upper bound $\aleph(B) \leq \mathcal{P}^{i+1}(B)$. We need an occasion to apply the generalised continuum hypothesis, so we deduce

$$\mathcal{P}^i(B) \leq \aleph(B) + \mathcal{P}^i(B) \leq \mathcal{P}^{i+1}(B),$$

where the first inequality is trivial and the second one follows with Equation 4.1 as

$$\begin{aligned} \aleph(B) + \mathcal{P}^i(B) &\leq \mathcal{P}^{i+1}(B) + \mathcal{P}^i(B) \\ &\leq \mathcal{P}^{i+1}(B) + \mathcal{P}^{i+1}(B) \\ &\leq \mathcal{P}^{i+1}(B). \end{aligned}$$

By the generalised continuum hypothesis, this leads to two possible cases:

Case 1. If $\aleph(B) + \mathcal{P}^i(B) \sim \mathcal{P}^i(B)$ then we can immediately conclude that $\aleph(B) \leq \mathcal{P}^i(B)$. This is exactly the condition for the inductive hypothesis which proves our goal.

Case 2. If $\aleph(B) + \mathcal{P}^{i(B)} \sim \mathcal{P}^{i+1}(B)$ then $\mathcal{P}^{i+1}(B) \leq \aleph(B)$ by Lemma 4.2.4 with Equation 4.1. Hence, $A \leq B \leq \mathcal{P}^{i+1}(B) \leq \aleph(B)$ and since the Hartogs number is an ordinal and therefore has a well-order, we conclude that A has a well-order too. □

5 First-order set theory

In this chapter, we introduce our second set theory, ZF'. It is very close to the first-order formulation of ZF that is widely accepted as the standard foundation for mathematics. We compare the two theories in more detail in Section 5.1. We build on the notion of ZF structures from Chapter 3. We also reuse the core axioms from Section 3.1. What distinguishes ZF' from ZF_2, are the precise formulation of the axioms of separation/replacement. In the case of ZF_2, those axioms take a set and a predicate/function and postulate the existence of a new set based on those (Axiom 3.2.1 and Axiom 3.2.2). This formulation is too strong for the next chapter where we will construct models of set theory. The problem is that we do not really know how many predicates and functions there are. Hence, we would not know how many sets we need to include in the model in order to satisfy the axioms of separation an replacement. For ZF', we state a weaker version of those axioms that accept only predicates and functions represented by first-order formulas. This gives us more freedom for the model construction.

We will formally introduce our syntax of first-order logic now. A notable detail is that we use a syntax that allows us to use arbitrary sets as constant terms. Thus, we fix a ZF structure $\mathscr{S}$ now and define the syntax depending on this structure.

5.0.1 Definition. We define the abstract syntax for first-order **terms** and first-order **formulas** by the rules

$$\begin{aligned} t &::= x \mid a && x : \mathbb{N}, a : \mathscr{S} \\ \varphi &::= \dot\exists\varphi \mid \varphi_1 \mathbin{\dot\wedge} \varphi_2 \mid \dot\neg\varphi \mid t_1 \doteq t_2 \mid t_1 \mathbin{\dot\in} t_2. \end{aligned}$$

The natural number x represents a variable in de Bruijn notation [5], that means it indicates the levels of $\exists$-quantifiers between the occurrence of x and the quantifier that introduces the variable. If x is greater than the number of such levels then it stands for a free variable. By Term_n, we denote the type of first-order terms with n free variables and by $\mathsf{Formula}_n$ the type of first-order formulas with n free variables. We use the operator precedences that are common in logical notation.

Now, we define the semantics of terms. Terms with free variables can only be evaluated if we have an environment which tells us how the variables are assigned. We model such environments as members of $\mathscr{S}^n$ where n is the number of free variables. We define the evaluation function for terms as

$$\begin{aligned} [\![\_]\!]_\_ &: \mathsf{Term}_n \to \mathscr{S}^n \to \mathscr{S} \\ [\![x]\!]_\gamma &:= \gamma[x] \\ [\![a]\!]_\gamma &:= a. \end{aligned}$$

Here, $\gamma[x]$ stands for the component of γ at position x and n can be seen as an implicit argument of the function.

Now, we define the satisfaction relation that describes when a formula is satisfied by an environment.

$$\begin{aligned}
&_ \vDash _ && : \mathscr{S}^n \to \mathsf{Formula}_n \to \mathbb{P} \\
&\gamma \vDash \exists\varphi && :\Leftrightarrow \text{There is an } a, \text{ such that } \gamma, a \vDash \varphi \\
&\gamma \vDash \varphi_1 \wedge \varphi_2 && :\Leftrightarrow \gamma \vDash \varphi_1 \text{ and } \gamma \vDash \varphi_2 \\
&\gamma \vDash \neg\varphi && :\Leftrightarrow \gamma \nvDash \varphi \\
&\gamma \vDash t_1 = t_2 && :\Leftrightarrow (\llbracket t_1 \rrbracket_\gamma = \llbracket t_2 \rrbracket_\gamma) \\
&\gamma \vDash t_1 \in t_2 && :\Leftrightarrow (\llbracket t_1 \rrbracket_\gamma \in \llbracket t_2 \rrbracket_\gamma).
\end{aligned}$$

We say that an environment γ satisfies a formula φ if $\gamma \vDash \varphi$. We say that a formula without free variables is true if it is satisfied by the empty environment ( ). This is all that we need to formulate the first-order versions of separation and replacement.

5.0.2 Axiom of First-Order Separation. *For every set A and formula φ with one free variable, the class*

$$\{x \in A \mid (x) \vDash \varphi\}$$

is a set.

This axiom is not expressible in first-order logic as it quantifies over formulas. In first-order logic, separation is typically handled as an axiom scheme, an infinite set with one axiom for every formula φ. The same holds for the next axiom.

5.0.3 Axiom of First-Order Replacement. *For every set A and formula φ with two free variables, the class*

$$\{y \mid \exists y : A.\, (x, y) \vDash \varphi\}$$

is a set under the condition that the relation $\lambda x, y.\, (x, y) \vDash \varphi$ is functional.

Both axioms are analogous to those in ZF_2, except that separation expects a formula instead of an arbitrary predicate and replacement expects a formula describing a functional relation instead of a function.

5.1 Comparison to standard first-order formulation

In this section, we point out the differences between ZF' and the standard first-order axiomatisation of ZF. The obvious first difference is that our axioms are not formulated in first-order logic. But this is just a cosmetic difference because it does no rule out semantic equivalence to first-order formulations. This is the case for all axioms except the axiom of foundation and the axiom of infinity.

In most cases, this equivalence is trivial to check but in the case of separation and replacement, the difference is a bit greater since standard first-order logic does not have

constants in our sense. We will look at separation as an example. For readability, we take some freedom with the notation here and do not use the first-order syntax that we defined before. As mentioned earlier, we need an infinite axiom scheme to describe separation in first-order logic. For every natural number n and every constant-free formula φ with $n+1$ free variables, one would typically have an axiom similar to

$$\forall c_1, \dots, c_n. \forall A. \exists B. (\forall x. x \in B \Leftrightarrow x \in A \wedge \varphi(c_1, \dots, c_n, x)).$$

The c_i are used to simulate constants in φ. In order to prove this axiom scheme from our axiom of replacement, one would introduce the c_i and A and use the set $B \coloneqq \{x \in A \mid (x) \vDash \varphi(c_1, \dots, c_n)\}$ where $\varphi(c_1, \dots, c_n)$ stands for φ with the c_i inserted as constants in place of the first n free variables.

In the other direction, we assume this axiom scheme and derive our axiom of separation: Given a formula ψ with constants, we can extract the constants $c_1, \dots, c_n$ and replace them by free variables in ψ to obtain a constant-free formula φ. Then we can apply the instance of the axiom scheme for ψ to the constants c_i and obtain a set that satisfies the condition of separation. The first-order formulation of replacement is more complex because it contains a functionality condition. But it simulates constants in the same way and the equivalence with our axiom can be established in the same way.

The axiom of foundation is typically formulated as

$$\forall x. x \neq \emptyset \Rightarrow \exists y \in x. x \cap y = \emptyset.$$

In other words, every set has a least element with respect to the element relation. This is a kind of well-foundedness but one that is internal to the set theory. It does not imply well-foundedness in the external sense that we assume in the axiom of foundation. The restriction that we only have least elements for *sets* and not for arbitrary classes makes the statement weaker. If we try to derive an induction principle from this statement then we get it only for representable predicates.

To prove that a pure first-order axiomatisation cannot imply our strong axiom of foundation, we can add infinitely many constant symbols $a_0, a_1, a_2, \dots$ to the theory and infinitely many axiom $a_0 \in a_1$, $a_1 \in a_2$, ... If the original axiomatisation is consistent then it is consistent with every finite subset of those additional axioms and by the compactness theorem, there is a model that satisfies them all simultaneously. But together, they imply the existence of an externally non-well-founded set in that model.

The reason why we assume the stronger foundation axiom is convenience. We do not need to prove representability before we apply well-founded induction and, it allows us to derive a stronger well-ordering theorem in the next chapter and, abstractly speaking, it brings the set theory and the theory of Coq closer together.

Assuming that we do not have the strong axiom of foundation, the case of the axiom of infinity is similar. We define the class $\bar{\mathbb{N}}$ and assume that this class is a set. In first-order logic, one cannot express this. Instead, one would typically assume that there is a set that contains the empty set and is closed under the successor function. One can then define the set of numerals as the smallest set with that property. This set will certainly include $\bar{\mathbb{N}}$ but it might be strictly greater. Similar to above, one can prove by the compactness

theorem that there is a model with a numeral that contains infinitely many smaller numerals. That one would not be in $\bar{\mathbb{N}}$. One can even prove the existence of a model with uncountably many numerals. In that case, we do not even get a bijection between the set of numerals and $\mathbb{N}$.

The lack of such a bijection would make our notion of encodings much less useful since we could not define representable encodings for many interesting infinite types like $\mathbb{N}$ or the type of formulas. That is the strongest argument why we do not work with the pure first-order axiomatisation. One could likely relax the notion of encodings in a reasonable way which provides the same benefits but works in the first-order axiomatisation. However, that would certainly add complexity to the foundation.

As a final remark, under the strong axiom of foundation, our formulation of the infinity axiom is equivalent to the one in first order logic. One could use well-founded induction on the element relation to prove from the first-order version of the infinity axiom that every numeral is an element of $\bar{\mathbb{N}}$. This means that the only distinction between our axiomatisation and one in first-order logic is the additional assumption that the model is well-founded in an external sense. But that is not such a big difference. Given a model of the first-order axiomatisation, one can cut out the well-founded part and check that it still satisfies all the axioms and additionally the strong axiom of foundation. In the other direction, every model of ZF' is trivially a model of the first-order axiomatisation.

5.2 Evolution of first-order representations

To use first-order replacement and separation, we need to create representations of predicates and functions in first-order logic. This is not always a trivial task. For example, we may want to prove by separation that a class of the form

$$\{x \in A \mid B \cap x \in C\}$$

is a set. We cannot literally translate the defining condition of this set into a first-order formula since it contains the composed term $x \cap B$ that is neither a variable nor a constant and, hence, cannot be expressed in a first-order term. The simplest possible first-order representation of this condition seems to be

$$\exists y.\, (\forall z.\, z \in y \Leftrightarrow z \in B \wedge z \in x) \wedge z \in C.$$

Of course, we would still need to express this condition in our formal first-order syntax with de Bruijn notation before we can apply the axiom of separation. If we want to produce such first-order representations for more complex predicates or functions then we should do that in a systematic way. to handle the inherent complexity.

To this end, we introduce a powerful reification framework for first-order logic. It allows us to translate Coq predicates to first-order formulas in a compositional way, following the term-structure. On the example from above, this compositionality means that we can build a first-order representation of the predicate $\lambda x.\, B \cap x \in C$ from representations of B, C, $\cap$ and $\in$. This way, we can easily structure the construction

of representations. Moreover, representations can be constructed automatically if we already have representations of all atomic subterms.

We want to make the framework powerful enough such that one does not have to make any compromises in the formulation of predicates that one wants to reify. It should be possible to formulate predicates in the most natural form and the reification framework should be flexible enough to handle them. To achieve this goal, we will have to handle subterms of almost arbitrary types, not just predicates or functions over sets. Among others, our framework will allow us to represent higher-order functions, dependent functions and quantification over types other than $\mathscr{S}$. Moreover, one can dynamically extend the framework to new types if necessary. This fits well with the philosophy of the calculus of inductive constructions where inductive definitions, for example, introduce new types to the language.

The core of our reification framework contains multiple different ideas. To introduce them one-by-one, we need to build the framework in multiple iteration steps. Each of the following subsection presents a new version that extends the capabilities of the previous one. The complete final framework is presented in the next section.

5.2.1 Basic idea (Version 1)

Let us consider the example from above again. If we want to construct a representation of the function $\lambda x.B \cap x \in C$ in a compositional way, that means we have to build it from a representation of $B \cap x \in C$. But this term has a free variable x. In order to work under binders, we need to talk about representations of terms with free variables and we need a way to model such terms. We will model terms with n free $\mathscr{S}$-typed variables as families of a type $\mathscr{S}^n \to \ldots$ where first argument stands for an **environment** that provides the values for the variables. For example, we model the term $x \cap B \in C$ as the family $(B \cap \gamma[1] \in C)_{\gamma:\mathscr{S}^1}$ and the term $\lambda x.\, B \cap x \in C$ as the constant family $(\lambda x.\, B \cap x \in C)_{\gamma:\mathscr{S}^0}$.

5.2.1 Definition (Provisional)**.** A representation of a family of **relations** $(P_\gamma : \mathscr{S}^k \to \mathbb{P})_{\gamma:\mathscr{S}^n}$ is a first-order formula φ, such that, for all sets $x_1, \ldots, x_k$ and environments $\gamma : \mathscr{S}^n$, we have $((\gamma, x_1, \ldots, x_k) \vDash \varphi) \Leftrightarrow P_\gamma(x_1, \ldots, x_k)$.

A representation of a family of **functions** $(f_\gamma : \mathscr{S}^k \to \mathscr{S})_{\gamma:\mathscr{S}^n}$ is a representation of

$$(\lambda x_1, \ldots, x_k, z.\, f_\gamma(x_1, \ldots, x_k) = z)_{\gamma:\mathscr{S}^n}.$$

A family of **sets** is representable if it is representable as a family of functions with zero arguments.

We say that a relation of type $\mathscr{S}^k \to \mathbb{P}$ or a function of type $\mathscr{S}^k \to \mathscr{S}$ is representable if it is representable as a constant family over the environment type $\mathscr{S}^0$.

By this definition, every representable predicate of type $\mathscr{S} \to \mathbb{P}$ comes with a formula that we can use for separation and every representable function of type $\mathscr{S} \to \mathscr{S}$ comes with a formula that we can use for replacement. Moreover, we can construct representations in a compositional way. The following lemma allows composition for application of relations:

5.2.2 Lemma. *Fix a representable family of relations* $(P_\gamma : \mathscr{S}^{k+1} \to \mathbb{P})_{\gamma:\mathscr{S}^n}$ *and a representable family of sets* $(a_\gamma : \mathscr{S})_{\gamma:\mathscr{S}^n}$*. Then the family of results* $(P_\gamma(a_\gamma) : \mathscr{S}^k \to \mathbb{P})_{\gamma:\mathscr{S}^n}$ *is also representable.*

Proof. The representation of P is a formula φ_P, such that $((\gamma, x_1, \ldots, x_{k+1}) \vDash \varphi_P) \Leftrightarrow P_\gamma(x_1, \ldots, x_{k+1})$ for all γ and $x_1, \ldots, x_{k+1}$. The representation of a is a formula φ_a, such that $((\gamma, y) \vDash \varphi_a) \Leftrightarrow y = a_\gamma$ for all γ and y. We can substitute the free variables in φ_a to obtain a formula φ'_a, such that $((\gamma, x_1, \ldots, x_k, y) \vDash \varphi'_a) \Leftrightarrow y = a_\gamma$. The formula $\varphi_{P(a)} \coloneqq \dot{\exists}\varphi'_a \mathbin{\dot{\wedge}} \varphi_P$ is the representation that we are looking for since it satisfies

$$\begin{aligned}
&((\gamma, x_1, \ldots, x_k) \vDash \varphi_{P(a)}) \\
&\quad \Leftrightarrow (\exists y.\, ((\gamma, x_1, \ldots, x_k, y) \vDash \varphi'_a) \wedge ((\gamma, x_1, \ldots, x_k, \gamma) \vDash \varphi_P)) \\
&\quad \Leftrightarrow (\exists y.\, y = a \wedge P_\gamma(x_1, \ldots, x_k, y)) \\
&\quad \Leftrightarrow P_\gamma(x_1, \ldots, x_k, a).
\end{aligned}$$

□

For compositional proofs on lambdas, we need the next lemma.

