Saarland University
Faculty of Natural Sciences and Technology I
Department of Computer Science

Masters Thesis

Formal Construction of a Set Theory in Coq

submitted by

Jonas Kaiser

on November 23, 2012

Supervisor

Prof. Dr. Gert Smolka

Advisor

Dr. Chad E. Brown

Reviewers

Prof. Dr. Gert Smolka

Dr. Chad E. Brown

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.

Saarbrücken, _____ _____

(Datum/Date) (Unterschrift/Signature)

Acknowledgements

First of all I would like to express my sincerest gratitude towards my advisor, Chad Brown, who supported me throughout this work. His extensive knowledge and insights opened my eyes to the beauty of axiomatic set theory and foundational mathematics. We spent many hours discussing the minute details of the various constructions and he taught me the importance of mathematical rigour.

Equally important was the support of my supervisor, Prof. Smolka, who first introduced me to the topic and was there whenever a question arose. During his course on formal logic I first experienced the delight of working with the proof assistant Coq.

I am also massively grateful for the countless more or less formal discussions with my colleagues at the Programming Systems Lab and throughout the university. They helped to keep up my spirits over the longer stretches and provided valuable second perspectives on my work.

Last but not least I would like to thank my friends and family for their unwavering support throughout the writing of this thesis.

Abstract

In this thesis we present a formalisation of Tarski-Grothendieck set theory, that is, Zermelo-Fraenkel set theory with Grothendieck universes. The development is executed in Coq, extended with the law of the excluded middle and a powerful choice principle. This yields a powerful metatheory where the inhabitance of every type is decidable. From the axiomatisation we develop the usual theory of sets, including restricted comprehension, ordered pairs, functions and finite ordinals.

The formalisation is designed to support the construction of proof-irrelevant, classical models for type theories like the Calculus of (co)inductive Constructions. The actual model constructions are left for future work.

Contents

CHAPTER 1

INTRODUCTION

The topic of this thesis is the formalisation of a set theory which admits the construction of set-theoretic models for type theories.

Both *type theory* as well as *set theory* concern themselves with the foundations of logic and mathematics albeit from rather different points of view. Before we introduce the material of this work in any detail it seems prudent to briefly survey the developments, formalisms and philosophies of the two camps. This overview is certainly not exhaustive and there is a wealth of secondary literature available for the interested reader. Our intention is simply to provide a small amount of historical context for the developments presented herein.

Set theory can be traced back as far as 1874 to the work of Cantor [Can74] and the mantra of this camp is "everything is a set". All the constructions familiar from mathematics can be built from sets and all operations on them can be encoded as operations on sets. A major achievement in this line of work was Zermelo's first axiomatisation in 1908 [Zer08]. Fuelled by this result, the first third of the 20^{th} century saw a lot of work around the construction of the best possible axiomatisation. The goal was to maximise the expressiveness of the theory without introducing any of the known inconsistencies like the Russell set [Rus02], and with as few axioms as possible. Beyond these mutually agreed upon criteria there was a lot of debate on which group of axioms was the "right" one. This debate and the studies it entailed led to the extensive knowledge we posses today about the relative strengths and weaknesses of the various axiomatisations. A set theory that stood the test of time and is still in use today is *Zermelo-Fraenkel set theory with the Axiom of Choice* (ZFC). It is a cleaned-up variant of Zermelo's original work with a major modification introduced by Fraenkel in 1922 [Fra22]. The axiom of

choice, although already part of Zermelo's original collection of axioms, is mentioned explicitly since it tends to be the most controversial one. Accepting it into the theory has far reaching consequences and leads to rather unexpected effects, e.g., consider the Banach-Tarski paradox [Wag85]. When people speak about set theory without explicitly mentioning an axiomatisation they usually have ZFC in mind. In 1938 Tarski augmented ZFC with the assumption that there exist large inaccessible cardinals [Tar38] and later Grothendieck formulated a set-theoretic notion of universes, which when added to ZFC provided a theory equivalent to Tarski's variant, albeit without the need to explicitly talk about the concept of cardinality [Gab62, Kru65, GV72]. The resulting theory is called *Tarski-Grothendieck* set theory (TG) and its formalisation makes up the bulk of this work.

Type theory on the other hand mostly focuses on the foundation of logic and concerns itself with the question of "what exactly is a proof". This question is of course as old as logic itself, but new answers began to appear after Brouwer challenged the justification for the *law of the excluded middle* [Bro23] and introduced intuitionism. Based on this, Kolmogorov and Heyting investigated the nature of proofs further [Kol32, Hey74] and thus developed what is now known as the constructive or intuitionistic approach to logic. Its key feature is the absence of the law of the excluded middle. The basic idea is that to give a proof one has to explicitly construct a proof object, e.g., for a (binary) conjunction one has to provide a pair of proof objects, one for each conjunct. The most influential insight was that the proof of an implication is a procedure that transforms any proof object for the antecedent into a proof object for the conclusion, i.e., essentially a function on proof objects. What makes this development interesting is the close correspondence between intuitionistic logic and typed, computational calculi, like the simply typed λ-calculus or Girard's System F [GTL89]. The *Curry-Howard correspondence* [CF58, How80] or more generally the *Propositions as Types principle* states that theorems can be seen as types and proofs as terms. Then stating that a term has a type is essentially the same as presenting the proof of a theorem. In other words, a theorem is proved by the proofs that inhabit it. The use of typed, functional calculi as a logic is then the core doctrine of *type theory*. There are other type theories next to System F, like Coquand's and Huet's Calculus of Constructions (CC) [CH85, CH88] or Martin-Löf's predicative Type Theory (MLTT) [ML75]. Ideas taken from all of these resulted in Luo's Extended Calculus of Constructions (ECC) [Luo94]. Then with the addition of inductive types and further modifications we arrive at the Calculus of (co)inductive Constructions (CiC) [PPM89]. It is the logic implemented in the proof assistant Coq[1], which has been used for several extensive, machine-verified proofs, not least among which are a fully verified C-compiler [BDL06, Ler09b, Ler09a] and the four colour theorem [Gon07].

It is natural to ask how these two distinct approaches relate to each other. To answer this question one usually takes one formalism and constructs a model for another. Given that there are many formalisms and various kinds of models this

[1] A recent version and associated documentation can be found at `http://coq.inria.fr/`

is admittedly a rather broad statement. To make this more precise we have to ask ourselves what exactly we want to deduce from the construction of the model and the answer will fall into one of two categories.

The first thing we may ask is: *What is the proof-theoretic strength of one formalism relative to another*, i.e., is formalism *A* strong enough to model formalism *B*. These kinds of models are useful to investigate the relative consistency of a formalism and in general one has to be very careful about the assumptions and axioms used throughout the model construction.

Since the models which aim at proof-theoretic strength tend to be as minimal as possible, they are usually ill suited to provide an intuition for, say, the set-theoretic semantics of a type theory, or vice versa. To achieve the latter it is common to introduce additional assumptions with the hope to construct a simple model with easy and straightforward definitions. These are the second kind of models and to some degree one can still reason about consistency but the obtained answers are not as strong as with the first kind.

A lot of work has been conduct in both of these areas. A good example for a comparison of the first kind is the rigorous work of Aczel [Acz98] as well as the work of Werner [Wer97] which investigate the relation between the predicative type universes of CiC and ZFC with inaccessible cardinals. The latter is partially formalised in Coq. Barras' development [Bar10], also formalised in Coq, on the other hand is a prime example for the second kind of investigation. There are of course a lot more and we will see a few of them in the section on related work.

Why do we care about the formalisation of such model constructions? One benefit is that we cannot gloss over "supposedly trivial" sections of the development which can bring unforeseen and subtle problems in existing semantics to light. A particular such case was revealed by Miquel and Werner [MW02]. With respect to "simple" proof-irrelevant models (where the model interprets propositions with one of exactly two objects) of CC they observed that the combination of impredicativity and subtyping led to unforeseen complications. Lee and Werner present a partial solution to this issue, which relies on an altered notion of equality [LW11].

Motivated by the recent work of Barras [Bar10, Bar12] we formalise Tarski-Grothendieck set theory with the goal to eventually construct proof-irrelevant models for the type theories ECC and later CiC. In his development Barras works with a formalisation of *intuitionistic Zermelo-Fraenkel set theory* (IZF) in Coq and constructs models of CiC. We deviate from his work in that we not only use ZFC instead of IZF but we assume a classical metatheory right from the start. Our metatheory is of course considerably stronger than the one Barras is using. We hope that this will enable us to construct a simple and easy to understand, classical model. In both cases one may wonder at the inherent circularity of the approach, i.e., we take an implementation of CiC and therein formalise a set theory which in turn is used to model CiC. To this we answer that the scenario is only problematic

if we were to make claims with respect to the proof-theoretic strength of the modelled type theories. Since this is not our goal we are not considering the issue any further.

The proof-irrelevant, classical models we want to construct should allow us to reason about the mutual consistency of Coq's standard library. This means that we want to construct models where the axioms found in the library are interpreted by *inhabited* sets. The actual model constructions are not part of the present text but are left for future work. This main purpose is relevant though since it guides several of the design decisions we make along the way.

From a logical point of view we start with a reasonably well understood intuitionistic type theory, namely CiC. This type theory is then consistently extended to a classical logic with the law of the excluded middle and then equipped with a strong choice principle. In combination this yields a fairly powerful metatheory, which by no means should be called computational anymore (the inhabitance of every type is decidable). The next major step is to formally introduce the axioms of TG, which adds an internally untyped set theory into our framework. On top of this machinery we then perform the development of the remainder of the set theory.

The remainder of the document is structured as follows. In Chapter 2 we take a closer look at the metatheory underlying our development. Since we are working with the Coq proof assistant, this will include a brief discussion of the type theories CiC and ECC. We will also consider Coq's basic logic together with the addition of the law of the excluded middle and a choice axiom. The chapter concludes with a discussion of the consequences of combining these two additional axioms.

In Chapter 3 we then motivate and present our axiomatisation of Tarski-Grothendieck set theory, both mathematically and on the level of our Coq formalisation. Some of the more interesting axioms are also discussed in the light of historical context.

With the axiomatisation laid out, Chapter 4 then develops further set-theoretic constructions of which some are just generally useful, while others are provided and designed with respect to the main purpose of this development, namely the construction of models for a number of type theories. Of particular interest here are Section 4.6 on the Aczel function encoding as well as Section 4.7 which provides a construction of finite ordinals.

Finally, Chapter 5 concludes this work with a discussion of related efforts in this field and a brief outline of how we intend to make use of this formalisation in future developments.

CHAPTER 2

METATHEORY

In this thesis we are going to provide a formal development of a set theory. For this statement to make any sense at all we first have to clearly pin down the metatheory within which we will be working. This will allow us to put the development in context. We will later also discuss issues of consistency and for any reasonably expressive system we know that such arguments will always be of a relative nature. This provides another justification for making the background logic precise.

We will be formalising the development with the proof assistant Coq which implements the *Calculus of (co)inductive Constructions* (CiC for short). This logic is an *intuitionistic type theory* and we will discuss it first. We will augment this basic system with two additional axioms so that we effectively work with a fairly powerful, classical logic.

The presentation of CiC serves another purpose as well. As we pointed out in the introduction we design our set theory for a very particular purpose, namely the construction of a formal, set-theoretic model for exactly this calculus. Hence the following presentation can also be understood as a "design guide" for the constructs and properties we eventually expect from our set theory. Especially the formal definition of the ECC fragment of CiC's typing relation is instructive.

2.1 CiC and ECC

The Calculus of (co)inductive Constructions is a logic build around the *propositions as types* principle. As such the Calculus is more often referred to as a *type theory*. CiC is built from Luo's Extended Calculus of Constructions [Luo94] (hereafter ECC) by adding inductive and coinductive types [PPM89]. We will only discuss Luo's presentation of ECC in any detail since the added power of inductive types is only marginally used. To be precise, we are going to use only the inductive proposition inhabited T, strong sums and the inductive type of natural numbers, nat. Technically, the last could be avoided but it makes our life easier in the final part of the development, while the strong sums are already part of ECC, albeit not as an inductive type. ECC provides us with the following language of *terms*:

- The constants **Prop** and $\mathbf{Type}_0$, $\mathbf{Type}_1$, $\mathbf{Type}_2$, ... are terms, called *universes*, where the subscript of $\mathbf{Type}_i$ is referred to as the *universe index*
- *Variables* $(x, y, \ldots)$ are terms.
- Let M, N, A and B be terms, then each of the following is a term:
  - $\Pi x : A,\ B$, called *dependent product*
  - $\lambda x : A,\ N$, called *abstraction*
  - $M\ N$, called *application*
  - $\Sigma x : A,\ B$, called *strong sum*
  - $\mathbf{pair}_{\Sigma x:A,\ B}(M, N)$, called *pair*
  - $\pi_1 M$, called *first projection*
  - $\pi_2 M$, called *second projection*

In contrast to earlier theories (e.g., the simply typed lambda calculus or Girard's System F [GTL89]) the calculus does not make a syntactic distinction between *terms* and *types*, which means that terms are used to play both roles. The theory is equipped with a ternary typing judgement of the form $\Gamma \vdash a : A$, which ascribes the term A as a *type* to the term a under the typing environment Γ (when Γ is empty we simply write $\vdash a : A$). The judgement is build up inductively from the rules in Figure 2.1 and the rules make use of a subtyping relation ($\preceq$). The most important fact about this relation is that it provides a total order for the universes: $\mathbf{Prop} \preceq \mathbf{Type}_0 \preceq \mathbf{Type}_1 \ldots$ The remaining details can be found in Luo's book [Luo94].

At its heart, the Coq proof assistant is simply a type checker that confirms or rejects the existence of such judgement triples with respect to the typing rules of CiC. Practically we are also provided with a number of so-called *tactics* which guide and simplify the construction of terms inhabiting certain other terms.

$$\textsc{Ax}\ \frac{}{\vdash \textbf{Prop} : \textbf{Type}_0} \qquad \textsc{C}\ \frac{\Gamma \vdash A : \textbf{Type}_j}{\Gamma,\ x : A \vdash \textbf{Prop} : \textbf{Type}_0}\ (x \notin FV(\Gamma))$$

$$\textsc{T}\ \frac{\Gamma \vdash \textbf{Prop} : \textbf{Type}_0}{\Gamma \vdash \textbf{Type}_j : \textbf{Type}_{j+1}} \qquad \textsc{Var}\ \frac{\Gamma,\ x : A,\ \Gamma' \vdash \textbf{Prop} : \textbf{Type}_0}{\Gamma,\ x : A,\ \Gamma' \vdash x : A}$$

$$\textsc{Impred}\ \frac{\Gamma,\ x : A \vdash B : \textbf{Prop}}{\Gamma \vdash \Pi x : A,\ B : \textbf{Prop}} \qquad \textsc{Prod}\ \frac{\Gamma \vdash A : \textbf{Type}_j \qquad \Gamma,\ x : A \vdash B : \textbf{Type}_j}{\Gamma \vdash \Pi x : A,\ B : \textbf{Type}_j}$$

$$\textsc{Lam}\ \frac{\Gamma,\ x : A \vdash M : B}{\Gamma \vdash \lambda x : A.\ M : \Pi x : A,\ B} \qquad \textsc{App}\ \frac{\Gamma \vdash M : \Pi x : A,\ B \qquad \Gamma \vdash N : A}{\Gamma \vdash M\ N : [N/x]B}$$

$$\textsc{Sigma}\ \frac{\Gamma \vdash A : \textbf{Type}_j \qquad \Gamma,\ x : A \vdash B : \textbf{Type}_j}{\Gamma \vdash \Sigma x : A,\ B : \textbf{Type}_j}$$

$$\textsc{Pair}\ \frac{\Gamma \vdash M : A \qquad \Gamma \vdash N : [M/x]B \qquad \Gamma,\ x : A \vdash B : \textbf{Type}_j}{\Gamma \vdash \langle M, N \rangle_{\Sigma x:A,\ B} : \Sigma x : A,\ B}$$

$$\textsc{Proj1}\ \frac{\Gamma \vdash M : \Sigma x : A,\ B}{\Gamma \vdash \pi_1(M) : A} \qquad \textsc{Proj2}\ \frac{\Gamma \vdash M : \Sigma x : A,\ B}{\Gamma \vdash \pi_2(M) : [\pi_1(M)/x]B}$$

$$\textsc{Sub}\ \frac{\Gamma \vdash M : A \qquad \Gamma \vdash A' : \textbf{Type}_j}{\Gamma \vdash M : A'}\ (A \preceq A')$$