5.2.3 Lemma. *A family of relations* $(P_\gamma : \mathscr{S}^{k+1} \to \mathbb{P})_{\gamma:\mathscr{S}^n}$ *is representable if the family of partial applications* $(P_\gamma(x) : \mathscr{S}^k \to \mathscr{S})_{(\gamma,x):\mathscr{S}^{n+1}}$ *is representable.*

Proof. This holds trivially by definition. □

Analogous lemmas hold for functions. Together, those four lemmas suffice to decompose the example from above. But they do not tell us what to do with logical operators. We need additional lemmas for that.

5.2.4 Lemma. *Fix two representable families of predicates* $(P_\gamma : \mathscr{S} \to \mathbb{P})_{\gamma:\mathscr{S}^n}$ *and* $(Q_\gamma : \mathscr{S} \to \mathbb{P})_{\gamma:\mathscr{S}^n}$*. The following families are representable:*

$$(\neg P_\gamma)_{\gamma:\mathscr{S}^n} \quad (P_\gamma \wedge Q_\gamma)_{\gamma:\mathscr{S}^n} \quad (P_\gamma \vee Q_\gamma)_{\gamma:\mathscr{S}^n} \quad (P_\gamma \Rightarrow Q_\gamma)_{\gamma:\mathscr{S}^n} \quad (P_\gamma \Leftrightarrow Q_\gamma)_{\gamma:\mathscr{S}^n}$$

The proof is easy. We will not present it here.

5.2.5 Lemma. *Fix a representable family of predicated* $(P_\gamma : \mathscr{S} \to \mathbb{P})_{\gamma:\mathscr{S}^n}$*. The family of existential statements* $(\exists x : \mathscr{S}.\, P_\gamma(x))_{\gamma:\mathscr{S}^n}$ *and the family of universal statements* $(\forall x : \mathscr{S}.\, P_\gamma(x))_{\gamma:\mathscr{S}^n}$ *are representable.*

Proof. Let φ_P be the formula representing P. Then $\dot{\exists}\varphi_P$ represents $(\exists x.\, P_\gamma(x))_{\gamma:\mathscr{S}^n}$ and $\dot{\neg}\dot{\forall}\dot{\neg}\varphi_P$ represents $(\forall x.\, P_\gamma(x))_{\gamma:\mathscr{S}^n}$. □

Now we have various lemmas that allow us to compose representations from representations of subterms. But we still don't know how to handle atomic subterms. The next two lemmas will change that. We will not provide proofs.

5.2.6 Lemma. *Fix two natural numbers* n *and* k *with* $k < n$*. The family* $(\gamma[k] : \mathscr{S})_{\gamma:\mathscr{S}^n}$ *is representable.*

Proof. The representation is given by the formula $0 \doteq k + 1$. □

5.2.7 Lemma. *Fix a natural number n and a set c. The constant family $(c : \mathscr{S})_{\gamma:\mathscr{S}^n}$ is representable.*

Proof. The representation is given by the formula $0 \doteq c$. □

5.2.8 Lemma. *Fix a natural number n. The constant families $(\bot : \mathbb{P})_{\gamma:\mathscr{S}^n}$ and $(\top : \mathbb{P})_{\gamma:\mathscr{S}^n}$ are representable.*

Proof. The first representation is given by $\dot{\exists}\dot{\neg}(0 \doteq 0)$. The second one is given by $\dot{\forall} 0 \doteq 0$. □

5.2.9 Lemma. *Fix a natural number n. The constant families $(\_ \in \_ : \mathscr{S} \to \mathscr{S} \to \mathbb{P})_{\gamma:\mathscr{S}^n}$ and $(\_ = \_ : \mathscr{S} \to \mathscr{S} \to \mathbb{P})_{\gamma:\mathscr{S}^n}$ are representable.*

Proof. The first representation is given by $1 \dot{\in} 0$ and the second one by $1 \doteq 0$. □

With the definitions and lemmas from this chapter, we have everything that we need for the compositional construction of first-order representation for predicates or functions on sets, as long as every subterm is also such a predicate or function.

To return to the example, we can build a representation of $(\lambda x.B \cap x \in C)_{\gamma:\mathscr{S}^0}$ from a representation of $(B \cap \gamma[0] \in C)_{\gamma:\mathscr{S}^1}$ (Lemma 5.2.5. We can build that representation from a representation of $(B \cap \gamma[0] \in \_)_{\gamma:\mathscr{S}^1}$ and a representation of $(C)_{\gamma:\mathscr{S}^1}$ (Lemma 5.2.3). We get the second one immediately by Lemma 5.2.7. In multiple steps, the first one can be composed of representations of $(B)_{\gamma:\mathscr{S}^1}$, $(\_ \cap \_)_{\gamma:\mathscr{S}^1}$, $(\gamma[0])_{\gamma:\mathscr{S}^1}$ and $(\_ \in \_)_{\gamma:\mathscr{S}^1}$. We do not have a representation for $(\_ \cap \_)_{\gamma:\mathscr{S}^1}$ yet but the other ones are given by Lemma 5.2.7, Lemma 5.2.6 and Lemma 5.2.9.

This works well in such toy examples, but the restrictions of this approach are too strong to build more serious set-theoretic constructions conveniently. Hence, the next subsection will introduce a more flexible approach.

5.2.2 Functions (Version 2)

This subsection introduces the most important ideas of our reification framework. The main motivation behind those ideas is the goal to represent higher-order functions. To do this, we redefine representability on function types in in a general way. Roughly speaking, we will call a function representable if it maps representable inputs to representable outputs. The general nature of this new notion of representability on function types allows us to extend representability to even more types in a meaningful way. For example, we will be able to formulate representability on pairs such that the pair constructor and the projections π_1 and π_2 are representable.

To take full advantage of this generality, we will make the notion of representations extensible. Every type can have its own so-called representation scheme that tells us which families in that type are representable. Before we define what exactly a representation scheme is, we need to introduce an operation that takes a family over $\mathscr{S}^n$ for some n and lifts it to a family over $\mathscr{S}^k$ for some $k \geq n$.

5.2.10 Definition (Provisional)**.** Fix a family $x : \mathscr{S}^n \to A$ and a natural number $k \geq n$. We define a family $\Uparrow_n^k x : \mathscr{S}^k \to A$ by $\Uparrow_n^k x := x \circ \mathsf{reduce}_n^k$ where reduce_n^k is the function that takes a member of $\mathscr{S}^k$ and discards the first $k - n$ components to obtain a member of $\mathscr{S}^n$. We usually write just $\Uparrow x$ and leave n and k implicit.

5.2.11 Definition (Provisional)**.** A **representation scheme** on a type A consists of a function $\mathsf{LocalRep} : \prod_{n:\mathbb{N}}(\mathscr{S}^n \to A) \to \mathsf{Type}$. This function takes a number n and a family $(x_\gamma : A)_{\gamma:\mathscr{S}^n}$ and returns the type of **local representations** of x.

Given a representation scheme on A and a family $(x_\gamma : A)_{\gamma:\mathscr{S}^n}$, a (full) **representation** of x is a family of type $\prod_{k>n} \mathsf{LocalRep}_k(\Uparrow x)$.

Given a type type A with a representation scheme and a member $x : A$, a representation of x is a representation of the constant family $(x)_{\gamma:\mathscr{S}^0}$.

Often, we will handle representations schemes implicitly to keep formulations simple. The reason why a representation has to consist of a family of local representations is that it provides us with the following lemma.

5.2.12 Lemma. *Fix a type A with a representation scheme and a family $(x_\gamma : A)_{\gamma:\mathscr{S}^n}$. If x is representable then, for every $n' \geq n$, the lifted family $\Uparrow^{n'} x$ is also representable.*

Proof. Fix an $n' \geq n$. The representation of x is a function of type $\prod_{k>n} \mathsf{LocalRep}_k(\Uparrow x)$. Since $k > n'$ implies $k > n$, we can restrict this function to one of type $\prod_{k>n'} \mathsf{LocalRep}_k(\Uparrow x)$. □

This lemma will be important when we use values under a binder that were introduced outside of the binder. Intuitively speaking, something that is fully representable in a certain context will still be fully representable if we add more free variables to that context.

Now, we define representation schemes on $\mathbb{P}$ and $\mathscr{S}$. Those should not be surprising since they are similar to the ones for predicates and functions from the previous subsection.

5.2.13 Definition (Provisional)**.**

- A local representation of a family of **propositions** $(P_\gamma : \mathbb{P})_{\gamma:\mathscr{S}^n}$ is a formula φ with n free variables, such that $(\gamma \vDash \varphi) \Leftrightarrow P_\gamma$ for all environments $\gamma : \mathscr{S}^n$.
- A local representation of a family of **sets** $(x_\gamma : \mathscr{S})_{\gamma:\mathscr{S}^n}$ is a local representation of the family of propositions $(x_\gamma = y)_{(\gamma,y):\mathscr{S}^{n+1}}$.

Next, we define a representation scheme for general function types.

5.2.14 Definition (Provisional)**.** A local representation of a family of **functions** $(f_\gamma : A_\gamma \to B_\gamma)_{\gamma:\mathscr{S}^n}$ is a function that takes a *full* representation of any family $(x_\gamma : A_\gamma)_{\gamma:\mathscr{S}^n}$ and returns a local representation of $(f_\gamma(x_\gamma))_{\gamma:\mathscr{S}^n}$.

We can immediately work with this definition to build representations. Let us consider the function $\lambda x : \mathscr{S}.\, \lambda f : \mathscr{S} \to \mathscr{S}.\, f(x)$ as an example. A representation of this function is a representation of the constant family $(\lambda x : \mathscr{S}.\, \lambda f : \mathscr{S} \to \mathscr{S}.\, f(x))_{\gamma:\mathscr{S}^0}$.

Following the definition of representability, we fix a natural number $k \geq 0$ and provide a local representation of the family $(\lambda x : \mathscr{S}.\, \lambda f : \mathscr{S} \to \mathscr{S}.\, f(x))_{\gamma:\mathscr{S}^k}$. Following the representation scheme on functions types twice, we fix fully representable families $(x_\gamma : \mathscr{S})_{\gamma:\mathscr{S}^k}$ and $(f_\gamma : \mathscr{S} \to \mathscr{S})_{\gamma:\mathscr{S}^k}$ and provide a local representation of $(f_\gamma(x_\gamma))_{\gamma:\mathscr{S}^k}$. By the representation scheme on function types, it suffices to apply the local representation of f to the representation of x to obtain the desired representation.

For our new notion of representations to be useful, it is important that we can still use representable predicates for separation and representable functions for replacement. The following formula provides the necessary property for predicates if we set $n \coloneqq 0$.

5.2.15 Lemma. *Let $(P_\gamma : \mathscr{S} \to \mathbb{P})_{\gamma:\mathscr{S}^n}$ be a representable family of predicates. Then there is a formula φ such that $((\gamma, x) \vDash \varphi) \Leftrightarrow P_\gamma(x)$ for γ and x.*

Proof. By the representation scheme on $\mathbb{P}$, what we are looking for, is exactly a local representation of $(P_\gamma(x))_{(\gamma,x):\mathscr{S}^{n+1}}$. We obtain such a local representation if we apply the local representation of $(P_\gamma)_{(\gamma,x):\mathscr{S}^{n+1}}$ to the local representation of $(x)_{(\gamma,x):\mathscr{S}^{n+1}}$ that we define as $0 \doteq 1$. □

An analogous statement holds for representable functions of type $\mathscr{S} \to \mathscr{S}$ and allows us to use such functions for replacement.

With this version of representability, we can construct all the same basic representations as with the previous one. But now, we can treat logical operators in an even cleaner way since we know how to represent functions that take propositions as arguments.

5.2.16 Lemma. *Conjunction, disjunction, negation, implication and equivalence are representable.*

This lemma alone provides everything that we need to handle logical operator during reification. We will not go into the proof of the lemma which is not much different from the analogous lemma in the previous subsection.

The ability to represent higher order-functions allows us to treat quantifiers in the same simple way:

5.2.17 Lemma. *The quantifiers $(\exists\_ : \mathscr{S}.\, \_) : (\mathscr{S} \to \mathbb{P}) \to \mathbb{P}$ and $(\forall\_ : \mathscr{S}.\, \_) : (\mathscr{S} \to \mathbb{P}) \to \mathbb{P}$ are representable.*

Proof. We just consider the existential quantifier. Fix a natural number k and a representable family of predicates $(P_\gamma : \mathscr{S} \to \mathbb{P})_{\gamma:\mathscr{S}^k}$. We need to provide a local representation of the family $(\exists x.\, P_\gamma(x))_{\gamma:\mathscr{S}^k}$, that is, a formula φ such that $(\gamma \vDash \varphi) \Leftrightarrow \exists x.\, P_\gamma(x)$. By Lemma 5.2.15, we obtain a predicate φ_P such that $((\gamma, x) \vDash \varphi) \Leftrightarrow P_\gamma(x))$ for all γ and x. The formula $\varphi \coloneqq \dot{\exists}\varphi_P$ satisfies the desired property. □

Note that this proof only works because we can assume a *full* representation of P. That is the reason why representations of function families must take *full* representations as inputs and why local representations do not suffice.

The flexible notion of representability that we introduced in this subsection, allows us to formulate meaningful representation schemes on other types than $\mathscr{S}$, $\mathbb{P}$ or function types built from those. The most obvious example are product types.

5.2.18 Definition (Provisional)**.** A local representation of a family of **pairs** $(x_\gamma, y_\gamma)_{\gamma:\mathscr{S}^n}$ is a pair of local representations of $(x_\gamma)_{\gamma:\mathscr{S}^n}$ and $(y_\gamma)_{\gamma:\mathscr{S}^n}$.

Under this scheme, all the basic operations are representable:

5.2.19 Lemma. *Given representation schemes on types A and B, the pair constructor $(\_,\_) : A \to B \to A \times B$ and the projection $\pi_1 : A \times B \to A$ and $\pi_2 : A \times B \to B$ are representable.*

Now we can mechanically construct, for example, a representation of the function

$$\lambda x : \mathscr{S} \times \mathscr{S}.\, \lambda f, g : \mathscr{S} \to \mathscr{S}.\, (f(\pi_1(x)), g(\pi_2(x))).$$

We can even represent members of arbitrary class-like types, like natural numbers:

5.2.20 Definition (Provisional)**.** Given a class-like type A, a family of members $(x_\gamma : A)_{\gamma:\mathscr{S}^n}$ is representable if the family of encodings $(\mathsf{encode}(x_\gamma))_{\gamma:\mathscr{S}^n}$ is representable.

5.2.21 Lemma. *Given a class-like type, the functions $\mathsf{encode}_A : A \to \bar{A}$ and $\mathsf{decode} : \bar{A} \to A$ are representable.*

This leads leads to a certain conflict. We can have multiple different representation schemes on a single type. For example, we have a representation scheme on $\mathscr{S} \times \mathscr{S}$ as a class-like type but we also have the general representation scheme for product types. In this example, one can check that both representation schemes are equivalent and that will typically be true for similar conflicts.

5.2.3 Quantification over class-like types (Version 3)

So far, we can only represent quantification over $\mathscr{S}$. It is desirable to include quantification over other types. For arbitrary types, this does not seem possible but with a modification of our framework, we can represent quantifications over class-like types. To this end, it is necessary to modify our notion of environments. So far, we used tuples from a type $\mathscr{S}^n$ as environments that provide values for free variables. If we quantify over other types than $\mathscr{S}$ then we need environments that may contain members of those types

5.2.22 Definition (Provisional)**.** An **environment type** of size $n : \mathbb{N}$ is a product $A_1 \times \cdots \times A_n$ of class-like types. Members of environment types are called **environments**.

Now, it does not suffice to compare environment types by their size. We need to define the notion of environment type extension.

5.2.23 Definition (Provisional)**.** We define inductively what it means that an environment type Δ **extends** an environment type Γ (abbreviated as $\Delta \succ \Gamma$):

- $\Gamma \succ \Gamma$ for all environment types Γ.
- If $\Delta \succ \Gamma$ then $\Delta \times A \succ \Gamma$ for all environment types Γ and Δ and class-like types A.

We need to redefine the lifting operation from the previous subsection.

5.2.24 Definition (Provisional)**.** Fix a family $x : \Gamma \to A$ over some environment type Γ and an environment type $\Delta \succ \Gamma$. We define a family $\Uparrow^{\Delta}_{\Gamma} x : \Delta \to A$ by $\Uparrow^{\Delta}_{\Gamma} x \coloneqq x \circ \mathsf{reduce}^{\Delta}_{\Gamma}$ where reduce is the function that takes an environment of type Δ and obtains an environment of type Γ by discarding the last few elements. We will apply reduce implicitly where it is clear from the context. We usually write just $\Uparrow x$ and leave n and k implicit.

We can now define the new notion of representation schemes.

5.2.25 Definition (Provisional)**.** A **representation scheme** on a type A consists of a function $\mathsf{LocalRep} : \prod_{\Gamma}(\Gamma \to A) \to \mathsf{Type}$. This function takes an environment type Γ and a family $(x_\gamma : A)_{\gamma:\Gamma}$ and returns the type of **local representations** of x.

Given a representation scheme on A and a family $(x_\gamma : A)_{\gamma:\Gamma}$ over some environment type Γ, a (full) **representation** of x is a family of type $\prod_{\Delta \succ \Gamma} \mathsf{LocalRep}_{\Delta}(\Uparrow x)$.

Given a type type A with a representation scheme and a member $x : A$, a representation of x is a representation of the constant family $(x)_{\gamma:\mathsf{Unit}}$.

The representation schemes for functions and sets do not need to be changed significantly but to define the representation scheme for propositions, we need an operation that takes an environment and turns it into a tuple of sets that we can use as a first-order environment.

5.2.26 Definition (Provisional)**.** Given an environment type $\Gamma = A_1 \times \cdots \times A_n$, we define the operation of **environment encoding** an Γ that maps every environment $(x_1, \ldots, x_n) : \Gamma$ to $(\mathsf{encode}(x_1), \ldots, \mathsf{encode}(x_n)) : \mathscr{S}^n$. This is possible since all A_i are class-like. We will always apply this operation implicitly.

Now, we define the new representation scheme for propositions.

5.2.27 Definition (Provisional)**.** Fix an environment type Γ. A local representation of a family of **propositions** $(P_\gamma : \mathbb{P})_{\gamma:\Gamma}$ is a formula φ, such that $(\gamma \vDash \varphi) \Leftrightarrow P_\gamma$ for all environments $\gamma : \Gamma$.

The of this subsection was to represent quantification over arbitrary class-like types. The next lemma prepares this.

5.2.28 Lemma. *Fix an environment type Γ and a class-like type A. Let $(P_\gamma : A \to \mathbb{P})_{\gamma:\Gamma}$ be a representable family of predicates. Then there is a formula φ such that $((\gamma, x) \vDash \varphi) \Leftrightarrow P_\gamma(x)$ for γ and x.*

Proof. The proof is completely analogous to the one from the previous subsection. By the representation scheme on $\mathbb{P}$, what we are looking for, is exactly a local representation of $(P_\gamma(x))_{(\gamma,x):\Gamma\times A}$. We obtain such a local representation if we apply the local representation of $(P_\gamma)_{(\gamma,x):\Gamma\times A}$ to the local representation of $(x)_{(\gamma,x):\Gamma\times A}$ that we define as $0 \doteq 1$. □

5.2.29 Lemma. *Fix a class-like type A. The quantifiers $(\exists\_ : A.\_) : (A \to \mathbb{P}) \to \mathbb{P}$ and $(\forall\_ : A.\_) : (A \to \mathbb{P}) \to \mathbb{P}$ are representable.*

Proof. Again, the proof is analogous to the previous subsection. □

5.2.4 Dependent functions (Version 4)

Next, we want to be able to represent dependent functions. This has two important consequences: The types of subterms may depend on variables from the environment. Hence, we need representation schemes on families of types, not just constant types.

Moreover, we need to generalise the definition of environment types to allow type dependencies. Informally, we can think of those environment types as nested sigma types of the form $\sum_{x_1:A_1} \dots \sum_{x_n:A_n} \mathsf{Unit}$ where every A_i stands for a class-like type and may contain $x_1, \dots, x_{i-1}$ as free variables. Formally, it will be more convenient to nest the sigma types in the other direction, or rather stack them notationally. This leads to our first final definition.

5.2.30 Definition. We define inductively what constitutes an **environment type** of size $n : \mathbb{N}$:

- Unit is the only environment type of size 0.
- If Γ is an environment type of size n and $(A_\gamma)_{\gamma:\Gamma}$ is a family of class-like types then $\sum_{\gamma:\Gamma} A_\gamma$ is an environment type of size $n+1$.

Members of an environment type are called **environments**.

In analogy to the previous subsection, we can define what it means that one environment type Δ extends another environment type Γ and how we can lift families over Γ to families over Δ in this case.

5.2.31 Definition (Provisional)**.** Fix an environment type Γ. A **representation scheme** on a family of types $(A_\gamma : \mathsf{Type})_{\gamma:\Gamma}$ consists of a function $\mathsf{LocalRep} : \prod_{\Delta \succ \Gamma} (\prod_{\delta:\Delta} A_\delta) \to \mathsf{Type}$. This function takes an environment type $\Delta \succ \Gamma$ and a family $(x_\delta : A_\delta)_{\delta:\Delta}$ and returns the type of **local representations** of x.

Given a representation scheme on $(A_\gamma : \mathsf{Type})_{\gamma:\Gamma}$ and a family $(x_\delta : A_\delta)_{\delta:\Delta}$ over Γ, a (full) **representation** of x is a family of type $\prod_{\Delta \succ \Gamma} \mathsf{LocalRep}_\Delta(\Uparrow x)$.

The representation scheme for function types can now be generalised to dependent function types.

5.2.32 Definition (Provisional)**.** A local representation of a family of **dependent functions** $(f_\gamma : \prod_{x:A_\gamma} B_\gamma(x))_{\gamma:\Gamma}$ is a function that takes a *full* representation of any family $(x_\gamma : A_\gamma)_{\gamma:\Gamma}$ and returns a local representation of $(f_\gamma(x_\gamma) : B_\gamma(x_\gamma))_{\gamma:\Gamma}$.

Other definitions and lemmas do not change significantly, except for additional type annotations.

5.2.5 Polymorphic functions (Version 5)

By polymorphic functions we mean functions that take a type as argument and use that type for further arguments or for the result. Some example are the identity function

$\mathsf{id} : \prod_{A:\mathsf{Type}} A \to A$, the general existential quantifier $(\exists\_.\,\_) : \prod_{A:\mathsf{Type}}(A \to \mathbb{P}) \to \mathbb{P}$ and the recursor

$$\mathsf{rec}_{\mathbb{N}} : \prod_{C:\mathbb{N}\to\mathsf{Type}} C(0) \to (\prod_n C(n) \to C(n+1)) \to \prod_n C(n).$$

There are two reasons why we cannot represent such polymorphic functions yet. First, we have not defined a representation scheme for types. Second, more importantly, we need a more general representation scheme on function types. In the example of the identity function, we do not get a representation scheme on $A \to A$ unless we know more about A. The representation scheme of the result type depends on the specific input and our representation scheme on function types should permit that.

We will slightly reformulate the definition of representation schemes in order to introduce the notion of local representation schemes which we will use as one form of representation for type families.

5.2.33 Definition. Fix an environment type Γ.

A **local representation scheme** on a family of types $(A_\gamma : \mathsf{Type})_{\gamma:\Gamma}$ consists of a function $\mathsf{LocalRep} : (\prod_{\gamma:\Gamma} A_\gamma) \to \mathsf{Type}$. This function takes an environment type $\Delta \succ \Gamma$ and a family $(x_\delta : A_\delta)_{\delta:\Delta}$ and returns the type of **local representations** of x.

A (full) **representation scheme** on a family of types $(A_\gamma : \mathsf{Type})_{\gamma:\Gamma}$ consists of local representation schemes on all lifted families $\Uparrow^\Delta A$ for $\Delta \succ \Gamma$. Given a representation scheme on A and a family $(x_\gamma : A_\gamma)_{\gamma:\Gamma}$, a (full) **representation** of x is a family of type $\prod_{\Delta\succ\Gamma} \mathsf{LocalRep}(\Uparrow x)$.

Now let us look at the representation scheme for function types.

5.2.34 Definition. A local representation of a family of **dependent functions** $(f_\gamma : \prod_{x:A_\gamma} B_\gamma(x))_{\gamma:\Gamma}$ is a function that takes a *full* representation of any family $(x_\gamma : A_\gamma)_{\gamma:\Gamma}$ and returns a local representation of $(f_\gamma(x_\gamma) : B_\gamma(x_\gamma))_{\gamma:\Gamma}$.

This definition is an exact copy of previous version but the important point are the implicit assumptions. For this definition to work, it suffices to have a representation scheme on $(x_\gamma : A_\gamma)_{\gamma:\Gamma}$ and a local representation scheme on $(B_\gamma(x_\gamma))_{\gamma:\Gamma}$ only when x is *representable*. This means that we get a representation scheme on the type of the identity function $\mathsf{id} : \prod_{A:\mathsf{Type}} A \to A$ if we formulate the representation scheme on types in a way such that every representable family $(A_\gamma : \mathsf{Type})_{\gamma:\Gamma}$ has a representation scheme. The next definition follows exactly that goal.

5.2.35 Definition. A local representation of a family of types $(A_\gamma : \mathsf{Type})_{\gamma:\Gamma}$ (**on type level**) is just a local representation scheme on A.

With this scheme, local representability of a family of types is equivalent to existence of a local representations scheme and full representability is equivalent to the existence of a full representation scheme. We could now easily represent the identity function. But for the recursor on $\mathbb{N}$ and the existential quantifier, this does not seem to suffice. Those are only representable in a meaningful way when the type is also class-like. This inspires the following definition.

5.2.36 Definition. A local representation of a family $(A_\gamma : \mathsf{Type})_\gamma$ **on class level** is a representable family of encodings of all A_γ.

Since every family of class-like types has a representation scheme, this formulation is stronger than the one for type-level representations. Finally, we define a similar representation scheme on set level.

5.2.37 Definition. A local representation of a family $(A_\gamma : \mathsf{Type})_\gamma$ **on set level** is a representable family of set encodings of all A_γ.

Generally, we will say that something is representable on type level if we use a representation scheme that relies only on the type-level representability of types. We will do the same class-level and set-level representability.

The representation schemes for types are not only useful for polymorphic function, they also make sense on type constructors. For example, we can prove that the type product is representable on set-level. This knowledge gives us representation schemes on families of products. representations schemes on families of the form $(A_\gamma \times B_\gamma)_\gamma$. The representation scheme on class level gives us representable families of encodings and the representation scheme on set level additionally tells us that those encodings are sets. It seems like sets, classes and just types with representation schemes are three different type-like concepts in the world of set theory and we may need each of them depending on the context.

5.3 First-order representations (Final version)

In this section, we combine the definitions that were developed in the previous section into a self-contained formulation of representability that we will work with. In addition, we will fill some formal gaps. We will not prove any of the lemmas here. The proofs are analogous to those of the previous section.

5.3.1 Definition. We define inductively what constitutes an **environment type** of size $n : \mathbb{N}$:

- Unit is the only environment type of size 0.
- If Γ is an environment type of size n and $(A_\gamma)_{\gamma:\Gamma}$ is a family of class-like types then $\sum_{\gamma:\Gamma} A_\gamma$ is an environment type of size $n + 1$.

Members of an environment type are called **environments**.

5.3.2 Definition. Given an environment type Γ of size n, we define the operation of **environment encoding** an Γ that maps every environment $(x_1, \ldots, x_n) : \Gamma$ to $(\mathsf{encode}(x_1), \ldots, \mathsf{encode}(x_n)) : \mathscr{S}^n$. We will always apply this operation implicitly.

5.3.3 Definition. We define inductively what it means that an environment type Δ **extends** an environment type Γ (abbreviated as $\Delta \succ \Gamma$):

- $\Gamma \succ \Gamma$ for all environment types Γ.
- If $\Delta \succ \Gamma$ then $\sum_{\delta:\Delta} A_\delta \succ \Gamma$ for all environment types Γ and Δ and families of class-like types $(A_\delta : \mathsf{Type})_{\delta:\Delta}$.

5.3.4 Definition. Fix environment types Γ and $\Delta \succ \Gamma$, a family of types $(A_\gamma : \mathsf{Type})_\gamma$ and a family of members $x : \prod_\gamma A_\gamma$. We define a family $\Uparrow_\Gamma^\Delta x : \prod_{\delta:\Delta} A_\delta$ by $\Uparrow_\Gamma^\Delta x := x \circ \mathsf{reduce}_\Gamma^\Delta$ where reduce is the function that takes an environment of type Δ and obtains an environment of type Γ by discarding the last few elements. We will apply reduce implicitly where it is clear from the context. We usually write just $\Uparrow x$ and leave n and k implicit.

5.3.5 Definition. Fix an environment type Γ.

A **local representation scheme** on a family of types $(A_\gamma : \mathsf{Type})_{\gamma:\Gamma}$ consists of a function $\mathsf{LocalRep} : (\prod_{\gamma:\Gamma} A_\gamma) \to \mathsf{Type}$. This function takes an environment type $\Delta \succ \Gamma$ and a family $(x_\delta : A_\delta)_{\delta:\Delta}$ and returns the type of **local representations** of x.

A (full) **representation scheme** on a family of types $(A_\gamma : \mathsf{Type})_{\gamma:\Gamma}$ consists of local representation schemes on all lifted families $\Uparrow^\Delta A$ for $\Delta \succ \Gamma$. Given a representation scheme on A and a family $(x_\gamma : A_\gamma)_{\gamma:\Gamma}$, a (full) **representation** of x is a family of type $\prod_{\Delta \succ \Gamma} \mathsf{LocalRep}(\Uparrow x)$.

We sometimes say that a value x is representable to mean that the constant family $(x)_{\gamma:\mathsf{Unit}}$ is representable.

5.3.6 Lemma. *Fix two environment types Γ and $\Gamma' \succ \Gamma$, a representation scheme on a family $(A_\gamma : \mathsf{Type})_{\gamma:\Gamma}$ and a family of members $(x_\gamma : A_\gamma)_{\gamma:\Gamma}$. We can restrict the representation of A to a representation of $\Uparrow^{\Gamma'} A$ and we can restrict every representation of x to a representation of the lifted family $\Uparrow^{\Gamma'} A$. We call this weakening.*

5.3.7 Definition. Fix an environment type Γ of size n.

- A local representation of a family of **dependent functions** $(f_\gamma : \Pi_{x:A_\gamma} B_\gamma(x))_{\gamma:\Gamma}$ is a function that takes a *full* representation of any family $(x_\gamma : A_\gamma)_{\gamma:\Gamma}$ and returns a local representation of $(f_\gamma(x_\gamma))_{\gamma:\Gamma}$.
- A local representation of a family of **propositions** $(P_\gamma : \mathbb{P})_{\gamma:\Gamma}$ is a formula φ with n free variables, such that $(\gamma \vDash \varphi) \Leftrightarrow P_\gamma$ for all environments $\gamma : \Gamma$.
- A local representation of a family of **classes** $(C_\gamma : \mathsf{Class})_{\gamma:\Gamma}$ is a local representation of the family of propositions $(x \in C_\gamma)_{\gamma,x}$.
- A local representation of a family of **sets** $(x_\gamma : \mathscr{S})_{\gamma:\Gamma}$ is a local representation of the family of propositions $(x_\gamma = y)_{\gamma,y}$.
- Fix a family of class-like types $(A_\gamma : \mathsf{Type})_{\gamma:\Gamma}$. A local representation of a family of members $(x_\gamma : A_\gamma)_{\gamma:\Gamma}$ is a local representation of the family of encodings $(\mathsf{encode}(x_\gamma))_{\gamma:\Gamma}$.

- Fix a family of propositions $(P_\gamma : \mathbb{P})_{\gamma:\Gamma}$. Every family of **proofs** $(p_\gamma : P_\gamma)_{\gamma:\Gamma}$ has exactly one local representation which caries no further information.
- Fix a family of types $(A_\gamma : \mathsf{Type})_{\gamma:\Gamma}$. A local representation of a family of **encodings** of the A_γ is a representation of the family of encoding classes $(\bar{A}_\gamma : \mathsf{Class})_{\gamma:\Gamma}$.
- A local representation of a family of **types** $(A_\gamma : \mathsf{Type})_{\gamma:\Gamma}$ (**on type level**) is just a local representation scheme on A.
- A local representation of a family $(A_\gamma : \mathsf{Type})_\gamma$ **on class level** is a representable family of encodings of all A_γ.
- A local representation of a family $(A_\gamma : \mathsf{Type})_\gamma$ **on set level** is a representable family of set encodings of all A_γ.