Figure 2.1: *The typing rules of Luo's ECC.*

With this understanding of the basic machinery of Coq we are going to make precise the terminology we will use. So far we have only seen terms (and of course contexts) while the notion of a *type* was only mentioned loosely. So what exactly is a *type*? We are going to call a term A a *type* if there exists a universe index i such that the judgement $\vdash A : \textbf{Type}_i$ holds. Similarly we are going to call a term A a *proposition* if the judgement $\vdash A : \textbf{Prop}$ holds. From the subtyping rules it follows that a proposition is also a type, albeit a rather special one. When we say something is a type we usually mean a *non-propositional* type, although we highlight the distinction where necessary.

It is instructive to consider how these notions of type theory are related to concepts found in standard mathematics. There we have a number of objects like 0, 1 or **true** and **false**. These objects are grouped together to form something that is loosely called a set or class, with varying degrees of formalism behind the respect-

ive expressions. We encounter things like "the set of natural numbers" or "**false** is a boolean value" and would like to make these notions precise.

This is achieved with types. As an example we consider the type of natural numbers $\mathbb{N}$, i.e., we have $\vdash \mathbb{N} : \mathbf{Type}_0$, which allows us to formally state that 0 is a natural number as $\vdash 0 : \mathbb{N}$. Similarly we have $\vdash \mathbb{B} : \mathbf{Type}_0$ for boolean values and $\vdash \mathbf{true} : \mathbb{B}$.

Apart from these objects we are usually faced with statements or, to be precise, properties of these objects, e.g., $5 = 7$ or $\mathbf{true} \vee \mathbf{false}$, which may or may not be provable.

In our system these statements are those special *types* we call *propositions* and their inhabitants are the corresponding proofs. A proposition that does not hold is of course empty, i.e., there are no inhabitants that act as proofs. So we have the formal judgements $\vdash \mathbf{true} \vee \mathbf{false} : \mathbf{Prop}$ and $\vdash p : \mathbf{true} \vee \mathbf{false}$ (where p is a proof of the disjunction) and $\vdash 5 = 7 : \mathbf{Prop}$ but $\nvdash q : 5 = 7$ for any q.

As a consequence there are two separate ways to interpret the two terms in the judgement $\vdash a : A$ depending on whether $\vdash A : \mathbf{Prop}$ or $\vdash A : \mathbf{Type}_i$ (one of these has to be the case, since otherwise $\vdash a : A$ would not hold). In the first case we read A as some mathematical statement and a as a proof of this statement. In the second case it makes more sense to understand A as a class of mathematical objects and a as one particular such object.

We will illustrate this distinction with a small example. When working with Coq we are often talking about the inhabitance of types, i.e., we are dealing with the question if some term s exists such that $\vdash s : T$ for a given *type* T. Now T is a *type* but the statement about its inhabitance is a property of that type and should be seen as a *proposition*. It is something we may be able to prove. Coq uses an *inductive proposition* to encode this:

```
Inductive inhabited (A : Type) : Prop := inhabits : A → inhabited A.
```

So if we actually have a particular member s then the fact that T is inhabited is expressed in Coq with the judgement $s : T \vdash \mathsf{inhabits}\ s : \mathsf{inhabited}\ T$, which should be read as "s is an *object* in the *type* T and therefore inhabits s is the *proof* of the *proposition* inhabited T".

While this distinction is mostly cosmetic from a mathematical point of view, it leads to very real consequence in the context of type theories like those we have in Coq. At the heart of the problem lies the way implications or respectively function spaces are expected to behave at **Prop** and at $\mathbf{Type}_i$. In type theory, implications and function spaces are the same thing. The problem though is that for any $\vdash B : \mathbf{Prop}$ we expect $\vdash A \to B : \mathbf{Prop}$. This property is called *impredicativity* and for a single universe this is perfectly fine but when we extend this to any of the other universes the system is known to become inconsistent [Gir72, Hur95].

Another aspect of this division is that everything which lives in $\mathbf{Type}_i$ can be extracted to executable code. Hence we tend to call this part of the theory the computational fragment.

To preserve this clear boundary between the logical, impredicative universe **Prop** and the computational universes $\mathbf{Type}_i$ Coq employs a mechanism which we are going to call the *elim-restriction*. The system does not allow us to eliminate an *inductive proposition* into a *type*. In other words, this means that in the process of constructing an element of a *type* Coq only allows us to use the fact that a particular *proposition* holds, but not the details of its proof.[1]

In our development we are going to assume axioms that weaken this boundary somewhat. As a consequence we will lose the capability to extract code, i.e., our metatheory cannot justly be called computational, but we retain consistency. Since we never planned to use the extraction feature this is harmless and we will not concern ourselves with issues of computability in the remainder of this work.

2.2 Logic

Coq's universe **Prop** is the impredicative universe of properties and we present the basic logical connectives here. Universal quantification and implication are primitive in Coq and they come with corresponding abstraction and application mechanisms. They are the dependent and non-dependent function spaces in the universe **Prop**. We write $\forall x : A,\ B$ and respectively $A \to B$ for these.

With the help of *inductive propositions* we are then given the false proposition, negation, conjunction, disjunction, existential quantification and equality as presented here to complete our logical tool set.

```
Inductive ⊥ : Prop := .

Definition not : Prop → Prop := λ A : Prop . A → ⊥.

Inductive and (A B : Prop) : Prop := conj : A → B → A ∧ B.

Inductive or (A B : Prop) : Prop :=
  or_introl : A → A ∨ B | or_intror : B → A ∨ B.

Inductive ex (A : Type) (P : A → Prop) : Prop :=
  ex_intro : forall x : A, P x → ex P.

Inductive eq (A : Type) (x : A) : A → Prop := eq_refl : x = x
```

Each of these behaves exactly as one would expect, i.e., we are given appropriate introduction and elimination rules. As an example we can consider a disjunction $A \vee B$ among our assumptions. The elimination rule for disjunctions now allows us to split the proof in two halves, one with the assumption A and one with the assumption B. Conversely, to show a disjunction, we can use either of the two

[1] The actual mechanism is slightly more involved and a certain number of propositions are exempt from this restriction. Further details can be found in the Coq Reference Manual, located at `http://coq.inria.fr`

constructors, which are the introduction rules. The other definitions are similar. The inductive equality presented here is provably equivalent to Leibniz equality. It is an equivalence relation that allows to substitute equals for equals in the formal development.

It is interesting to observe that each of these constructions has a counterpart that lives in the universe of types. We look at two of these in particular, namely sums which correspond to disjunctions and sigmas corresponding to existential quantification.

```
Inductive sum (A B : Type) : Type :=
  inl : A → A + B | inr : B → A + B.

Inductive sig (A : Type) (P : A → Prop) : Type :=
  exist : forall x : A, P x → sig P.
```

The crucial thing to note is that while $A \vee B$ is a *proposition*, $A + B$ is a *non-propositional type*. And despite the syntactic similarity of their definitions they are not the same. Assume we are constructing an inhabitant of some *type*. Then if we are given an inhabitant of $A + B$ Coq allows us to perform a case distinction on whether this inhabitant was constructed with `inl` or `inr`. If, on the other hand, we only have the assumption $A \vee B$, such a case split is prevented by the *elim-restriction*, as outlined at the end of Section 2.1.