If we want to construct representations mechanically, then we just need to use the representation scheme for functions: To construct a representation for an application, we need representations for the arguments. To construct a representation for a lambda expression, we need a representation for the body. Like this, we reduce the goal to representability of subterms that can already represent or that we need to handle manually. For an example, see the proof of Lemma 5.4.1.

5.3.8 Lemma. *Fix an environment type Γ, a family of class-like types $(A_\gamma : \mathsf{Type})_{\gamma:\Gamma}$ and a family of predicates $(P_\gamma : A_\gamma \to \mathbb{P})_{\gamma:\Gamma}$. Then P is representable iff there is a formula φ such that $((\gamma, x) \vDash \varphi) \Leftrightarrow P_\gamma(x)$ for γ and x.*

5.3.9 Lemma. *Fix an environment type Γ, a family of class-like types $(A_\gamma : \mathsf{Type})_{\gamma:\Gamma}$ and a family of functions $(f_\gamma : A_\gamma \to \mathscr{S})_{\gamma:\Gamma}$. Then f is representable iff there is a formula φ such that $((\gamma, x, y) \vDash \varphi) \Leftrightarrow f_\gamma(x) = y$ for all γ, x and y.*

5.4 Representations of logical operations

We will provide representations for the common logical operations and all the basic set operations. In the end, we can automatically represent arbitrary predicates that are constructed from those operators. The second paragraph of the following proof serves as a demonstration for the automatic construction of representations.

5.4.1 Lemma. *The operations of logical conjunction, negation, disjunction, implication and equivalence are representable.*

Proof. We start with conjunction and negation since those are immediately expressed in our version of first-order formulas. To define a representation of the conjunction operation $\_ \wedge \_ : \mathbb{P} \to \mathbb{P} \to \mathbb{P}$, we fix an environment type Γ and two representable families of propositions $(P_\gamma : \mathbb{P})_{\gamma:\Gamma}$ and $(Q_\gamma : \mathbb{P})_{\gamma:\Gamma}$. According to the representation scheme for families of propositions, the representation schemes consist of formulas φ_P and φ_Q, such that $(\gamma \vDash \varphi_P) \Leftrightarrow P_\gamma$ and $(\gamma \vDash \varphi_Q) \Leftrightarrow Q_\gamma$ for all $\gamma : \Gamma$. We need to provide a formula

$\varphi_{P\wedge Q}$ such that $(\gamma \vDash \varphi_{P\wedge Q}) \Leftrightarrow P_\gamma \wedge Q_\gamma$ for all $\gamma : \Gamma$. By definition, $\varphi_{P\wedge Q} \coloneqq \varphi_P \mathbin{\dot\wedge} \varphi_Q$ satisfies this property.

In the same way, we can define a representation of the negation operation. To define a representation of disjunction, it suffices to express this operation in terms of conjunction and negation which we can already represent. Consider the equality $(\_ \vee \_) = (\lambda P, Q.\, \neg(\neg P \wedge \neg Q))$ that we get by function extensionality, proposition extensionality and the law of excluded middle. The representation of the right side can be built automatically from the representations of negation and conjunction. For demonstration purposes, we will go through the mechanical process explicitly: First, we fix an environment type Γ. Following the rule for lambda expressions (twice), we introduce two representable families of propositions $(P_\gamma : \mathbb{P})_{\gamma:\Gamma}$ and $(Q_\gamma : \mathbb{P})_{\gamma:\Gamma}$ and need to represent $(\neg(\neg P_\gamma \wedge \neg Q_\gamma))_{\gamma:\Gamma}$ locally. Following the rule for applications, we need to represent $(\neg\_)_{\gamma:\Gamma}$ and $(\neg P_\gamma \wedge \neg Q_\gamma)_{\gamma:\Gamma}$ locally. We get the first representation from the representation of negation. For the second representation, we follow the rule for applications (twice) and need to represent $(\_ \wedge \_)_{\gamma:\Gamma}$, $(\neg P_\gamma)_{\gamma:\Gamma}$ and $(\neg Q_\gamma)_{\gamma:\Gamma}$ locally. Again, we get the first representation from the representation of conjunction by weakening. For each of the other two, we apply the application rule once more and need to represent $(\neg\_)_{\gamma:\Gamma}$, P and Q locally. We have already done the first and have representations of P and Q by assumption.

In a similar way, we can obtain a representation of implication and equivalence with the equalities $(\_ \Rightarrow \_) = (\lambda P, Q.\, \neg P \vee Q)$ and $(\_ \Leftrightarrow \_) = (\lambda P, Q.\, (P \Rightarrow Q) \wedge (Q \Rightarrow P))$. □

5.4.2 Lemma. *The element relation and equality on sets are representable.*

Proof. We consider only the element relation. The construction for the equality relation is analogous. Let A be a class-like type. We want to show that the function $\_ \in \_ : A \to A \to \mathbb{P}$ is representable. Fix an environment type Γ and two representable families $(x_\gamma : A)_{\gamma:\Gamma}$ and $(y_\gamma : A)_{\gamma:\Gamma}$. By definition, their representations are representations of the families $(x_\gamma = z)_{\gamma,z}$ and $(y_\gamma = z)_{\gamma,z}$ which are, by definition, two formulas φ_x and φ_y, such that $(\gamma, u \vDash \varphi_x) \Leftrightarrow (x_\gamma = u)$ and $(\gamma, v \vDash \varphi_y) \Leftrightarrow (y_\gamma = v)$ for all u,v and γ. We can represent $(x_\gamma = y_\gamma)_\gamma$ by the formula $\dot\exists\varphi_x \mathbin{\dot\wedge} \dot\exists\varphi_y \mathbin{\dot\wedge} 1 \mathbin{\dot\in} 0$, as one can see by the equivalence

$$\begin{aligned}
&(\gamma \vDash \dot\exists\varphi_x \mathbin{\dot\wedge} \dot\exists\varphi_y \mathbin{\dot\wedge} 1 \mathbin{\dot\in} 0)\\
&\quad\Leftrightarrow (\exists u.\, (\gamma, u \vDash \varphi_x) \wedge \exists v.\, (\gamma, v \vDash \varphi_y) \wedge u \in v)\\
&\quad\Leftrightarrow (\exists u.\, x_\gamma = u \wedge \exists v.\, y_\gamma = v \wedge u \in v)\\
&\quad\Leftrightarrow (x_\gamma \in y_\gamma).
\end{aligned}$$

□

5.4.3 Lemma. *The existential quantifier* $\exists\_.\_ : \prod_{A:\mathsf{Type}}(A \to \mathbb{P}) \to \mathbb{P}$ *and the universal quantifier* $\forall\_.\_ : \prod_{A:\mathsf{Type}}(A \to \mathbb{P}) \to \mathbb{P}$ *are representable on class-level.*

Proof. First, we treat the existential quantifier, then we derive the representation of the universal quantifier from that. Fix an environment type Γ, a family of types $(A_\gamma : \mathsf{Type})_{\gamma:\Gamma}$ with a representable family of encodings and a representable family of

predicates $(P_\gamma : A_\gamma \to \mathbb{P})_{\gamma:\Gamma}$. We will construct a local representation of the family of existential statements $(\exists x : A_\gamma.\, P_\gamma(x))_{\gamma:\Gamma}$. From the representation of A, we obtain a formula φ_A, such that $(\gamma, x \vDash \varphi_A) \Leftrightarrow x \in \bar{A}_\gamma$ for all γ and x. From the representation of P, we obtain by Lemma 5.3.8 a formula φ_P, such that $(\gamma, x \vDash \varphi_P) \Leftrightarrow P_\gamma(\mathsf{decode}(x))$ for all γ and $x : \bar{A}_\gamma$. The desired representation is $\dot{\exists}\varphi_A \mathbin{\dot{\wedge}} \varphi_P$, as we can see from the equivalence

$$\begin{aligned}(\gamma \vDash \dot{\exists}\varphi_A \mathbin{\dot{\wedge}} \varphi_P)\\ &\Leftrightarrow (\exists x : \mathscr{S}.\, x \in \bar{A}_\gamma \wedge P_\gamma(\mathsf{decode}(x)))\\ &\Leftrightarrow (\exists x : A_\gamma.\, P_\gamma(x)).\end{aligned}$$

The representability of universal quantification follows from the equality $(\forall \_.\, \_) = (\lambda A, P.\, \neg\exists x : A.\, \neg P(x))$ since we can automatically construct a representation of the right side. □

5.4.4 Lemma. *Constant elements of class-like types are representable.*

Proof. Fix a class-like type A, a member $x : A$ and an environment type Γ. We need to provide a formula φ_x, such that $(\gamma, y \vDash \varphi_x) \Leftrightarrow (\mathsf{encode}(x) = y)$. This is obviously satisfied by $\varphi_x :\equiv (\mathsf{encode}(x) \doteq 0)$. □

5.4.5 Lemma. *The most basic class-operations, the empty class, unordered pairing, the power class operation, finite set comprehensions, replacement, indexed union, binary union, indexed intersection, binary intersection and difference all have representations.*

Proof. Their defining predicates are composed of the simple logical operations that we can represent by the results of this section. They can be derived mechanically. □

5.5 Automation

There are two thing in this chapter that we would like to automate: inference of representation schemes and construction of representations. Since representation schemes are also just representations of types, automatic construction of representation suffices. We will take a short look at the ideal properties of an automation approach and challenges.

The automation should recurse on the term structure. For applications and lambda expressions, that can be done using the representation scheme for functions. The automation should look for representations of subterms in a database where the user can register representations that they already proved. If the automation gets stuck at a subterm then it should present the corresponding goal to the user. Finally, it should be convenient to debug the automation process if it can't solve a goal or is too slow.

The first challenge is that not all Coq-expressions are applications or lambda expressions. There are match-expressions, for example. The automation would need to treat them separately which is difficult because of their complex structure. Other problematic constructs are definition of recursive functions as fixed points and inductive type definitions.

A subtle problem appears when the automation tries to construct a local representation of a family of applications $(f_\gamma(x_\gamma))_{\gamma:\Gamma}$. The usual rule says that we need a local representation for f and one for x. But under which representation scheme? There can be multiple options. Some of them might work and some might not. This can potentially lead to wide branching.

A third problem has to do with the wish that the automation leaves unsolved subgoals to the user. That is difficult because the branching that we mentioned will lead to multiple alternatives and it is likely not useful to choose one of them at random.

One promising approach to solve the last two problems would be to make the inference of representation schemes predictable. The first try would be to have one default representation scheme on every family. If the user wants to use another scheme then they can make that explicit as we do with representations on type level, class level and set level.

A final problem consists in the fact that we have groups of representation schemes that come from different definitions but are semantically equivalent, for example, we have explicitly defined a representation scheme on $(\mathscr{S})_{\gamma:\mathsf{Unit}}$ but we also get one because $\mathscr{S}$ is class-like. The automation should support such equivalence classes and be able to build representations under one scheme from representations under another. If done naively, that would lead to cycles.

The current implementation in Coq is based on type class inference. But that does not seem like a good solution. The automation is difficult to debug, cannot handle multiple representation schemes on the same family and either solves a goal completely or fails but does not produce simpler subgoals. A custom automation tactic would likely be the right solution.

5.6 Representations of basic constructions

In this section, we will provide representations for many of the most fundamental operations of type theory. In each subsection, we consider one particular type constructor and the term constructor and eliminators associated to it. Since we talk about types, we will always have three potential representation schemes to consider. We start with ordered pairs, disjoint sums and function types then we continue with natural numbers and ordinals and, finally, with formulas as and example for a more complex inductive type.

5.6.1 Ordered pairs

Ordered pairs are one of the most important tools in type theory. We can characterise ordered pairs by the following things.

- One type constructor $\_ \times \_ : \mathsf{Type} \to \mathsf{Type} \to \mathsf{Type}$
- One pair constructor $(\_,\_) : \prod_{A,B} A \to B \to A \times B$
- Two projections $\pi_1 : \prod_{A,B} A \times B \to A$ and $\pi_2 : \prod_{A,B} A \times B \to B$

- The usual equalities describing the interaction between the pair constructor and the projections

The equalities are proofs and, hence, trivially representable. The other objects can be represented on type level, on class level and on set level. We start with type level.

5.6.1 Lemma. *The product operation, the pair constructor and the projections are representable on type level.*

Proof. For the product, fix an environment type Γ and two families of types $(A_\gamma : \mathsf{Type})_{\gamma:\Gamma}$ and $(B_\gamma : \mathsf{Type})_{\gamma:\Gamma}$ with representation schemes. We establish a local representation scheme by defining that a local representation of a pair $((x_\gamma, y_\gamma) : A_\gamma \times B_\gamma)_{\gamma:\Gamma}$ is a pair of local representations of $(x_\gamma)_{\gamma:\Gamma}$ and $(y_\gamma)_{\gamma:\Gamma}$.

With this definition, the representability of the pair constructor and the projections holds trivially. But for illustration, let us spell out the representation of the pair constructor. Fix an environment type Γ and two families of types $(A_\gamma : \mathsf{Type})_{\gamma:\Gamma}$ and $(B_\gamma : \mathsf{Type})_{\gamma:\Gamma}$ with representations schemes. Moreover, fix two families of members $(x_\gamma : A_\gamma)_{\gamma:\Gamma}$ and $(y_\gamma : B_\gamma)_{\gamma:\Gamma}$ with representations r_x and r_y. Then (r_x, r_y) is a local representation of the resulting family $((x_\gamma, y_\gamma))_{\gamma:\Gamma}$ by the local representation scheme from the previous paragraph. □

Now, we continue on class-level.

5.6.2 Lemma. *The product* $\_ \times \_ : \mathsf{Type} \to \mathsf{Type} \to \mathsf{Type}$ *is representable on class-level.*

Proof. Fix an environment type Γ and two families $A, B : \Gamma \to \mathsf{Type}$ with representable families of encodings. We need to provide a representable family of encodings of $(A_\gamma \times B_\gamma)_{\gamma:\Gamma}$. We use the same encoding for pairs that we defined in ZF_2 (Lemma 3.4.3). We can use it in ZF' since the definition only depends on the shared core axioms. The encoding class $\{\langle \mathsf{encode}(x), \mathsf{encode}(y)\rangle \mid x : A_\gamma, y : B_\gamma\}$ is trivially representable from $\gamma : \Gamma$ because it is composed of A_γ, B_γ, $\mathsf{encode}_{A_\gamma}$, $\mathsf{encode}_{B_\gamma}$ and the pairing operation, all of which are representable from Γ. □

5.6.3 Lemma. *The pairing function* $(\_,\_) : \prod_{A,B}. A \to B \to A \times B$ *is representable on class level.*

Proof. This follows from the definitional equality

$$(\lambda AB.\, \lambda(x : A).\, \lambda(y : B).\, (x, y)) \\ = (\lambda A, B, x, y.\, \{\{\mathsf{encode}_A(x)\}, \{\mathsf{encode}_A(x), \mathsf{encode}_B(y)\}\}).$$

The right side is representable since it is composed of finite set builders and encodings which are all representable. □

5.6.4 Lemma. *The projections* $\pi_1 : \prod_{A,B} A \times B \to A$ *and* $\pi_2 : \prod_{A,B} A \times B \to B$ *are representable on class level.*

Proof. We do the proof only for π_1. The projection is representable since it maps A, B and z to the unique $x : A$ that satisfies $\exists y : B.\, z = (x, y)$. This mapping is representable since the pair constructor is representable. □

5.6.5 Lemma. *The product, the pair constructor and the projections are representable on set-level.*

Proof. We start with the product. Most of the work is already done by representation on class level (Lemma 5.6.2). We only need to show, that the encodings are sets. But we have done that in the chapter about ZF_2 (Lemma 3.4.4).

Since we use the same encoding as for general class-like types, we can use the same representation for the pair constructor and the projections as in the previous two lemmas. □

We can also prove all results from this subsection for the more general sigma types.

5.6.6 Lemma. *The sigma operator, the pair constructor for sigma types and the projections for sigma types are representable on type-level, class-level and set-level.*

Proof. The proofs are structurally the same as those for the simple product. They only more complex type annotations. □

All the three representation schemes that we introduced on products are compatible. A family of pairs that is representable on class level is representable on type level and a family of pairs that is representable on set level is representable on class-level.

5.6.2 Sum types

As in the previous section, we will provide representations of the type constructor, the element constructors and the eliminator and we will start with the most general representations scheme for types.

5.6.7 Lemma. *The sum operation* $\_ + \_ : \mathsf{Type} \to \mathsf{Type} \to \mathsf{Type}$*, the injections* $\mathsf{inj}_1 : \prod_{A,B} A \to A + B$ *and* $\mathsf{inj}_2 : \prod_{A,B} B \to A + B$ *and the sum eliminator*

$$\mathsf{rec}_+ : \prod_{A,B} \prod_{C : A+B \to \mathsf{Type}} (\prod_a C(\mathsf{inj}_1(a))) \to (\prod_b C(\mathsf{inj}_2(b))) \to \prod_{x : A+B} C(x)$$

are representable on type-level.

Proof. We start with the sum operation. Fix an environment type Γ and two families $A, B : \Gamma \to \mathsf{Type}$. We establish a representation scheme on $(A_\gamma + B_\gamma)_\gamma$ by defining that a local representation of a family $x : \prod_\delta A_\delta + B_\delta$ for $\Delta \succ \Gamma$ is a local representation of the partial application of the eliminator

$$(\lambda\Big(C : A_\gamma + B_\gamma \to \mathsf{Type}\Big).\, \lambda\Big(f : \prod_{a : A_\gamma} C(\mathsf{inj}_1(a))\Big).\, \lambda\Big(g : \prod_{b : B_\gamma} C(\mathsf{inj}_2(b))\Big).\, \mathsf{rec}_+(C, f, g, x).$$

The representation of the eliminator is then trivial to define. The representations of the injections is also easy. □

5.6.8 Lemma. *The sum operation is representable on class-level.*

Proof. Fix an environment type Γ and two families $A, B : \Gamma \to \mathsf{Type}$ with representable families of encodings. We need to provide a representable family of encodings of $(A_\gamma + B_\gamma)_{\gamma:\Gamma}$. We use the same encoding that we defined in ZF_2 (Lemma 3.4.6). We can use it in ZF' since the definition only depends on the shared core axioms. The encoding classes are

$$\{\mathsf{encode}((0, x)) \mid x : A_\gamma\} \cup \{\mathsf{encode}((1, y)) \mid y : B_\gamma\}$$

for $\gamma : \Gamma$. Their family is trivially representable because they are composed of A_γ, B_γ, $\mathsf{encode}_{A_\gamma}$, $\mathsf{encode}_{B_\gamma}$ and the pair constructor, all of which are representable from Γ. □

5.6.9 Lemma. *The injections* $\mathsf{inj}_1 : \prod_{A,B} A \to A + B$ *and* $\mathsf{inj}_2 : \prod_{A,B} B \to A + B$ *are representable on class-level.*

Proof. We will only consider inj_1. The other one is analogous. Fix representable families $A, B : \Gamma \to \mathsf{Type}$ and $x : \prod_{\gamma:\Gamma} A$ over an environment type Γ. A representation of $(\mathsf{inj}_1(x_\gamma))_\gamma$ is by definition nothing but a representation of $(\mathsf{encode}(\mathsf{inj}_1(x_\gamma)))_\gamma$ which is itself nothing but a representation of $(\mathsf{encode}((0, x_\gamma)))_\gamma$. But this family is composed of representable objects and, hence, representable. □

5.6.10 Lemma. *The dependent eliminator*

$$\begin{aligned}\mathsf{rec}_+ : \prod_{A,B}. \forall C : A + B \to \mathsf{Type}.\\ (\prod_x C(\mathsf{inj}_1(x))) \to\\ (\prod_y C(\mathsf{inj}_2(y))) \to\\ \prod_{z:A+B} P(z)\end{aligned}$$

is representable.

Proof. Fix representable families

$$\begin{aligned}A, B : \Gamma \to \mathsf{Type},\\ C : \prod_\gamma A_\gamma + B_\gamma \to \mathsf{Type},\\ f : \prod_\gamma \prod_x C_\gamma(\mathsf{inj}_1(x)),\\ g : \prod_\gamma \prod_y C_\gamma(\mathsf{inj}_2(y)) \text{ and}\\ z : \prod_\gamma A_\gamma + B_\gamma.\end{aligned}$$

The encoding of the result $(\mathsf{rec}_+(A_\gamma, B_\gamma, C_\gamma, f_\gamma, g_\gamma, z_\gamma))_\gamma$ is representable by as the unique solution to the representable predicate

$$\lambda u : C.\Big(\exists x : A_\gamma.\, z = \mathsf{inj}_1(x) \wedge u = f_\gamma(x)\Big) \vee \Big(\exists y : B_\gamma.\, z = \mathsf{inj}_2(y) \wedge u = g_\gamma(y)\Big).$$ □

5.6.11 Lemma. *The sum operation, the injection and the sum eliminator are representable on set-level.*

Proof. As in the case of ordered pairs, we can build on the previous lemmas and it only remains to show that the encoding type of a sum $A + B$ of set-like types is a set. But we also have already done that in the chapter about ZF_2 (Lemma 3.4.7). □

5.6.3 Functions

We already have a representation of function types by representation schemes and know how applications and lambda expressions are representable in those representations schemes. Now, we focus on representations of function types by encodings.

5.6.12 Lemma. *The dependent product operator* $(\prod\_\,\_) : \prod_{A:\mathsf{Type}}(A \to \mathsf{Type}) \to \mathsf{Type}$ *is representable on class-level.*

Proof. Fix families $A : \Gamma \to \mathsf{Type}$ and $B : \prod_\gamma A_\gamma \to \mathsf{Type}$ that are representable if we represent types by encodings. We need to provide a representable family of encodings of $(\prod_{a:A_\gamma}(B_\gamma(a)))_\gamma$. We will use the encodings that we have already defined in ZF_2 (Lemma 3.4.8): $\mathsf{encode}(f : \prod_{a:A_\gamma} B_\gamma(a)) \coloneqq \{\mathsf{encode}_{\sum_{x:A_\gamma} B_\gamma(x)}((x, f(x)) \mid x : A_\gamma\}$. We only need a representation for the family of images. We can write the images as

$$(\{R \subseteq \overline{\sum_{x:A_\gamma} B_\gamma(x)} \mid \forall x : A_\gamma.\, \exists! y : B_\gamma(x).\, \mathsf{encode}((x, y)) \in R\})_\gamma.$$

The range restricts R to relations that relate some $x : A_\gamma$ to elements from $B_\gamma(x)$ and the condition requires that this relation has to be total and functional, which means exactly that it encodes a function. The image is composed of representable components and, hence, representable. □

5.6.13 Lemma. *The pi operator* $(\prod\_\,\_) : \prod_{A:\mathsf{Type}}(A \to \mathsf{Type}) \to \mathsf{Type}$ *is representable on set-level.*

Proof. The construction is analogous to the previous lemma. From Lemma 3.4.9, we know that the encodings are sets. □

5.6.4 Natural numbers and ordinals

In this section, we consider the constant type $\mathbb{N}$. We already know that this type is set-like and will show that it is representable by a set encoding. This already subsumes the other two forms of representability, which keeps this subsection simpler.

5.6.14 Lemma. *The type $\mathbb{N}$ is representable on set-level.*

Proof. We already have the encoding and only need to provide a representation of the image $\bar{\mathbb{N}}$. It is easy to see that $\bar{\mathbb{N}}$ is the least solution (by set inclusion) of the predicate $\lambda A : \mathscr{S}.\, \bar{\mathbb{N}} \subseteq A$. By induction, we can prove that this predicate is equal to $\lambda A : \mathscr{S}.\, \emptyset \in A \wedge \forall x \in A.\, x^+ \in A$. In this formulation, it is trivially representable as a composition of representable parts. The property of being a least solution is also easily representable. Hence, we get a representation of $\bar{\mathbb{N}}$. □

5.6.15 Lemma. *The constructors $0 : \mathbb{N}$ and $\_ + 1 : \mathbb{N} \to \mathbb{N}$ are representable.*

Proof. This follows immediately from the definitions $\mathsf{encode}(0) \coloneqq \emptyset$ and $\mathsf{encode}(n+1) \coloneqq \mathsf{encode}(n)^+$ which are composed of representable objects. □

5.6.16 Lemma. *The recursor $\mathsf{rec}_{\mathbb{N}} : \prod_{A:\mathbb{N}\to\mathsf{Type}} A(0) \to (\prod_n A(n) \to A(n+1)) \to \prod_n A(n)$ is representable on class-level.*

Proof. Fix representable families $A : \Gamma \to \mathbb{N} \to \mathsf{Type}$, $x : \prod_\gamma A_\gamma(0)$, $f : \prod_\gamma \prod_n A_\gamma(n) \to A_\gamma(n+1)$ and $n : \Gamma \to \mathbb{N}$. We can represent the family of results $(\mathsf{rec}_{\mathbb{N}}(A_\gamma, x_\gamma, f_\gamma, n_\gamma) : A_\gamma(n_\gamma))_\gamma$ as the unique solutions to the family of representable predicates

$$\begin{aligned}\lambda\gamma.\, \lambda y.\, \exists(r : \prod_{k \le n_\gamma} A_\gamma(k)).r(0) = x_\gamma \wedge \\ \forall k : \mathbb{N}.\, k < n_\gamma \Rightarrow r(k+1) = f_\gamma(r(k)) \wedge \\ r(n) = y.\end{aligned}$$

What those predicates say in natural language is that there is a function r that satisfies the recursion rules on all $k < n_\gamma$ and that y is the result of r on n_γ. The results of the recursor satisfy the predicate trivially. It is easy to show by induction that they are the only values satisfying the predicate. The representability follows mostly from representations that we already have. The only new components are the relation $<$ and $\le$. But, since the encoding of a natural number contains exactly the encodings of all smaller numbers, we can formulate those in terms of other representable relations: $k < n \Leftrightarrow k \in n$ and $k \le n \Leftrightarrow k \in n \vee k = n$. □

Representations for operations on ordinals are completely analogous to natural numbers, except that we have one constructor more.

5.6.17 Lemma. *The zero ordinal, the successor function on ordinals, the union of sets of ordinals and the recursor on ordinals are representable on class-level.*

Proof. The proofs are analogous to those for natural numbers. □

We cannot represent the class of ordinals. The reason is that ordinals need to be well-founded in the strong external sense and we cannot express that in first-order logic (Section 5.1).

5.6.5 Formulas

In this subsection, we represent types of formulas and operations on formulas. This can be seen as an example of how to handle inductive types in general. We repeat the definition of first-order syntax over a class A.

$$\begin{aligned} t &::= x \mid a && x : \mathbb{N}, a : A \\ \varphi &::= \dot\exists\varphi \mid \varphi_1 \mathbin{\dot\wedge} \varphi_2 \mid \dot\neg\varphi \mid t_1 \doteq t_2 \mid t_1 \mathbin{\dot\in} t_2. \end{aligned}$$

Here, t stands for terms over A (elements of Term^A) and φ stands for formulas over A (elements of $\mathsf{Formula}^A$).

5.6.18 Lemma. *The type* Term^A *over a class* A *is class-like.*

Proof. Terms are either natural numbers that stand for variables or elements of A that stand for themselves. Hence, there is a bijection $\mathsf{Term}^A \sim \mathbb{N} + A$. Since the right side is class-like, the left side is class-like too. □

5.6.19 Lemma. *The type* $\mathsf{Formula}^A$ *over a class* A *is class-like.*

Proof. We use the following recursively defined encoding.

$$\begin{aligned} \mathsf{encode}(\dot\exists\varphi) &:= \langle 0, \mathsf{encode}(\varphi)\rangle \\ \mathsf{encode}(\varphi_1 \mathbin{\dot\wedge} \varphi_2) &:= \langle 1, \langle \mathsf{encode}(\varphi_1), \mathsf{encode}(\varphi_2)\rangle\rangle \\ \mathsf{encode}(\dot\neg\varphi) &:= \langle 2, \mathsf{encode}(\varphi)\rangle \\ \mathsf{encode}(t_1 \doteq t_2) &:= \langle 3, \langle \mathsf{encode}(t_1), \mathsf{encode}(t_2)\rangle\rangle \\ \mathsf{encode}(t_1 \mathbin{\dot\in} t_2) &:= \langle 4, \langle \mathsf{encode}(t_1), \mathsf{encode}(t_2)\rangle\rangle\,. \end{aligned}$$

Injectivity follows by induction from injectivity of the pairing function and the encoding on terms. □

5.6.20 Lemma. *The function* $\lambda A.\, \mathsf{Term}^A$ *is representable on class-level.*

Proof. This follows from $\overline{\mathsf{Term}^A} = \overline{\mathbb{N} + A}$ (Lemma 5.6.18) and the fact that $\lambda A.\, \overline{\mathbb{N} + A}$ is representable. □

5.6.21 Lemma. *The function* $\lambda A.\, \mathsf{Formula}^A$ *is representable on class-level.*

Proof. We already have encodings for $\mathsf{Formula}^A$; it suffices to show that the function $\lambda A.\,\overline{\mathsf{Formula}^A}$ is representable. To get a better understanding of those encoding classes, we write them down in an explicit form. We define a function $H : \mathsf{Class} \to \mathsf{Class} \to \mathsf{Class}$ by

$$\begin{aligned} H(A,B) \coloneqq {} & \{\langle 0, x\rangle \mid x \in B\} \\ & \cup \{\langle 1, \langle x, y\rangle\rangle \mid x, y \in B\} \\ & \cup \{\langle 2, x\rangle \mid x \in B\} \\ & \cup \{\langle 3, \langle u, v\rangle\rangle \mid u, v \in \overline{\mathsf{Term}^A}\} \\ & \cup \{\langle 4, \langle u, v\rangle\rangle \mid u, v \in \overline{\mathsf{Term}^A}\}. \end{aligned}$$

Then the encoding class is the fixed-point $\overline{\mathsf{Formula}^A} = \bigcup_{n:\mathbb{N}} H(A)^n(\emptyset)$ of the partially applied $H(A)$. Since the indexed union, the iteration on natural numbers and the empty set a representable, we can use this equation to reduce the representability of $\lambda A.\,\overline{\mathsf{Formula}^A}$ to representability of H. But that one reduces further to representability of $\lambda A.\,\overline{\mathsf{Term}^A}$ which we have by the previous lemma. □

5.6.22 Lemma. *The function $\lambda A.\,\mathsf{Term}^A$ is representable on set-level.*

Proof. We use use the same encodings as before from which we already know that they are representable (Lemma 5.6.20). It suffices to show that $\overline{\mathsf{Term}^A}$ is a set for every set A. This follows from the equality $\overline{\mathsf{Formula}^A} = \overline{\mathbb{N} + A}$ since the right side is a set. □

5.6.23 Lemma. *The function $\lambda A.\,\mathsf{Formula}^A$ is representable on set-level.*

Proof. We use the same encodings as before and we already showed that they are representable (Lemma 5.6.21). It suffices to show now that the $\overline{\mathsf{Formula}^A}$ is a set for every set A. Fix a set A. Then $\overline{\mathsf{Term}^A}$ is a set by the previous lemma. Hence, $H(A,B)$ is a set for every set B. By induction, it follows that $H^n(A,\emptyset)$ is a set for every natural number n and finally that $\overline{\mathsf{Formula}^A} = \bigcup_{n:\mathbb{N}} H^n(A,\emptyset)$ is a set. □

5.6.24 Lemma. *The constructors for terms and formulas over a class are representable.*

Proof. One just needs to unfold the encoding for formulas. Then it is routine to produce the representation. □

5.6.25 Lemma. *The recursors for terms and formulas over a class are representable.*

Proof. Since we represent terms over a class A as members of $A + \mathbb{N}$, we can represent the recursor by the recursor on the sum.

We only sketch the representation of the recursor for formulas. Recursion on formulas can be expressed as recursion on their depth. The depth of a formula φ representable as the least $n : \mathbb{N}$ such that $\mathsf{encode}(\varphi) \in H(A)^n(\emptyset)$ with H defined as in the encoding of formulas above. Since recursion on natural numbers is representable, recursion on formulas is then also representable. □

As a consequence, we get the next lemma.

5.6.26 Lemma. *The satisfaction relation*

$$\lambda(A : \mathsf{Class}), (n : \mathbb{N}), (\gamma : A^n), (\varphi : \mathsf{Formula}^A).\, \gamma \vDash^A \varphi$$

is representable.

Proof. With the representations of the recursor for formulas, the pair operations and the basic logical operations, one can construct this representation mechanically. □

6 Consistency of the axiom of choice

This chapter provides a proof of the relative semantic consistency of the axiom of choice. That is, we assume that we already have a model of ZF' and need to construct a new model L that satisfies AC. The definitions for the axiom of choice that we provided in ZF_2 are not suitable for ZF'. We copy them but include additional representability conditions:

6.0.1 Definition. The **axiom of choice** in ZF' states that for every *representable* family $F : I \to \mathscr{S}$ of inhabited sets over an index set I, there is propositionally a *representable* choice function that maps every $i \in I$ to an element of $F(i)$.