The reader may have noticed that we introduced a more convenient notation for the defined logic operators. In particular `A ∧ B`, `A ∨ B`, `A + B` and `x = y` correspond to `and A B`, `or A B`, `sum A B` and `eq x y` respectively. For negation we will use `¬ A` instead of `A → ⊥`. In addition we will also use a more convenient form for existential quantification and sigma types, which each involve a binding operation. We write **exists** `x : A, P` for `ex A (λ x . P)`, where `P` can have free occurrences of `x`, and similarly for sigma types we have `{x : A | P }` in place of `sig A (λ x . P)`. When the type of the bound variable can be inferred, we may drop it.

2.3 Classical Logic

The type theory CiC underlying Coq is an intuitionistic, or constructive logic. This means that we are not by default given the law of the excluded middle. The law states that every proposition either holds or its negation holds. We should be careful to distinguish this statement about the inherent truth-value of all propositions from notions of provability, i.e., it does explicitly *not* claim that all propositions are provable or not. The inclusion of this axiom changes the intuitionistic logic to a *classical* logic and there are good reasons for as well as against it.

On the one hand there seems to be a connection between the law of the excluded middle and computational control operators like `call/cc` and unguarded `goto` jumps [Gri90]. Since adding these operators to a programming language tends to have subtle and far reaching consequences, some people are sceptical with respect

to the addition of the law on the logic side. The intuitionistic objection is that the (proven) absence of a proof object for $\neg P$ does not "magically" generate a proof object for P.

On the other hand it appears that this principle is an inherent ingredient in the way most people tend to reason in everyday situations, usually without noticing the use of a logical axiom. Similarly most presentations of mathematical proofs tacitly assume it. On these grounds it is fair to argue that classical reasoning is more natural, or intuitive, than intuitionistic reasoning (despite the name).

Due to this disagreement about which kind of logic can be called the "right" logic, the Coq system was designed to accommodate both views. In its basic setup, Coq's underlying theory is intuitionistic, but in such a way that it is compatible with the law of the excluded middle. This means that we can add the law as an axiom without breaking the consistency of the system.

We decided to include the law with the hope to simplify some of the proofs and maybe discover helpful shortcuts. In the context of constructing a proof we can now perform a case split of whether a particular proposition holds or does not hold, generating additional assumptions. Formally the axiom looks as follows.

```
Definition XM : Prop := forall P : Prop, P ∨ ¬P.

Axiom classic : XM.
```

This is not exactly the way the axiom is presented in the library, where a single statement is used, but it is useful to name the type $\forall P : \textbf{Prop},\ P \vee \neg P$, for reasons of presentation.

In the following, many of the proofs are presented from a higher-level, mathematical point of view where uses of this axiom are not explicitly mentioned. The reader should be aware however that when a proof says *"Either X holds or Y"* we are often implicitly using it. If there is any doubt to whether it was used of not, then the Coq proof script should be considered as the authoritative source.

2.4 Choice

We will also require some form of choice principle since a few of the later constructions require a case split already at the point of definition. In the next section we will see how this is achieved, but let us first look at choice principles in general.

Intuitively, they allow us to extract an explicit witness satisfying a particular predicate from the knowledge that some unspecified witness exists. In the light of type theory this means that we can map existentials (which live in **Prop**) to sigmas / strong sums (which live in **Type**).

For our development we select the strongest choice axiom, known as *Hilbert's epsilon statement*, from the library:

```
Axiom ε_statement : forall (A : Type) (P : A→Prop),
  inhabited A → { x : A | (exists x, P x) → P x }.
```

So what does this give us? For any inhabited type A and predicate P on this type we can obtain a particular dependent pair. Its first component is a member of A and its second component is a proof that the predicate holds for this particular member, as long as we can establish that the predicate holds at least for some member.

In practice we are usually working with the left and right projections of this pair, and not directly with the statement itself. The first projection is a higher-order construct similar in flavour to the ε-operator introduced by Hilbert in the context of first order logic [Ack25]:

```
Definition ε (A : Type) (i :inhabited A) (P : A→Prop) : A
  := proj1_sig (ε_statement P i).
```

The second projection is the proof method to establish that we are really obtaining something meaningful with our ε-definition:

```
Definition ε_spec (A : Type) (i :inhabited A) (P : A→Prop) :
  (exists x, P x) → P (ε i P)
  := proj2_sig (ε_statement P i).
```

We will occasionally consider the scenario where we have some particular type T for which the inhabitance has been established, i.e., we have some fixed i : inhabited T. In this case we can form the partial application $\varepsilon\ i$, which we are going to denote by ε_T[2]. When we have established the existence of such a specialised ε_T we will informally speak of having an "ε *at* type T". Note that Coq implicitly infers the first type argument from the inhabitance proof. Let us go through a few scenarios and see how the axiom behaves at different types.

Example 1 (T is empty)**.** *Since T is empty we cannot prove* inhabited T*, and therefore we do not have an ε at T. This makes sense, as an ε at T would allow us to produce elements in our supposedly empty type T.*

Example 2 ($T := \mathbb{N}$)**.** *First of all we can set i :* inhabited $T :=$ inhabits 5 *(or use any other number), so we do have $\varepsilon_{\mathbb{N}}$. We will now look at a number of predicates $P : \mathbb{N} \to$* **Prop**.

Case 1: ($P := \lambda x.\ x = 2 * x$) *It is clear that this predicate is satisfied only by 0, i.e., we can prove $P\ 0$ and $\forall x \in \mathbb{N},\ P\ x \to x = 0$. Looking at the statement of the axiom we know that $\varepsilon_{\mathbb{N}} P : \mathbb{N}$. Since the predicate is satisfiable we also know from* ε_spec *that $P(\varepsilon_{\mathbb{N}} P)$. With the information at hand we can make this even more concrete and infer $\varepsilon_{\mathbb{N}} P = 0$.*

Case 2: ($P := \lambda x.\ x \leq 1$) *Here we have another satisfiable predicate, i.e., we can prove $P\ 0$ and $P\ 1$. As before we have $\varepsilon_{\mathbb{N}} P : \mathbb{N}$ and we also have $P(\varepsilon_{\mathbb{N}} P)$. Interestingly we will be able to prove $\varepsilon_{\mathbb{N}} P = 0 \vee \varepsilon_{\mathbb{N}} P = 1$, however neither $\varepsilon_{\mathbb{N}} P = 0$*

[2] In particular we will be looking at ε_{set}, the choice operator at the type of our formalised sets.

nor $\varepsilon_{\mathbb{N}} P = 1$ *is provable on its own. This highlights the fact that* $\varepsilon_{\mathbb{N}}$ does *pick a satisfying witness, if possible, but it* does not *reveal which one it chose if there were multiple choices.*

Case 3: ($P := \lambda x.\ x = 2 * x \wedge x \neq 0$) *It should be clear that this time there is no possible natural number satisfying the predicate. We still obtain* $\varepsilon_{\mathbb{N}} P : \mathbb{N}$*, but* $P(\varepsilon_{\mathbb{N}} P)$ *will not be provable. In fact its negation,* $\neg P(\varepsilon_{\mathbb{N}} P)$*, is provable.*

2.5 Decidability

It is interesting to consider the consequences of assuming both the *law of the excluded middle* and the choice principle presented above. Together they entail an *informative* variant of excluded middle. This means that we can lift the classical reasoning to the level of types, i.e., we are at this point bypassing the *elim-restriction*. As a further consequence we also obtain a decision procedure on the inhabitance of types that is capable of providing a witness when the type is inhabited (effectively another form of choice principle). Both results are similar in flavour to the basic XM:

Definition IXM : **Type** := **forall** P : **Prop**, P + ¬P.
Definition DIT : **Type** := **forall** T : **Type**, T + (T → ⊥).

In the context of Coq's type theory, this is a serious increase of our reasoning strength and we are now going to prove these claims.

Theorem 1 (Informative Excluded Middle)**.** IXM *is inhabited.*

Proof. Given an arbitrary proposition P we want to establish, that $P + \neg P$. We know that we are classical so we would like to perform a case split on whether P holds or not. We note, however, that we are in the process of showing the inhabitance of a (non-propositional) type, while excluded middle is a proposition and due to the *elim-restriction* we cannot (yet) perform the case split.