6.0.2 Definition. The **well-ordering theorem** in ZF' states that every set has a *representable* well-order.

As mentioned in Section 5.1, our notion of well-orders is stronger than anything that one could define in first-order logic. Hence, this well-ordering theorem is stronger than the typical formulation. It also implies a global version of the axiom of choice that works on classes.

The axiom of choice and the well-ordering theorem are still equivalent in ZF' but we only need one direction of this equivalence again:

6.0.3 Lemma. *The well-ordering theorem implies the axiom of choice.*

Proof. We can derive this from the corresponding lemma in ZF_2 (Theorem 3.6.4). We only need to show that the choice functions obtained by that lemma are representable. Constructing the representations is routine. □

The relative consistency proof for the axiom of choice has four steps:

1. Define a class L.
2. Show that L as a ZF structure with the relation $\in|_L$ is a model of ZF'
3. Show that $L^L = L$, where L^L stands for the definition of L inside of L as a model.
4. Show that L (and L^L) has a well-ordering. This implies the axiom of choice.

Those steps form the outline of this chapter; every step has a section. For the whole chapter, we fix a ZF model $\mathscr{S}$ that we call the **outer model**. Whenever we talk about set-related definitions without explicit specification of the model then we implicitly mean $\mathscr{S}$.

One may think that it would be easier to start with a model of ZF_2 and define an inner model of ZF' from that. This way, the outer theory would be strong and the inner theory would be weak, which makes the model construction as easy as possible. But our only work in the outer model is the definition of L and we need to define that one in the inner model anyway. Our treatment of set theory with Coq as a strong metatheory means that $\mathscr{S}$ and the inner model can both be models of ZF' in exactly the same formal sense. The definition of L that can also be used inside of L. Hence, it is even easier to use the same axiomatisation on both levels.

The formalisation of this whole chapter is work-in-progress.

6.1 The constructible hierarchy L

The rough idea behind the definition of L is that we start with the empty class and repeatedly add sets that can be defined (or constructed) from those that are already contained. We repeat this process transfinitely often, that is, once for every ordinal. There are two motivations for this construction. First, every set that is constructible from L in a certain sense, will be contained in L. This includes all sets that must exist by the ZF' axiom. Second, we obtain a description of L that is graspable enough to define a well-order on L.

We start by the formal definition of L. For clarity, we use $\alpha + 1$ as another notation for the successor α^+.

6.1.1 Definition. We define a transfinite sequence of **stages** $(L_\alpha : \mathscr{S})_{\alpha \in \mathsf{Ord}}$ by transfinite recursion:

$$\begin{aligned} L_0 &\coloneqq \emptyset \\ L_{\alpha+1} &\coloneqq \{A \subseteq L_\alpha \mid A : \mathsf{Class}^{L_\alpha} \text{ is representable over } L_\alpha\} \\ L_\lambda &\coloneqq \bigcup_{\alpha \in \lambda} L_\alpha. \end{aligned}$$

Remember that the superscript in $\mathsf{Class}^{L_\alpha}$ indicates that we talk about classes over L_α as a ZF structure. We define the **constructible universe** L as the union of those stages:

$$L \coloneqq \bigcup_{\alpha : \mathsf{Ord}} L_\alpha.$$

L is a proper class since the union ranges over all ordinals. As a result, L contains all ordinals, as we will see later. We can see by transfinite induction that all stages are sets. The only non-trivial induction case is the one for $\alpha + 1$. By Lemma 5.4.5, we only need to show that representability of classes over L_α is representable over the outer model. A class A over L_α is by definition representable over L_α iff there is a formula φ over L_α, such that $((x) \vDash^{L_\alpha} \varphi) \Leftrightarrow x \in A$ for all $x \in L_\alpha$. This condition is representable over the outer model since the set of formulas over a set is representable and the satisfaction relation on such formulas is representable (Lemma 5.6.26).

We finish the section with some simple lemmas that can help to understand the structure of L.

6.1.2 Lemma. *An inclusion of ordinals $\alpha \subseteq \beta$ implies that $L_\alpha \subseteq L_\beta$.*

We prove this lemma simultaneously with the next one.

6.1.3 Lemma. *Every stage is transitive.*

Proof of Lemma 6.1.2 and Lemma 6.1.3.
We do this proof by ordinal induction on β.

- If $\beta = \emptyset$ then $\alpha \subseteq \beta$ implies $\alpha = \emptyset$. It follows that $L_\alpha = \emptyset \subseteq L_\beta$.

  The empty set is trivially transitive.

- If $\beta = \beta' + 1 = \beta' \cup \{\beta'\}$ then $\alpha \subseteq \beta$ implies $\alpha = \beta$ (if $\beta' \in \alpha$) or $\alpha \subseteq \beta'$. In the first case, $L_\alpha \subseteq L_\beta$ holds trivially. In the second case, we have $L_\alpha \subseteq L_{\beta'}$ by the induction hypothesis. It remains to show that $L_{\beta'} \subseteq L_{\beta'+1}$. Fix $x \in L_{\beta'}$. Then $x \subseteq L_{\beta'}$ by transitivity of $L_{\beta'}$ (from the induction hypothesis). As a subset, $x \subseteq L_{\beta'}$ is representable over $L_{\beta'}$ by the predicate $\lambda y.\, y \in x$. Hence, $x \in L_{\beta'+1}$. Together, this implies $L_\alpha \subseteq L_\beta$.

  For transitivity of L_β, fix an $x \in L_\beta$. Then x must be a representable subset of $L_{\beta'}$ and, by $L_{\beta'} \subseteq L_\beta$ from the previous paragraph, a subset of of L_β.

- Let $\beta = \bigcup_{\xi \in \beta} \xi$ be a limit ordinal. If $\alpha = \beta$ then the goal is trivial. Otherwise, we have $\alpha \subsetneq \beta$ which implies $\alpha \in \beta$ (Lemma 3.5.9) and, hence, $\alpha \in \xi$ for some $\xi \in \beta$. By the induction hypothesis, $L_\alpha \subseteq L_\xi \subseteq \bigcup_{\xi \in \beta} L_\xi = L_\beta$.

  For transitivity, fix an $x \in L_\beta$. Then $x \in L_\xi$ for some $\xi \in \beta$. By the induction hypothesis, $x \subseteq L_\xi \subseteq \bigcup_{\xi \in \beta} L_\xi = L_\beta$. □

6.1.4 Lemma. *L is transitive.*

Proof. Fix an $x \in L = \bigcup_\alpha L_\alpha$. Then $x \in L_\alpha$ for some ordinal α. By the previous lemma, $x \subseteq L_\alpha \subseteq L$. □

6.2 L is a model of ZF′

We show now that the class L as a ZF structure satisfies all ZF axioms. We start with the structural axiom of extensionality and foundation which are inherited from the outer model.

6.2.1 Lemma. *L satisfies the axiom of extensionality.*

Proof. Fix $A, B \in L$ and assume that $x \in^L A \Leftrightarrow x \in^L B$ for all $x \in L$. By definition, this is the same as $x \in A \Leftrightarrow x \in B$. We have to prove that, under this assumption, $A = B$. By extensionality of the outer model, it suffices to show this property more generally for all $x \in \mathscr{S}$. We fix an $x \in \mathscr{S}$ and show the direction $x \in A \Rightarrow x \in B$. The other one is analogous. Assume that $x \in A$. By transitivity of L (Lemma 6.1.4), we know that $x \in L$ and, hence, that $x \in B$ by our assumption. □

We proved this lemma just for L but the proof actually works for all transitive classes.

6.2.2 Lemma. *L satisfies the axiom of foundation.*

Proof. We have to show that the relation $\in^L$ is well-founded. This holds since $\in^L$ is just the restriction of $\in$ and the restriction of a well-founded relation is well-founded. □

Now, we continue with the set construction axioms. For many of those axioms, it is rather easy to point out a stage by hand on which we find the desired sets. The empty set, for example is contained in $L_1 = \{\emptyset\}$. The axioms of separation and replacement are more difficult in this regard. We focus on the axiom of separation here. If we want to prove the axiom, we assume a set A over L and a predicate $P : A \to \mathbb{P}$ that is presentable over L. We need to show that $\{x \in A \mid P(x)\}$ is a set over L. We can easily obtain a stage that contains all elements of A (any stage that contains A, for example). If we find a stage L_α that additionally has a representation of P then $\{x \in A \mid P(x)\}$ is representable over that stage and, hence, an element of $L_{\alpha+1}$. To find such a stage, we will introduce the *reflection theorem* that we prepare with the following definition.

6.2.3 Definition. A set A **reflects** a class K with respect to a formula φ if $(\gamma \vDash^A \varphi) \Leftrightarrow (\gamma \vDash^K \varphi)$ for all γ.

The reflection theorem will allow us to obtain arbitrarily large stages that reflect L with respect to given formulas. Before we can prove it, we need the following auxiliary lemma.

6.2.4 Lemma. *Fix a class K and a countably infinite series of sets $A_1 \subseteq A_2 \subseteq A_3 \subseteq \cdots \subseteq K$ that reflect K with respect to a formula φ. Then the limit $\bigcup_{i:\mathbb{N}} A_i$ also reflects K with respect to φ.*

Proof. As a shorthand, we define $A \coloneqq \bigcup_{i:\mathbb{N}} A_i$. Let n be the number of free variables in φ. We prove the claim by induction on φ. The only non-trivial case is when $\varphi = \dot\exists\psi$. For the direction from left to right, fix $\gamma : A^n$ with $\gamma \vDash^A \dot\exists\psi$. There is some $x \in A$ with $(\gamma, x) \vDash^A \psi$. By the inductive hypothesis, we know that $(\gamma, x) \vDash^K \psi$. We also know that $x \in K$ since $A \subseteq K$. Hence, $\gamma \vDash^K \dot\exists\psi$.

For the other direction, assume that $\gamma \vDash^K \dot\exists\psi$. Let $i : \mathbb{N}$ be minimal such that $\gamma : A_i^n$. By assumption, A_i reflects K with respect to $\dot\exists\psi$, that is, there is an $x : A_i$ with $(\gamma, x) \vDash^{A_i} \varphi$. This implies that $(\gamma, x) \vDash^K \varphi$ and, by the inductive hypothesis, that $(\gamma, x) \vDash^A \varphi$. Since x is also in A, we have $\gamma \vDash \dot\exists\psi$. □

6.2.5 Reflection theorem. *For every formula φ, and stage L_α, there is a stage $L_\beta \supseteq L_\alpha$ that reflects L with respect to φ.*

Proof. We prove the statement by structural induction on φ.

- Assume $\varphi = \dot\neg\psi$. By the induction hypothesis, there is a stage $L_\beta \supseteq L_\alpha$, such that $(\gamma \vDash^{L_\beta} \psi) \Leftrightarrow (\gamma \vDash^L \psi)$ and, by definition of the satisfaction relation,

$$(\gamma \vDash^{L_\beta} \dot\neg\psi) \Leftrightarrow (\gamma \nvDash^{L_\beta} \psi) \Leftrightarrow (\gamma \nvDash^L \psi) \Leftrightarrow (\gamma \vDash^L \dot\neg\psi).$$

- Assume $\varphi = \varphi_1 \mathbin{\dot\wedge} \varphi_2$. It suffices to define a sequence of stages $L_\alpha \subseteq A_0 \subseteq B_0 \subseteq A_1 \subseteq B_1 \subseteq \cdots$, such that every A_i reflects L with respect to φ_1 and every φ_2 reflects L with respect to φ_2. Then, we can take $L_\beta \coloneqq A_0 \cup B_0 \cup A_1 \cup B_1 \cup \ldots$ which is a stage. It reflects L with respect to φ_1 by the previous lemma since it is equal to $A_0 \cup A_1 \cup \ldots$. By the same argument, it reflects L with respect to φ_2. Hence, it reflects $\varphi_1 \mathbin{\dot\wedge} \varphi_2$.

  To define such a sequence, we choose $A_0 \coloneqq L_\alpha$ and let A_{i+1} be the least stage that reflects L with respect to φ_1 and B_i the least stage that reflects L with respect to φ_2.

- Assume $\varphi = \dot\exists\psi$ and ψ has n free variables. We will choose a stage L_β that reflects L with respect to ψ. Moreover, it should contain enough witnesses from L to make even $\dot\exists\psi$ true whenever is is true over L. We define L_β by an iterative approach similar to the one from the previous case. We define a sequence of stages $L_\alpha \subseteq A_0 \subseteq B_0 \subseteq A_1 \subseteq B_1 \subseteq \cdots$, by the following rules. $A_0 \coloneqq L_\alpha$ and A_{i+1} is the least stage greater than B_i that reflects L with respect to ψ. We choose B_i as the least stage such that, whenever there is $\gamma : (A_i)^n$ and $x \in L$ with $(\gamma, x) \vDash^L \psi$, there is an $x' \in B_i$ that satisfies $(\gamma, x') \vDash^L \psi$. We can then take $L_\beta \coloneqq A_0 \cup A_1 \cup \ldots$. By the previous lemma, we know that this stage reflects L with respect to ψ. To check that it also reflects L with respect to $\dot\exists\psi$, fix a $\gamma : \mathscr{S}^n$.
  - Assume $\gamma \vDash^{L_\beta} \dot\exists\psi$. Then there is an $x \in L_\beta$, such that $\gamma, x \vDash^{L_\beta} \psi$ and, hence, $\gamma, x \vDash^L \psi$. It follows that $\gamma \vDash^L \dot\exists\psi$ since $x \in L_\beta \subseteq L$.
  - Assume $\gamma \vDash^L \dot\exists\psi$. Then there is an $x \in L$, such that $\gamma, x \vDash^L \psi$. Fix $i : \mathbb{N}$, such that all elements of γ are in A_i. Then there is, by definition, an $x' \in B_i \subseteq L_\beta$ that satisfies $\gamma, x' \vDash^L \psi$. Hence, $\gamma, x' \vDash^{L_\beta} \psi$ and $\gamma \vDash^{L_\beta} \dot\exists\psi$.

- Assume $\varphi = (t_1 \doteq t_2)$. The claim follows trivially by unfolding definitions:

  $$(\gamma \vDash^{L_\alpha} t_1 \doteq t_2) \Leftrightarrow [\![t_1]\!]^{L_\alpha}_\gamma = [\![t_2]\!]^{L_\alpha}_\gamma \Leftrightarrow [\![t_1]\!]^{L}_\gamma = [\![t_2]\!]^{L}_\gamma \Leftrightarrow (\gamma \vDash^{L} t_1 \doteq t_2).$$

  In the second step, we use that $[\![t]\!]^{L_\alpha}_\gamma = [\![t]\!]^{L}_\gamma$ for all t, which holds by a simple case analysis.

- Assume $\varphi = (t_1 \mathbin{\dot\in} t_2)$. The proof is analogous to the previous case.

□

We proved the reflection theorem for L and the stages of L but we used only few of their properties. Typically, it is stated in a more general form that applies also to other kinds of hierarchies.

From the reflection theorem, we derive a crucial lemma that can immediately be applied to prove the set construction axioms where we have to show that a certain class is a set over L.

6.2.6 Lemma. *Fix a class $A \subseteq L$. If A is representable over L and a set in the outer model, then it is a set in L.*

Proof. Since A is representable over L, we obtain a formula φ over L, such that $((x) \vDash^L \varphi) \Leftrightarrow x \in A$ for all $x \in L$. Fix a stage L_α that contains all constants of φ and all elements of A. Then apply the reflection theorem to obtain a greater stage L_β that reflects L with respect to φ. Now φ is a formula over L_β and $((x) \vDash^{L_\beta} \varphi) \Leftrightarrow ((x) \vDash^L \varphi) \Leftrightarrow x \in A$ for all $x \in L_\beta$. That means, A is representable over L_β. Hence, A is an element of $L_{\beta+1}$ and of L. □

We need this lemma primarily to prove the axioms of separation and replacement, but it proves useful for the other axioms as well. We start with the simplest ones.

6.2.7 Lemma. *L satisfies the axioms of the empty set, pairing, union, the power set and infinity.*

Proof. We just prove the axiom of pairing in detail. The other ones are similar. Fix sets $x, y \in L$. To see that the class $\{x, y\}$ is a set over L, we apply the previous lemma. The class is representable over L by the predicate $\lambda z.\, z = x \vee z = y$. It is a set over the outer model by the axiom of pairing.

The axiom of infinity is the only one where the representation is not trivial. But the standard representation of $\bar{\mathbb{N}}$ that we defined in Lemma 5.6.14 is already applicable here since it only uses the other axioms that we prove in the current lemma. □

Note that the empty class, the ordered pair and the union over L are the same as over the outer model. This will be important as it implies that the empty ordinal, the ordinal successor and the limit of ordinals coincide between L and the outer model.

6.2.8 Lemma. *L satisfies the axioms of separation and replacement.*

Proof. We only consider separation. Replacement is analogous. Fix a set A in L and a representable predicate $P : A \to \mathbb{P}$. Then $\{x \in A \mid P(x)\}$ is trivially representable over L. By the previous lemma, it suffices to show that this class is a set in the outer model. Moreover, by the axiom of replacement, it suffices to show that P is representable in the outer model. If φ is the formula representing P over L then we get the representation over the outer model by $P = (\lambda x.\, (x) \vDash \varphi)$ and the representability of $\vDash^L$ (Lemma 5.6.26). □

6.2.9 Corollary. *L is a model of ZF'.*

Proof. L satisfies all the axioms by Lemma 6.2.1, Lemma 6.2.2, Lemma 6.2.7 and Lemma 6.2.8. □

6.3 L satisfies the axiom of constructibility

6.3.1 Definition. The **axiom of constructibility** states that the class of all sets is equal to L.

We will show that this statement holds over L. This knowledge is helpful in the next section where we establish a well-order on L. If L contains all sets then this implies the well-ordering theorem and, with it, the axiom of choice.

Since L is defined by recursion on the class of ordinals, it will be useful to know that the ordinals over L are exactly the ordinals over the outer model and that ordinal recursion over both models behaves equally. Then we will be able to show that every stage L_α^L is equal to the corresponding stage L_α over the outer model. We start with a first description of the ordinals over L.

6.3.2 Lemma. $\mathsf{Ord}^L = \mathsf{Ord} \cap L$

Proof. We show the equivalent statement that $x \in \mathsf{Ord}^L \Leftrightarrow x \in \mathsf{Ord}$ for all $x \in L$. We apply well-founded induction on x.

Assume that $x \in \mathsf{Ord}^L$. Then x is transitive over L and every element is an ordinal over L. By the inductive hypothesis, every element is then also an ordinal over the outer model. It suffices to show now that x is also transitive over the outer model. Fix arbitrary sets y and z over the outer model with $z \in y \in x$. By transitivity of L (Lemma 6.1.4), $y, z \in L$. By transitivity of x over L, we conclude that $z \in x$.

For the other direction, assume that $x \in \mathsf{Ord}$. Then x is transitive and every element is an ordinal. Transitivity over L follows trivially. Every element is an ordinal over L by the induction hypothesis. Hence, x is also an ordinal over L. □

We also need the same lemma for stages.

6.3.3 Lemma. $\mathsf{Ord}^{L_\alpha} = \mathsf{Ord} \cap L_\alpha$

Proof. Analogous to the previous lemma. □

Now it remains to show that every ordinal over the outer model is actually contained in L. We will see that $\alpha \in L_{\alpha+1}$ for every ordinal α. As an intermediate step, it seems necessary to examine exactly which ordinals are contained in every stage.

6.3.4 Lemma. $L_\alpha \cap \mathsf{Ord} = \alpha$

Proof. We prove the statement by induction on α.

- Assume $\alpha = \emptyset$. $L_\emptyset \cap \mathsf{Ord} = \emptyset \cap \mathsf{Ord} = \emptyset$.
- Assume $\alpha = \alpha' + 1$. Since every element of the ordinal α is an ordinal, it suffices to show that an arbitrary but fixed ordinal β satisfies $\beta \in L_\alpha \Leftrightarrow \beta \in \alpha$.

  Assume that $\beta \in L_{\alpha'+1}$. Then $\beta \subseteq L_{\alpha'}$. With the inductive hypothesis, we get that $\beta \subseteq L_{\alpha'} \subseteq \alpha$. Hence, $\beta \in \alpha + 1$ (Lemma 3.5.9).

  For the other direction, assume that $\beta \in \alpha + 1 = \alpha \cup \{\alpha\}$. If $\beta \in \alpha$ then $\beta \in L_\alpha \subseteq L_{\alpha+1}$ by the inductive hypothesis and Lemma 6.1.2. If $\beta \in \{\alpha\}$ then $\beta = \alpha = L_\alpha \cap \mathsf{Ord} = \mathsf{Ord}^{L_\alpha} \in L_{\alpha+1}$ by the inductive hypothesis, the previous lemma and representability of Ord^{L_α} over L_α.
- Assume that $\alpha = \bigcup_{\xi \in \lambda} \xi$ is a limit ordinal. Then we have

$$L_{\bigcup_{\xi\in\alpha}\xi} \cap \mathsf{Ord} = \Big(\bigcup_{\xi\in\alpha} L_\xi\Big) \cap \mathsf{Ord} = \bigcup_{\xi\in\alpha} L_\xi \cap \mathsf{Ord} \overset{\text{IH}}{=} \bigcup_{\xi\in\alpha} \xi = \alpha. \qquad \square$$

We trivially get the next lemma.

6.3.5 Lemma. *Every ordinal α is an element of $L_{\alpha+1}$.*

Proof. Fix an ordinal α. Then $\alpha \in \alpha + 1 = L_{\alpha+1} \cap \mathsf{Ord} \subseteq L_{\alpha+1}$ by the previous lemma. □

Now, we can finally prove the lemma that we have been working for.

6.3.6 Lemma. $\mathsf{Ord} = \mathsf{Ord}^L$

Proof. We already know that $\mathsf{Ord} \cap L = \mathsf{Ord}^L$ (Lemma 6.3.2). Hence, it suffices to show that every ordinal α is an element of L. This follows by the previous lemma: $\alpha \in L_{\alpha+1} \subseteq L$. □

For clarity, we identify Ord and Ord^L from now on, that is, we write Ord instead of Ord^L and we implicitly convert members of one to members of the other. Now, we show that the recursion principles for both coincide.

6.3.7 Lemma. $\mathsf{rec}_{\mathsf{Ord}} = \mathsf{rec}^L_{\mathsf{Ord}}$.

Proof. Fix

- a family $A : \mathsf{Ord} \to \mathsf{Type}$,
- a family $x : A(\emptyset)$,
- a function $f : \Pi_\alpha A(\alpha) \to A(\alpha^+)$,
- a function $g : \Pi_\lambda(\Pi_{\alpha:\lambda} A(\alpha)) \to A(\lambda)$ and
- an ordinal α.

We show that $\mathsf{rec}_{\mathsf{Ord}}(A, x, f, g, \alpha) = \mathsf{rec}^L_{\mathsf{Ord}}(A, x, f, g, \alpha)$ by ordinal induction on α.

- Assume $\alpha = \emptyset$. Then $\mathsf{rec}_{\mathsf{Ord}}(A, x, f, g, \emptyset) = x = \mathsf{rec}^L_{\mathsf{Ord}}(A, x, f, g, \emptyset)$.
- Assume $\alpha = \alpha' + 1$. Then

$$\begin{aligned}
&\mathsf{rec}_{\mathsf{Ord}}(A, x, f, g, \alpha' + 1) \\
&\quad = f(\mathsf{rec}_{\mathsf{Ord}}(A, x, f, g, \alpha')) \\
&\quad = f(\mathsf{rec}^L_{\mathsf{Ord}}(A, x, f, g, \alpha')) \\
&\quad = \mathsf{rec}^L_{\mathsf{Ord}}(A, x, f, g, \alpha' +^L 1) \\
&\quad = \mathsf{rec}^L_{\mathsf{Ord}}(A, x, f, g, \alpha' + 1).
\end{aligned}$$

- Assume that α is a limit ordinal. Then

$$\begin{aligned}
&\mathsf{rec}_{\mathsf{Ord}}(A, x, f, g, \alpha) \\
&\quad = g(\lambda\xi \in \alpha.\, \mathsf{rec}_{\mathsf{Ord}}(A, x, f, g, \xi)) \\
&\quad = g(\lambda\xi \in \alpha.\, \mathsf{rec}^L_{\mathsf{Ord}}(A, x, f, g, \xi)) \\
&\quad = \mathsf{rec}^L_{\mathsf{Ord}}(A, x, f, g, \alpha).
\end{aligned}$$

□

We used in this proof that the empty ordinal, the successor function and, hence, the notion of limit ordinals over L and the outer model coincide.

6.3.8 Lemma. *$L_\alpha = L^L_\alpha$ for all ordinals α.*

Proof. It is easy to check that the recursion instruction in the definition of the stages are the in L and the outer model.The lemma follows with the previous lemma. □

Now, we can prove the main result of this section.

6.3.9 Lemma. *L satisfies the axiom of constructibility, that is $L = L^L$.*

Proof. Fix $x \in L$ and an ordinal α, such that $x \in L_\alpha$. We know that α is also an ordinal over L (Lemma 6.3.6). With the previous lemma, we get that $x \in L_\alpha = L^L_\alpha \subseteq L^L$. The other direction is trivial. □

Often, the axiom of constructibility is stated as $V = L$ where V stands for the class of all sets.

6.4 L satisfies the axiom of choice

We will provide a representable well-ordering on L. The main work is to show that representable well-orderings are inherited from every stage to its successor. Then we can compose those well-orderings to a representable well-ordering of L. We can derive the well-ordering theorem and, with it, the axiom of choice. Since successor stages are defined from formulas, we will need the following lemma.

6.4.1 Lemma. *If a class-like type I has a representable well-order and every member of a family $F : I \to \mathsf{Class}$ has a representable well-order then $\bigcup_{i:I} F(i)$ has a representable well-order.*

Proof. For $x, y \in \bigcup_{i:I} F(i)$, let $i_x, i_y : I$ be minimal, such that $x \in F(i_x)$ and $y \in F(i_y)$. We define that $x < y$ if $i_x < i_y$ or $i_x = i_y$ and $x < y$ by the well-order on $F(i_x)$. It is routine to check that this is representable and a well-order. □

6.4.2 Lemma. *If a class A has a representable well-order then then $\mathsf{Formula}^A$ has a representable well-order.*

Proof. The easiest way to see this is from the representation $\overline{\mathsf{Formula}^A} = \bigcup_{n:\mathbb{N}} H^n(\emptyset)$ where H is defined as in Lemma 5.6.21. By the previous lemma, it suffices to show that the greater relation on $\mathbb{N}$ is a representable well-order and that every $H^n(\emptyset)$ has a representable well-order, which can be done by induction on n. We will not go into the details. □

6.4.3 Lemma. *Every stage of L has a representable well-order.*

Proof. We prove this by ordinal induction on the rank of the stage.

- $L_\emptyset = \emptyset$ has a trivial representable well-order.
- $L_{\alpha+1} = \{\{x \in L_\alpha \mid (x) \vDash \varphi\} \mid \varphi : \mathsf{Formula}^{L_\alpha}\} \leq \mathsf{Formula}^{L_\alpha}$. This set has a representable well-order by the induction hypothesis and the previous lemma.
- Fix a limit ordinal λ. By definition, $L_\lambda = \bigcup_{\alpha\in\lambda} L_\alpha$. This has a representable well-order by the induction hypothesis and Lemma 6.4.1. □

6.4.4 Lemma. *L has a representable well-order.*

Proof. This follows by the previous lemma and Lemma 6.4.1. □

6.4.5 Lemma. *The axiom of choice holds over L.*

Proof. By Lemma 6.3.9, we know that $L = L^L$. If we interpret the previous lemma over L, it states that L^L and, hence, L has a well-ordering that is representable over L. We can restrict this well-ordering to a well-order on any given set over L. This restricted well-ordering will also be representable. Thus, the well-ordering theorem holds over L and with it the axiom of choice. □

6.4.6 Corollary. *Semantic consistency of ZF' implies semantic consistency of ZF' with the axiom of choice.*

Proof. We assumed a model of ZF' and constructed L which satisfies all the axioms of ZF' and the axiom of choice. □

As a closing remark, one could use this to derive relative consistency of the axiom of choice over a pure first-order axiomatisation of ZF. The approach would be to cut out the externally well-founded fragment of ZF to obtain a model of ZF' as suggested in Section 5.1. L in that model is then a model of ZF with the axiom of choice. But since ZF' is stronger than the first-order axiomatisation of ZF, L is then also a model of this first-order axiomatisation.

7 Formalisation details

We formalised the proof of Sierpiński's theorem and the most interesting parts of the consistency of the axiom of choice in the Coq proof assistant. Both formalisations can be found at https://www.ps.uni-saarland.de/~rech/master.php. They were built independently from each other, starting from the axiom of both set theories. They share many basic ideas but differ in some details. They also differ significantly from the presentation in this thesis due to the technical limitations of Coq and the fact that the thesis was written afterwards and contains some improvements over the formalisation.

The formalisation of Sierpiński's theorem is divided into five files that are listed with their line counts in Table 7.1. Most names describe pretty well what they contain and how they relate to the chapters and sections of this thesis. The file `Basic_Set_Theory.v` contains the axiomatisation of ZF_2 and the encodings that we need for the proof. Apart from coercions, there is no mechanism in Coq to *identify* sets with classes or classes with the type of their elements. For that reason, we formulate all the class-operations explicitly on sets and formulate the set-construction axioms as explicit existence statements about sets.

We have a coercion from classes to types which allows us to write for example $A \to B$ to stand for the type of functions from a class A to a class B. Formally, elements of a class A are dependent pairs consisting of a set x and a proof that $x \in A$. By proof irrelevance, equality between such pairs is equivalent to equality between the x. We often need to turn elements of classes into sets and sets into elements. For the first direction, we have a simple coercion. To make the other direction more convenient, we treat the element-property with a type class and hide it as an implicit argument. When the element-property cannot be inferred automatically then we often use the command `Program Definition`. In proof scripts, we achieve a similar effect with tactics like `unshelve eexact`, `unshelve eapply` and `unshelve eexists`. They introduce the property as an existential variable and then unshelve that variable to turn it into a goal.

When the formalisation was created, it was not clear how useful encodings could be. For that reason, the formalisation does not make encodings explicit but defines classes or sets with similar operations as the corresponding type. For example, we have a set of numerals with constructors and a recursor on that set. One reason to avoid explicit encodings was the need for additional axioms. The formalisation assumes only the law of excluded middle and functional extensionality. In particular, there is no general axiom of definite description. We only have a description principle for sets which comes from the specific formulation of replacement. But without definite description, we have no way to map sets in a meaningful way to anything that is not a set. For example, we cannot define a bijection between the set of numerals and the type of natural numbers. More generally, the recursor on numerals can only be used to construct other sets.

spec	proof	
494	538	`Basic_Set_Theory.v`
256	348	`Cardinality.v`
170	327	`Ordinals.v`
21	86	`Hartogs_Number.v`
36	123	`Sierpinskis_Theorem.v`
1036	1422	total

Table 7.1: Sierpiński's theorem

spec	proof	
110	13	`ZF_Structures.v`
576	41	`Definability.v`
45	0	`Axioms.v`
175	138	`Basic_Constructions.v`
239	410	`L.v`
1145	602	total

Table 7.2: Consistency of AC

The formulations of constructors and recursors on sets do not posses any useful conversion properties. Computation has to be done by rewriting. For that purpose, we generally register computation rules with `autorewrite`. This tactic is also useful to apply the defining properties of sets or set operations like the union. The tactics `apply` and `auto` could be used but are less useful for that purpose since we often do it on subterms.

The formalisation of the consistency of AC is work-in-progress. The files and their current line-counts are listed in table 7.2. The formalisation uses the expression definability instead of representability. The automatic inference of the equivalent of representations schemes and representations relies on the mechanism for type class inference. Due to technical limitations in this mechanism, the representability framework is not implemented in the same form as we present it here. It was necessary to avoid different representations schemes on the same family. This implies that we cannot work with the representation schemes for types. The formalisation does not distinguish between local representations and full representations but has only one such concept. We use the same coercions from sets to classes, from classes to types and from elements to sets as in the formalisation of Sierpiński's theorem. A difference between both is that we assume the axiom of definite description and use explicit encodings for the formalisation of the consistency proof.

The formalisation contains many admits. Many of the basic representations from Section 5.3 and Section 5.6 are admitted. We have large parts of the proof of the reflection theorem and the fact that L is a model. The facts that L satisfies the axiom of constructibility and the axiom of choice are not formalised at all.

8 Related work

History of AC. The first proof of the independence of a version of AC is due to Frankel [7]. In 1922, he established the independence of AC over a certain set theory with atoms called urelements. However, his proof does not apply to ZF. The first proof that works on ZF is the one by Gödel [10] that we have seen in chapter 6. After that one, Gödel discovered another similar proof that he claims to be simpler [9]. Instead of L, it uses the class of *hereditarily ordinal definable* sets as inner model. Gödel always believed in the truth of CH and tried to deduce the hypothesis from more foundational assumptions, even after the independence over ZF was established by Cohen. He hoped that the existence of large cardinals would suffice. Indeed, the standard large cardinal axioms imply many similar but weaker statements. However, it turned out that the standard axioms that Gödel had in mind have no influence on the truth of CH [22]. This discovery seems to have mostly settled the efforts to decide the continuum problem.