In this special case we can get around the restriction by first proving the *proposition* inhabited $(P + \neg P)$. If i is a proof of this proposition then the degenerate ε-construction $\varepsilon\ i\ (\lambda\_.\ \top)$ is a member of the *type* $P + \neg P$.

So we are left with the goal inhabited $(P + \neg P)$. This now is a proposition and Coq allows us to perform the case split on whether P holds or not. In either case it is straightforward to obtain an element in $P + \neg P$, which in turn allows us to conclude the desired goal. □

Theorem 2 (Decidability of the Inhabitance of Types)**.** DIT *is inhabited.*

Proof. Given an arbitrary type T, we wish to establish that $T + (T \to \bot)$. As before we are constructing a member of a type, but now we can exploit Theorem 1 to do a case split on whether the proposition $\mathsf{inhabited}\ T$ holds.

If it does hold, i.e., we have $i : \mathsf{inhabited}\ T$, then $\varepsilon\ i\ (\lambda\_.\ \top) : T$ and therefore $T + (T \to \bot)$.

Otherwise it is easy to establish $T \to \bot$. We simply assume T, which gives us $\mathsf{inhabited}\ T$ and thus a contradiction. The desired result again follows trivially from $T \to \bot$. □

We are now able to translate mathematical definitions of the form

$$A := \begin{cases} B & \text{if } P \\ C & \text{otherwise} \end{cases}$$

into direct Coq code like so

```
Definition A : T := match (ixm P) with
                    | inl H ⇒ B
                    | inr H ⇒ C
                    end.
```

Here ixm is the name of the formal proof of IXM and we have assumed that P : **Prop**. If we have to base the decision on the inhabitance of some T : **Type** (and possibly also need the explicit witness) we would use dit T instead. Effectively we now have an "if-then-else"-construction.

CHAPTER 3

Axiomatisation of Tarski-Grothendieck

We are formally presenting Tarski-Grothendieck set theory, i.e., ZFC with Grothendieck universes[1], as a collection of axioms. We present these axioms here and will elaborate the rôle of the nontrivial ones.

Before we take a look at each of the axioms and basic operations let us briefly reflect on why an axiomatisation is necessary. Cantor originally proposed a notion of set theory where it was possible to take an arbitrary predicate and form the set of all things satisfying it [Can74]. Russell discovered an inherent contradiction in this approach [Rus02] which made the theory unusable as a foundation of mathematics[2]. The problem was that Cantor's theory permitted the formation of sets which were in some sense "too large". The solution to this problem was to construct sets from a number of ground facts, or axioms, which were (mostly) considered noncontroversial and which avoided the introduction of paradoxical sets like the one presented by Russell into the theory. The first such axiomatisation was given by Zermelo in 1908 [Zer08].

We want to formally embed this setup into the type theory of Coq. To achieve this we will pose a type set and a membership relation $\in$. At this point we can of course construct predicates of the form P : set $\to$ **Prop**, which we are going to call *classes*. Then for a given set Y and a given class P, the statement "Y encodes P" corresponds formally to

$$\forall z : \mathsf{set},\ z \in Y \leftrightarrow P\ z.$$

[1] Technically we are working with ZFC + Grothendieck universes since we have a choice axiom in our metatheory prior to the set theory axioms

[2] To be precise, the contradiction appeared in the logic of Frege's Begriffsschrift [Fre79] but it carried over to set theory.

When we define a new Y satisfying a statement of this form it is common to use the set builder notation $Y := \{z \mid P\ z\}$, known as unrestricted comprehension. The important thing for now to remember is that a Y defined in such a way may or may not be a set, which brings us back to the point that some classes are "too large" to be encoded by sets. Our axiomatisation does not include this unrestricted comprehension so we do avoid these problems. Since this way of defining new sets is rather natural though, and since it yields well formed sets in many scenarios, we will later derive a restricted form of comprehension, known as separation. We postpone further discussion of these issues to the corresponding section.

Before we take a look at the actual axioms, we have to take a small technical detour. In Coq, there are a number of instances where several keywords encode the same underlying concepts, e.g., Coq does not distinguish between **Lemma**, **Theorem** or **Corollary**. The different keywords are there solely for structuring the presentation of results. Another instance of this phenomenon are the keywords **Parameter** and **Axiom**. They are technically identical and both introduce a new term at a given type into the general context of assumptions. In the following, we use **Axiom** if the type lives in **Prop** and **Parameter** otherwise.

3.1 Sets and Membership

As mentioned above we will need a type for our sets and a corresponding membership relation, so we start with these. First we augment our type theory with the required type.

Parameter set : **Type**.

The next thing we require is the notion of sets being elements of other sets, given as the binary relation $\in$ on set. The existence of such a relation is also assumed:

Parameter In : set → set → **Prop**.

Here In x X is understood to state that "the set x is an element in (or a member of) the set X".

For convenience we are going to use the relation in its more common infix notation, i.e., we are going to write $x \in X$ instead of In x X and we will use $x \notin X$ for the negation of the statement. We will use the word "containment" with respect to this relation, i.e., when we say "X contains x" we formally mean $x \in X$.

We will often work with properties of sets, which in Coq are given as predicates of the form p : set → **Prop**. These predicates are technically well defined on the complete type set, but we often only want to look at a particular set or domain X and it is sufficient to establish that the predicates hold on all elements of that domain (and we do not care what happens elsewhere). So instead of the proposition

$$\forall x : \mathsf{set},\ p\ x$$

we also use the following, where we add the assumption of being inside the desired domain:

$$\forall x : \mathsf{set},\ x \in X \to p\ x,$$

or, expressed in a more convenient notation we will adopt in the following:

$$\forall x \in X,\ p\ x.$$

Now the truth of this proposition only depends on how the predicate p acts on the domain X. Accordingly we observe that for any $x \notin X$ the body of the statement, namely $x \in X \to p\ x$, is vacuously true.

3.2 Inclusion

It is now easy to define another very common concept from set theory, namely the binary relation *subset* or set inclusion. As with membership we prefer the infix form $X \subseteq Y$ over the prefix notation Subq X Y. It expresses that "the set X is a subset of (or included in) the set Y", which can easily be stated in terms of the membership relation:

$$X \subseteq Y \coloneqq \forall x \in X,\ x \in Y.$$

Note that the statement "X includes Y", i.e., formally $Y \subseteq X$, is different from the statement "X contains Y". In Coq we have the following definition:

```
Definition Subq : set → set → Prop :=
  λ X Y . forall x:set, x ∈ X → x ∈ Y.
```

The following two facts should be clear from the definition.

Lemma 1. $\subseteq$ *is reflexive, formally*

$$\forall A : \mathsf{set},\ A \subseteq A.$$

Proof. Immediate from the definition. □

Lemma 2. $\subseteq$ *is transitive, formally*

$$\forall A\ B\ C : \mathsf{set},\ A \subseteq B \to B \subseteq C \to A \subseteq C.$$

Proof. Immediate from the definition. □

From Lemma 1 and Lemma 2 it follows that $\subseteq$ is a *preorder* on set. In fact we would like to have a *partial order* so we would additionally require *antisymmetry*:

$$X \subseteq Y \to Y \subseteq X \to X = Y. \tag{3.1}$$

This, however, is a statement about the *equality* of sets, a notion that we should first make precise before continuing.

3.3 Extensionality

With any kind of theory it is useful to know when exactly two things are equal. Coq provides us with an inbuilt notion of equality, =, that has all the properties one would expect from such a relation: it is an equivalence relation and we can use it to substitute equals for equals.

With respect to our sets we want something more, namely that *two sets are equal exactly when they contain the same elements*. In other words, our sets should be *extensional*. With the following axiom we augment Coq's equality on the type set with the capability to reason about elements.

Since we defined $\subseteq$ in terms of $\in$ we simply axiomatise the antisymmetry of $\subseteq$ (3.1), effectively turning $\subseteq$ into a *partial order* on the type set. The inverse implication $X = Y \to (\forall x,\ x \in X \leftrightarrow x \in Y)$ follows trivially from the substitution property of =.