Mizar. When it comes to formalisations inside set theory, a large body of such work can be found in the library of the proof assistant Mizar [12]. Mizar is based on Tarski-Grothendieck set theory, a non-conservative extension of ZF. The system contains first-order functions and predicates on sets as primitives. However, those are not treated as elements of the object logic. It is not allowed to quantify over them, except on a metalevel where the user can state generic definitions and theorems that serve as a kind of template and take functions or predicates as parameters. The lack of higher-order functions makes certain definitions difficult. For example, many definitions of recursive functions in the library are difficult to read since it is not possible to define a generic recursor. Instead, the instructions that would form the recursor, are inlined as boilerplate. There is a formalisation of the reflection theorem in Mizar by Bancerek [2].

Isabelle. The proof assistant Isabelle [31] offers a basic higher-order logic as foundation in which more powerful theories can be axiomatised. One of those theories which is Isabelle/ZF which is part of the standard library. It consists of a shallow embedding of first-order logic with the axioms of ZF inside. Higher-order functions are available and there is a convenient syntax for the definition of inductive or coinductive types and recursive functions. This syntax is not primitive, but defined from the axioms of ZF. The library contains, among others, many results about ordinals and cardinals and proofs of the equivalence between 20 formulations of AC and 7 formulations of the well-ordering theorem [24].

Unfortunately, Isabelle/ZF builds on a shallow embedding of first-order logic and Mizar does not have a metalogic at all. Hence, neither system allows any formal meta-reasoning. Nonetheless, Paulson produced a partial formalisation of the Montague-Levy reflection

theorem and the relative consistency of AC in Isabelle/ZF [23]. The formalisation covers only the internal steps that can be done inside ZF. There is no way to formalise the meta-reasoning without switching the foundation. The development contains, however, a mechanical procedure that produces instances of the reflection theorem for specific formulas.

Other related work in Isabelle includes a mechanical translation from higher-order logic to ZF by Krauss and Schropp [19]. Gordon explored how higher-order logic becomes more expressive if one assumes a model of ZF and outlined his idea to model higher-order logic in ZF [11]

Lean. A recent contribution comes with the Flypitch project by Han and van Doorn [13]. The project consists of a full formalisation of the independence of CH over ZF with the axiom of choice in the Lean theorem prover. The relative consistency of ¬CH is proved by Cohen forcing and the relative consistency of CH follows by a σ-closed forcing instead of the classical approaches through constructibility. This way, one can avoid extensive reasoning inside the set theory.

Coq. There is also notable work in type theory: Werner constructed models of the calculus of inductive constructions in ZF and vice versa [32]. Barras worked in the Coq proof assistant to explore set-theoretic models of fragments of the calculus of inductive constructions [3]. Recently, Kirst formalised a categoricity result for ZF_2 and constructed models for ZF_2 in Coq [17, 18]. Forster, Kirst and Wehr presented a formalisation of the completeness theorem for first-order logic in Coq [6].

9 Conclusion

We began this thesis by introducing an axiomatisation of the set theory ZF_2 in Coq. The axioms were inspired by the widely-used Zermelo-Fraenkel set theory but modified for more convenience. Most importantly, we chose a second-order axiomatisation, such that we could use arbitrary functions and predicates for replacement and separation. But we also chose versions of the axioms of foundation and infinity that take full advantage of the capabilities of Coq to make the statements and their application as natural as possible. We introduced the concept of encodings to build a connection between set-theoretic constructions like numerals or the Cartesian product on one hand and, on the other hand, their counterparts in the existing infrastructure of Coq, like natural numbers and the product of types. This allows us to mix set-theoretic definitions seamlessly with standard formulations in Coq.

We continued with the proof of Sierpiński's theorem. We developed some basic and some more specialised results about cardinality. At this point, it already appeared valuable that we have this link between set-theory and the metalogic. Technically, we can work with the convenience of inductive types and recursive definitions in Coq and the conversion mechanism and, in principle, we can reuse pre-existing results that are not specific to sets. In contrast to many informal presentations, we did not make cardinal numbers explicit. Those are unnecessary for our proofs and would only increase the complexity.

The next step was the introduction of some ordinal theory and, most importantly, the Hartogs number. In that chapter, we used ordinals solely in their function as representatives of well-orderings up to isomorphism. In the same way, in which we avoided to use cardinal numbers, it would likely be possible to avoid ordinals and just talk about well-ordered sets instead. But there are two reasons why one should still make the concept of ordinals explicit. First, it seems more understandable to talk about one concrete ordinal than about some unknown well-ordered set that only satisfies certain properties. This is most apparent when it comes to the Hartogs number. Second, we use ordinals for the definition of L, so it doesn't cause much overhead to apply them here, as well. Note that ordinals are the only set-theoretic construction that we do not treat as encodings of some other objects.

The proof of Sierpiński's theorem relies on the law of excluded middle in two places: We use it on a low level to prove that isomorphic ordinals are equal (Lemma 3.5.11) and to prove Lemma 4.2.4 where we first apply Cantor's theorem to prove that a specific function is not surjective and then obtain non-constructively an element that is not in the image. Moreover, we use classic reasoning in multiple places to establish properties of basic constructions. It seems plausible that Sierpiński's theorem can be translated to an intuitionistic setting. We do not care about isomorphic ordinals if we do not introduce

ordinals as unique representatives at all. Moreover, one can prove a stronger version of Cantor's theorem that does not only refute the existence of a surjection but gives us a witness for the non-surjectivity of functions. Finally, we can do the basic constructions without classical reasoning if we build on and intuitionistic foundation. This might be an interesting topic for future work.

For the second part of our project, we needed to switch to the set theory ZF' because it makes weaker assumptions and gives us more flexibility for the construction of models. Most axioms stayed the same as before. The only difference lies in the axioms of replacement and separation. Instead of arbitrary functions and relations on sets, we only accept those that are given by a first-order formula. The axiomatisation is still not equivalent to pure first-order ZF because the axioms of foundation and infinity allow us to use $\in$-induction and natural induction for arbitrary predicates, not just first-order formulas, as they should. We chose those strong formulations for foundation and infinity because they are necessary to keep the notion of encodings as bijection that we used before. The relative consistency of AC is still provable in this system. Alternatively, one could choose to switch to the traditional first-order axioms and relax the notion of encodings. The result seems equally useful for our purposes.

With the new axiomatisation, our work became significantly more difficult. Whenever we want to apply replacement or separation (or forms of induction in a pure first-order axiomatisation), we need to provide an appropriate formula. To make this step straightforward, we introduced the notion of first-order representations and refined it iteratively to a point where we can use almost any function or predicate for separation and replacement and construct their representation afterwards in a compositional and often automated fashion. Talking about representations could be quite simple if we worked only with objects that we can encode as sets. In that case, the representability of sets would be the only necessary notion. But for the consistency proof of AC, we need to talk about proper classes, functions on proper classes, higher-order functions on those, and other objects that make the formulation of representability more difficult.

'Representation' is a simple word that seems to represent a natural concept, but making formal what a first-order representation means in a general context, is surprisingly difficult. In our treatment, we stated some conditions on representations and produced some instances, but those are just an approximation of the general idea of representability. It would be interesting to get a better understanding of the concept, which might lead to a simpler definition. At the current state, it seems like there could be other definitions of representability that work equally well.

Technical considerations that drove many decisions behind our concrete framework were the ability to extend the notion of representability to new types and the intention to automate the construction of representations using type class inference. However, it turns out that this approach is limiting. The automation can fail when there are two representation schemes on the same type family that are not equal by conversion. Moreover, type class inference in Coq is unstructured and difficult to debug and it is easy to run into performance problems Generally, the performance should be expected to become worse, as the user adds more instances to the database. An ideal solution would be to implement a custom tactic for the construction of representations that accesses a

custom database of existing representations.

To make the treatment of representations clearer, it could help to switch to a metatheoretical level and talk about Coq values as terms that can have free variables and refer to an environment. This could be done using MetaCoq [27]. That viewpoint seems to be the original inspiration behind our definitions. One could say that our current framework does two things at once: It reifies values, then it translates their representation into first-order logic. One should try to uncouple those aspects.

To summarise, we need representability because we talk about the metatheory of ZF which requires a deep embedding that builds on formulas, but we also need to reason inside of ZF and want to do this in a convenient way without writing first-order formulas by hand or redefining the things that we already have in Coq. It is this mix of internal and external reasoning that distinguishes our project from many previous formalisations and makes a structured approach like our representability framework desirable.

Future work. The obvious next step would be to finish the proof for the relative consistency of the axiom of choice. Using the representation framework and following the approach in chapter 6, that should be routine. One could continue to prove relative consistency of the generalised continuum hypothesis, which can also be done from the axiom of constructibility [26]. Moreover, one could apply the representability framework to produce formalisations of more parts of set theory, first-order logic, Peano arithmetic, or even other logical systems.

Another interesting next step would be to prove a type-theoretic version of Sierpiński's theorem. Homotopy type theory [29] seems like a good basis for such a proof due to its extensionality principles on functions and types and its flexible notion of mere propositions. In homotopy type theory, one can formulate an axiom of choice and a generalised continuum hypothesis that talk about types instead of sets but express the same ideas as their set-theoretic counterpart. It seems plausible that one of the proof can be done in such a setting without non-standard axioms. In particular, we expect that one can do the proof without the law of excluded middle. Most applications of this law in our proof are only necessary to handle set-theoretic encodings of ordinals and other objects. In an intuitionistic foundation, they might not be necessary. If one does assume the law of excluded middle then it should be straight-forward to translate our formalisation into homotopy type theory.

Our notion of representability seems powerful enough to admit representation schemes on all families of types on which we might want to talk about representations. But it would be interesting to understand the notion better, to find better explanations, to understand the limits and maybe to extend them. It appears more natural to formulate representability on a metatheoretic level where we can talk about Coq terms with free variables and Coq terms have the same status as first-order formulas. The mechanisation would likely also profit from such an approach. As explained in Section 5.5, there are limitations to our current approach. Some of them can be overcome by switching from standard type class inference to a specialised automation tactic. Others can be overcome by such a switch to a meta-level. MetaCoq [27] could be useful for this. In general, the

further development of representability seems like a rather open-ended question and it is not clear what one can achieve.

Bibliography

[1] Stefan Banach and Alfred Tarski. "Sur la décomposition des ensembles de points en parties respectivement congruentes". In: *Fund. math* 6.1 (1924), pp. 244–277.

[2] Grzegor Bancerek. "The reflection theorem". In: *Journal of Formalized Mathematics* 2 (1990). URL: http://mizar.org/JFM/Vol2/zf_refle.html.

[3] Bruno Barras. "Sets in Coq, Coq in sets". In: *Journal of Formalized Reasoning* 3.1 (2010), pp. 29–48.

[4] Georg Cantor. *Über eine elementare Frage der Mannigfaltigkeitslehre*. Druck und Verlag von Georg Reimer, 1892.

[5] Nicolaas Govert De Bruijn. "A survey of the project AUTOMATH". In: *Studies in Logic and the Foundations of Mathematics*. Vol. 133. Elsevier, 1994, pp. 141–161.

[6] Yannick Forster, Dominik Kirst and Dominik Wehr. "Completeness Theorems for First-Order Logic Analysed in Constructive Type Theory". In: *International Symposium on Logical Foundations of Computer Science*. Springer. 2020, pp. 47–74.

[7] Abraham Fraenkel. "Über den Begriff 'definit' und die Unabhängigkeit des Auswahlaxioms". In: *Sitzungsberichte der Preussischen Akademie der Wissenschaften, Physikalisch-mathematische Klasse* (1922), pp. 253–257.

[8] Leonard Gillman. "Two classical surprises concerning the axiom of choice and the continuum hypothesis". In: *The American Mathematical Monthly* 109.6 (2002), pp. 544–553.

[9] Kurt Gödel. "Remarks before the Princeton bicentennial conference on problems in mathematics". In: *The Undecidable*. Ed. by Martin Davis. 1964, pp. 84–88.

[10] Kurt Gödel. "The consistency of the axiom of choice and of the generalized continuum-hypothesis". In: *Proceedings of the National Academy of Sciences of the United States of America* 24.12 (1938), p. 556.

[11] Mike Gordon. "Set theory, higher order logic or both?" In: *International Conference on Theorem Proving in Higher Order Logics*. Springer. 1996, pp. 191–201.

[12] Adam Grabowski, Artur Korniłowicz and Adam Naumowicz. "Four decades of Mizar". In: *Journal of Automated Reasoning* 55.3 (2015), pp. 191–198.

[13] Jesse Michael Han and Floris van Doorn. "A formal proof of the independence of the continuum hypothesis". In: *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs*. 2020, pp. 353–366.

[14] Friedrich Hartogs. "Über das Problem der Wohlordnung". In: *Mathematische Annalen* 76.4 (1915), pp. 438–443.

[15] David Hilbert. “Mathematical problems”. In: *Bulletin of the American Mathematical Society* 8.10 (1902), pp. 437–479.

[16] Thomas J Jech. “About the axiom of choice”. In: *Handbook of mathematical logic* 90 (1977), pp. 345–370.

[17] Dominik Kirst and Gert Smolka. “Categoricity results for second-order ZF in dependent type theory”. In: *International Conference on Interactive Theorem Proving*. Springer. 2017, pp. 304–318.

[18] Dominik Kirst and Gert Smolka. “Large model constructions for second-order ZF in dependent type theory”. In: *Proceedings of the 7th ACM SIGPLAN International Conference on Certified Programs and Proofs*. 2018, pp. 228–239.

[19] Alexander Krauss and Andreas Schropp. “A mechanized translation from higher-order logic to set theory”. In: *International Conference on Interactive Theorem Proving*. Springer. 2010, pp. 323–338.

[20] Casimir Kuratowski. “Sur la notion de l’ordre dans la théorie des ensembles”. In: *Fundamenta mathematicae* 2.1 (1921), pp. 161–171.

[21] Daniel Leivant. “Higher order logic.” In: *Handbook of Logic in Artificial Intelligence and Logic Programming*. Vol. 2. Oxford University Press, 1994, pp. 229–321.

[22] Azriel Lévy and Robert M Solovay. “Measurable cardinals and the continuum hypothesis”. In: *Israel Journal of Mathematics* 5.4 (1967), pp. 234–248.

[23] Lawrence C Paulson. “The Relative Consistency of the Axiom of Choice Mechanized Using Isabelle/ zf”. In: *LMS Journal of Computation and Mathematics* 6 (2003), pp. 198–248.

[24] Lawrence C Paulson and Krzysztof Grabczewski. “Mechanizing set theory”. In: *Journal of Automated Reasoning* 17.3 (1996), pp. 291–323.

[25] Wacław Sierpiński. “L’hypothèse généralisée du continu et l’axiome du choix”. In: *Fundamenta Mathematicae* 1.34 (1947), pp. 1–5.

[26] Raymond M. Smullyan and Melvin Fitting. *Set theory and the continuum problem*. Dover Publications, 2010.

[27] Matthieu Sozeau et al. “The MetaCoq Project”. In: *Journal of Automated Reasoning* (2020), pp. 1–53.

[28] The Coq Development Team. *The Coq Proof Assistant*. Version 8.9.0. Zenodo, 2019. DOI: 10.5281/zenodo.2554024.

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

[30] John Von Neumann. “Zur einführung der transfiniten zahlen”. In: *Acta Litterarum ac Scientiarum Regiae Universitatis Hungaricae Francisco-Josephinae, sectio scientiarum mathematicarum* 1 (1923), pp. 199–208.

[31] Makarius Wenzel, Lawrence C Paulson and Tobias Nipkow. "The isabelle framework". In: *International Conference on Theorem Proving in Higher Order Logics*. Springer. 2008, pp. 33–38.

[32] Benjamin Werner. "Sets in types, types in sets". In: *Theoretical Aspects of Computer Software*. 1997, pp. 530–546.

[33] Ernst Zermelo. "Beweis, daß jede Menge wohlgeordnet werden kann". In: *Mathematische Annalen* 59.4 (1904), pp. 514–516.

[34] Ernst Zermelo. "Über grenzzahlen und mengenbereiche: Neue untersuchungen über die grundlagen der mengenlehre". In: *Fundamenta Mathematicæ* 16 (1930), pp. 29–47.