Axiom 1 (Extensionality)**.** *Two sets X and Y are equal if they contain the same elements, formally*

$$(\forall x,\ x \in X \leftrightarrow x \in Y) \to X = Y.$$

In the formalisation we have

```
Axiom set_ext : forall X Y : set, X ⊆ Y → Y ⊆ X → X = Y.
```

It is interesting to compare the approach chosen here with the work of Barras [Bar10]. There we find an abstract equivalence relation instead of Coq's inductive equality. On the one hand this allows a greater number of possible models while on the other it leads to a different form of the axiom of replacement (which we will see later). On its own this alternate form of replacement is somewhat more complicated than our version, but it allows a very easy definition for the separation operation. For us, this is where a majority of the work has to be done. The price for working with an abstract equivalence is that compatibility of all the set constructions has to be established explicitly. With Coq's inbuilt equality this follows immediately from the definition of =.

Additionally, a set theory axiomatisation with an abstract equivalence can in some cases actually be implemented or modelled in Coq, (see Aczel's *sets-as-trees* interpretation [Acz78] and [Wer97] for such constructions). Since the addition of Grothendieck universes later will in our case prevent such an endeavour we decided to simply use =.

3.4 ∈-Induction & Regularity

Sets are formed iteratively, so we can prove properties about all sets by induction on membership. Suppose P is a property of sets. If we can prove P holds for X from the inductive hypothesis that P holds for all members of X, then P must hold for all sets.

Axiom 2 (∈-induction)**.** *The membership relation on sets satisfies the induction principle*

$$\forall P : \mathsf{set} \to \mathbf{Prop},\ (\forall X : \mathsf{set},\ (\forall x \in X,\ P\ x) \to P\ X) \to \forall X : \mathsf{set},\ P\ X.$$

In the formal development we have the following

```
Axiom In_ind : forall P : set → Prop,
  (forall X : set, (forall x, x ∈ X → P x) → P X) →
  forall X : set, P X.
```

The quantification over the predicate, or class, P places this statement firmly in the domain of higher-order logic. In a first-order presentation this would be an axiom *schema*, i.e., one axiom for each predicate P. In a first-order setting it is more common, though, to use the logically equivalent axiom of regularity, namely

$$\forall A : \mathsf{set}, (\exists B : \mathsf{set},\ B \in A) \to \exists B : \mathsf{set},\ B \in A \wedge (\neg\exists C : \mathsf{set},\ C \in A \wedge C \in B),$$

which was presented by Zermelo as the *axiom of foundation* [Zer30]. It states that every non-empty set A contains an element B disjoint from A.

In our higher-order metatheory the induction principle shown above is the more practical choice. Independently of the concrete form of the axiom we obtain the consequence that no set is a member of itself and more generally the fact that the membership relation (∈) is *well-founded*, explaining the original name.

As it turns out, this axiom is not required for most of the material presented here, and neither was it part of Zermelo's original theory [Zer08]. Only when we get to the construction of ordinals in the very end do we make use of the fact that *two-cycles* (two sets M and N containing each other) are prevented by this axiom.

3.5 The Empty Set

So far we have seen set membership, set inclusion, an induction principle and a notion of equality on sets, but up to this point we do not even know if our type set is inhabited, i.e., we have not yet claimed the existence of any actual sets. We will now rectify this shortcoming by posing the existence of one fundamental set.

Axiom 3 (The empty set)**.** *There exists a set which does not contain any elements. We call this set the empty set and denote it by* $\emptyset$*. Formally we have*

$$\forall x : \mathsf{set},\ x \notin \emptyset.$$

On the Coq level this is reflected with two statements. The first introduces a new term of type set *and the second clearly pins down the behaviour of the new term.*

```
Parameter Empty : set.
Axiom EmptyE : forall x : set, x ∉ Empty.
```

Now that we know of the existence of one set, all the previous definitions and axioms start to become useful. In the following we will pose a number of operations that build new sets from existing sets. This allows us to populate our type set from the ground up.

There is one aspect of the way the axiom above is presented that should give the careful reader pause. We first claim that some set with a given property exists and then immediately proceeded to talk about *the* set. Why is this ok? Especially since the informal, natural language usage of the notion of set allows us to express a set with no elements in many different ways? Take the set of all €-banknotes with a value strictly greater than 5 and strictly less than 10. Another example is the set of all unicorns.

Traditionally, axioms like this were stated in first-order logic, and we could have taken the same approach (in fact, even the axiom above starts with an existential quantification). It is, however, much easier to talk about and read the definitions and axioms when we assign a clear symbolic representation to the new construct. The justification of this relies on two principles we have already introduced. The type-theoretic choice principle allows us to extract an actual witness from an existential quantification, while the axiom of extensionality then ensures the uniqueness of this witness. Why is that the case? Well, if we look at the actual definition, we observe that it has the form $\forall x,\ x \in Z \leftrightarrow P\ x$ for a particular P that characterises Z. For the empty set, P is simply the constant-$\bot$ predicate so we only required the forward implication, the other holds vacuously. The interesting part is that this characteristic property precisely describes the elements which the newly introduced set Z does contain. Hence all possible Z satisfying this definition contain exactly the same elements, which by extensionality implies that they are in fact the same unique set. This procedure is sometimes referred to as Skolemisation, and the new parameters are called Skolem terms or Skolem constants.

Throughout the remainder of this work we will present most axioms and derived constructs in skolemised form. That is we first pose the existence of a new set-former (n-ary, type-theoretic function on type set) as a **Parameter** and then state a number of **Axiom**s, which together reflect its characteristic, or defining property. Since it is easier to work with in the formal development, we prefer to decompose equivalences into implications.

When working with sets it is often useful to know whether a set is inhabited or not, in much the same way we discussed with respect to types before. The easiest way to do this is to exploit the membership relation. We define the following predicate to express this.

Definition $\mathsf{inh_{set}} := \lambda$ S: set . **exists** w, w ∈ S.

As an example, consider $\neg\, \mathrm{inh}_{\mathrm{set}}\, \emptyset$. This holds since assuming $\mathrm{inh}_{\mathrm{set}}\, \emptyset$ immediately contradicts the axiom of the empty set. This is no surprise as the claim effectively restates the axiom with ($\forall\neg$) replaced by the equivalent ($\neg\exists$). The example simply serves to illustrate our new definition. We also have the following classical result:

$$\forall A : \mathsf{set},\ A = \emptyset \vee \mathrm{inh}_{\mathrm{set}}\ A. \tag{3.2}$$

Many of the results in the remainder depend on whether some of the sets involved are empty or not and, in the latter case, usually on witness of the inhabitance. The property above tells us that we can safely perform case splits of this form to establish the desired results. It is the most common form of classical reasoning we will see in the context of sets. We have, of course, used Coq's classical reasoning facilities to establish (3.2) in the first place.

3.6 Unordered Pairs

We are going to need more than just one set and the intuitive next step is to put two things together. So assume we are given two sets y and z, then we can construct a new set which we call an *unordered pair*.

Axiom 4 (Pairing)**.** *For all sets y and z there exists a set containing exactly y and z as elements. We call this set the* unordered pair *of y and z and denote it by* $\{y, z\}$*. Formally we have*

$$\forall x : \mathit{set},\ x \in \{y, z\} \leftrightarrow x = y \vee x = z,$$

which is coded in Coq as

```
Parameter UPair : set → set → set.
Axiom UPairI1 : forall y z : set, y ∈ (UPair y z).
Axiom UPairI2 : forall y z : set, z ∈ (UPair y z).
Axiom UPairE : forall x y z : set, x ∈ (UPair y z) → x = y ∨ x = z.
```

Note that the axioms on the Coq level can be understood, as the terminology suggests, as introduction and elimination rules with respect to membership in the newly defined set $\{y, z\}$ and we often use the axioms in this fashion. In this case we have also simplified the introduction rules with respect to the substitution property of =, i.e., we used that $x = y \to x \in \{y, z\}$ is equivalent to plain $y \in \{y, z\}$.

Now that we are able to produce sets which actually do contain elements it is a good time to verify that our *unordered pairs* are, as the name suggests, actually unordered.

Theorem 3. *The axiomatic pairing of sets a and b is agnostic with respect to their ordering, formally*

$$\{a, b\} = \{b, a\}.$$

Proof. Using set extensionality it is sufficient to show that $\{a, b\} \subseteq \{b, a\}$, as the other direction is symmetric. Taking an element $x \in \{a, b\}$ and employing UPairE we know that either $x = a$ or $x = b$. In the first case, we have $x \in \{b, a\}$ due to UPairI2, while in the second the same result is obtained with UPairI1, thus completing the proof. □

3.7 Unions of Sets

So far we can produce sets which have exactly zero, one[3] or two elements, regardless of how we twist the machinery already defined. This is not good enough and we would like to combine or merge smaller sets to obtain bigger sets of arbitrary size. This is achieved by forming the union of sets. The reader will likely be familiar with the binary union operator. Theoretically we could use it to express the union of any *finite* number of sets and with the help of the other axioms we could even deal with some *infinite* unions but this is unnecessarily complicated. Instead of providing a union over two sets we will axiomatise an operator that performs a union over an arbitrary *collection* of sets. To some degree this can be seen as a more uniform or generic way to deal with this kind of construction.

A *collection* is, just like everything else, a set whose elements are sets themselves. The terminology is meant to highlight that the union operation mainly affects the elements of the argument and more importantly their respective elements. With this in mind we pose the following.

Axiom 5 (Union)**.** *Given a collection of sets X, there exists a set whose elements are exactly those which are a member of at least one of the sets in the collection X. We call this set the* union *over X and denote it by* $\bigcup X$*. Formally we have*

$$\forall x : \mathsf{set},\ x \in \bigcup X \leftrightarrow \exists Y \in X,\ x \in Y,$$

which is coded in Coq as

```
Parameter Union : set → set.
Axiom UnionI : forall X x Y : set, x ∈ Y → Y ∈ X → x ∈ (Union X).
Axiom UnionE : forall X x : set, x ∈ (Union X) →
  exists Y : set, x ∈ Y ∧ Y ∈ X.
```

It should be clear that the binary union falls out as a special case but we will defer the formal construction until we have seen all the axioms.

[3] An unordered pair of the form $\{a, a\}$, more on that in Section 4.1.

3.8 Powerset

We are able to express that some set is a subset of another and all the sets we have seen so far have at least two such subsets: $\emptyset$ and the set itself; $\emptyset$ is of course an exception which is itself its sole subset. It would be very useful if we could form the set of all subsets of a set. This is known as forming (or taking) the powerset of a set and we postulate the existence of this operation.

Axiom 6 (Powerset)**.** *Given a set X, there exists a set which contains as its elements exactly those sets which are the subsets of X. We call this set the powerset of X and denote it by* $\mathscr{P}(X)$*. Formally we have*

$$\forall Y : \mathsf{set},\ Y \in \mathscr{P}(X) \leftrightarrow Y \subseteq X,$$

which is coded in Coq as

```
Parameter Power : set → set.
Axiom PowerI : forall X Y : set, Y ⊆ X → Y ∈ (Power X).
Axiom PowerE : forall X Y : set, Y ∈ (Power X) → Y ⊆ X.
```

This axiom is special in that it produces "large" sets at a much higher rate than what we have seen so far. If we take a set S with N elements, than $\mathscr{P}(S)$ will have 2^N elements, i.e., the sets grow exponentially. Later we are going to provide sets with certain closure properties and to comprehend the "size" of these it helps to keep this growth rate in mind.

3.9 Replacement

We are working in the context of type theory where higher-order functions are not uncommon. One such function is map which lifts functions on particular objects to lists of such objects, i.e., given a function F that can transform objects of one type to other objects of a possibly different type we can substitute every element in a list by its image under F. Can we do something similar with sets?

We consider mapping a function F over a set X, i.e., we are going to apply F to every element of X. Since the elements of X have type set we know that we have to restrict the argument type of F to set. Also, if the collection of things we obtain from F should form a set, the result type should be set as well. In conclusion we are looking for an $F : \mathsf{set} \to \mathsf{set}$, i.e., a unary set former. We now pose that what we get back after this mapping operation is still a set.

Axiom 7 (Replacement)**.** *Given a unary set former F and a set X, there exists a set which contains exactly those elements obtained by applying F to each element in X. We denote this construction with* $\{F\ x \mid x \in X\}$*. Formally we have*

$$\forall y : \mathsf{set},\ y \in \{F\ x \mid x \in X\} \leftrightarrow \exists x \in X,\ y = F\ x,$$

which is coded in Coq as

```
Parameter Repl : (set → set) → set → set.
Axiom ReplI : forall X : set, forall F : set → set, forall x : set,
  x ∈ X → (F x) ∈ (Repl F X).
Axiom ReplE : forall X : set, forall F : set → set, forall y : set,
  y ∈ (Repl F X) → exists x : set, x ∈ X ∧ y = F x.
```

This axiom was a later addition by Fraenkel in 1922 [Fra22]. He noticed *an up to that point undiscovered gap in Zermelo's foundational set theory that could be closed with the addition of a new axiom*[4]. In short, certain large but well-behaved sets could not be formed in Zermelo's axiomatisation. In place of the higher-order function argument in our type-theoretic approach the original text employs the rather vague notion of *replacing an element of a set with another thing from the basic domain*[5]. One should also mention that the same issue was discovered and repaired, also in 1922, by Skolem [Sko22] who later established that his solution was equivalent to Fraenkel's (pp. 7 - 9, [Sko29]).

It is interesting to compare our replacement axiom with the one found in [Bar10]. In our case we perform the substitution with a (total) function of type set → set while Barras takes a functional relation of type set → set → **Prop**. The important thing here is that his relations are not required to be total, thus admitting a trivial definition of separation. On the downside, Barras has to deal with the fact that the provided relation may not be a morphism with respect to his abstract equality. This leads to extra side conditions in his replacement axiom.

As it turns out this new axiom, in combination with the axiom of the empty set, implies and hence supersedes Zermelo's axiom of separation. Based on this we do not include the latter in our axiomatisation and instead present the separation over sets as a formally derived construction in Section 4.4.[6]

3.10 Grothendieck Universes

We have finally arrived at the point where we are departing considerably from ordinary ZFC. We are postulating the existence of a sequence of sets with certain closure properties. Even the smallest (nontrivial) of these will be considerably bigger than anything we have seen so far. Due to this and the rôle they usually play in formal developments we are going to call them *universes*.

The definition of these universes relies on the notion of a *transitive* set, which we need to make precise. We have seen a transitive relation before, namely ⊆, but

[4]Original: "Diese bisher nicht bemerkte Lücke in der Zermeloschen Begründung ist durch Hinzufügen eines neuen Axioms [...] auszufüllen" (p. 231, [Fra22])

[5]Original: "... und wird jedes Element von M durch ein 'Ding des Bereichs $\mathfrak{B}$' [...] ersetzt, ..." (p. 231, [Fra22])

[6]There seems to be no canonical source for this result and the evident redundancy of having both replacement and separation as axioms is considered folklore, at least according to private communication with Prof. Martin Hyland, DPMMS, Cambridge.

despite the same name and a similar logical structure of the definition, a transitive set should not be confused with a transitive relation. We say that a set U is *transitive*, when the following property holds:

$$\forall x\ X : \mathsf{set},\ x \in X \to X \in U \to x \in U.$$

Now a *Grothendieck universe* is a *transitive set* which is closed under the set operations of *unordered pairing*, *power set*, *union* and *replacement*.[7] We will now postulate the existence of a sequence of such universes.

Axiom 8 (Grothendieck Universes)**.** *For every set X there exists a least transitive set containing X which is closed under the basic set operations, namely pairing, power set, union, and replacement. We denote these universes by* GU_X*. In Coq we encode this as*

```
Parameter GU : set → set.
Axiom GUIn : forall N : set, N ∈ (GU N).
Axiom GUTrans : forall N X Y : set,
  X ∈ (GU N) → Y ∈ X → Y ∈ (GU N).
```

The following Coq statements establish the closure properties of GU*.*

```
Axiom GUUPair : forall N X Y : set,
  X ∈ (GU N) → Y ∈ (GU N) → (UPair X Y) ∈ (GU N).
Axiom GUPower : forall N X : set,
  X ∈ (GU N) → (Power X) ∈ (GU N).
Axiom GUUnion : forall N X : set,
  X ∈ (GU N) → (Union X) ∈ (GU N).
Axiom GURepl : forall N X : set, forall F : set → set, X ∈ (GU N) →
  (forall x : set, x ∈ X → (F x) ∈ (GU N)) → (Repl F X) ∈ (GU N).
```

Clearly, GU_X *is a Grothendieck universe by construction. To be able to talk about* the *Grothendieck universe containing X we are also going to enforce minimality with respect to any other Grothendieck universe U containing X.*

```
Axiom GUMin : forall N U : set, N ∈ U
  → (forall X Y : set, X ∈ U → Y ∈ X → Y ∈ U)
  → (forall X Y : set, X ∈ U → Y ∈ U → (UPair X Y) ∈ U)
  → (forall X : set, X ∈ U → (Power X) ∈ U)
  → (forall X : set, X ∈ U → (Union X) ∈ U)
  → (forall X : set, forall F : set → set, X ∈ U →
      (forall x : set, x ∈ X → (F x) ∈ U) → (Repl F X) ∈ U)
  → (GU N) ⊆ U.
```

[7] The definition can equivalently be given in terms of closure under the *union over a family of indexed sets*, in place of union and replacement. In our formalisation this alternative set operation is a derived construction we will see in Section 4.3.

It is interesting to take a brief detour and look at the history of this axiom and how we arrived at the present formulation of Tarski-Grothendieck set theory. Tarski observed that ZFC cannot prove the existence of strongly inaccessible cardinals (§2, [Tar38]). He assumed the existence to be undecidable and it is now known to be independent of ZFC. He rectified this shortcoming by adding his *Axiom A*. Apart from being able to construct inaccessible cardinals, he observes that the new axiom allows to derive several of the original axioms, most notably the axiom of choice.

Motivated by the simplification of his work in algebraic geometry and category theory Grothendieck defined the notion of universes we have seen above [Gab62, Kru65, GV72]. It turned out that these Grothendieck universes are related to the existence of *inaccessible cardinals* [Wil69], to be precise, "the set of ordinals of a Grothendieck universe is an inaccessible cardinal" (p. 22, [Bar12])[8]. In fact we can use the formulation with Grothendieck universes to obtain a set theory that is equivalent to Tarski' ZFC+A, although without the need to have an axiomatised notion of cardinal numbers. This is the approach that Barras has choseen [Bar10, Bar12] and that we follow as well. For comparison, the variant with Tarski's Axiom A is the formalism used in the Mizar project [Try90].

One may ask, why we include such a powerful construction into our formalisation, apart from the fact that Barras is following the same course of action. The crucial aspect is the correspondence to inaccessible cardinals. In his work, Werner demonstrates that the minimally required number of inaccessible cardinals and the number of type universes interleave when modelling ZFC in CiC and vice versa [Wer97]. With the definitions above, primarily with GUIn, we have introduced an infinite number of Grothendieck universes into our theory which should be more than sufficient to interpret the full, infinite type hierarchy of CiC.

3.11 Infinity

Normally, ZFC also comes with an axiom of infinity, i.e., there exists a set with infinitely many elements. In our case, this is a corollary. We can explicitly name one infinite set: $\mathsf{GU}_\emptyset$.

Corollary 1 (Infinity)**.** *There exists an infinite set, namely* $\mathsf{GU}_\emptyset$.

Proof. We will show that our candidate is infinite by exhibiting a function on it which is injective but not surjective. We have already seen the function on sets which we will use here: $\mathscr{P}(X)$, the powerset construction.

Let us first check that we really have a function *on* $\mathsf{GU}_\emptyset$. Take any $A \in \mathsf{GU}_\emptyset$, then $\mathscr{P}(A) \in \mathsf{GU}_\emptyset$ by GUPower. Next we have to establish that $\mathscr{P}(X)$ is not surjective, i.e., we have to provide an element in $\mathsf{GU}_\emptyset$ that is never produced by the powerset

[8]The correspondence given here relies on the presence of the axiom of choice, which is satisfied in our setting.

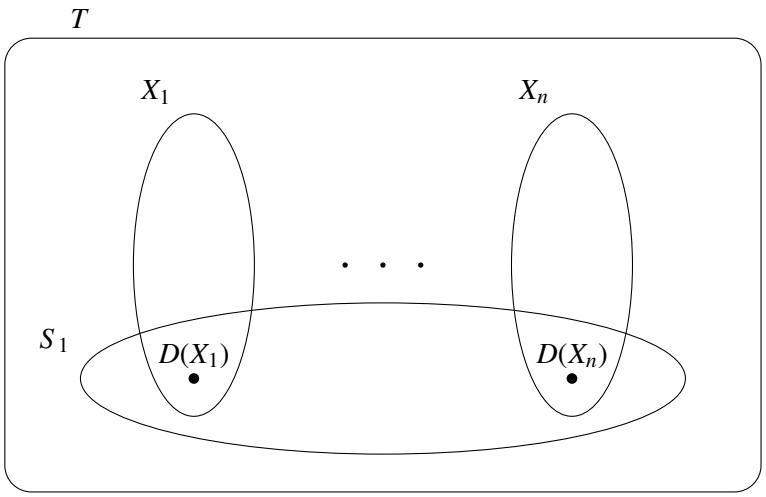

Figure 3.1: *Illustration of Zermelo's axiom of choice*

construction. Take $\emptyset$ where we have $\emptyset \in \mathsf{GU}_\emptyset$ from GUIn. Now it should be clear that $\emptyset \subseteq A$ for all sets A and hence, by PowerI, $\emptyset \in \mathscr{P}(A)$. This tells us that we must have $\mathscr{P}(A) \neq \emptyset$ for all A, since otherwise we would obtain the contradiction $\emptyset \in \emptyset$. So we have established that the powerset construction is not surjective on $\mathsf{GU}_\emptyset$, but what about injectivity?

Assume $\mathscr{P}(A) = \mathscr{P}(B)$, then we have to show $A = B$, which can be reduced to $A \subseteq B \wedge B \subseteq A$. We are going to show the left conjunct, the other half is symmetric. We clearly have $A \subseteq A$, and therefore we obtain $A \in \mathscr{P}(A)$ using PowerI. Our assumption now tells us that we also have $A \in \mathscr{P}(B)$, from which we can infer the desired result with PowerE. This establishes the injectivity of $\mathscr{P}(X)$ and completes the proof. □

3.12 Choice

We remarked before that we are axiomatising ZFC together with Grothendieck universes but so far we are missing a set-theoretic choice principle. In its original form this axiom was stated as follows:

For every collection T of non-empty, disjoint sets, there exists at least one subset of $S_1 \subseteq \bigcup T$ such that each set in T has exactly one element in common with S_1. This can alternatively be seen as the ability to choose one particular element from a number of sets and combine these choices into a single set.[9]

[9]Original: "Ist T eine Menge, deren sämtliche Elemente von 0 verschiedene Mengen und untereinander elementarfremd sind, so enthält ihre Vereinigung $\Im T$ mindestens eine Untermenge S_1, welche mit jedem Element von T ein und nur ein Element gemein hat. [...] Man kann das Axiom auch so ausdrücken[,] daß man sagt, es sei immer möglich, aus jedem Element $M, N, R, \cdots$ von T ein einzelnes Element $m, n, r, \cdots$ *auszuwählen* und alle diese Elemente zu einer Menge S_1 zu

We do already posses the choice principle introduced in Section 2.4 and it is strong enough to give us a choice principle or "axiom" on the level of set theory. It is in general easier, though, to work with the type-theoretic choice principle in the formalisation so the material presented in this section is neither part of the axiomatisation itself nor is it actually formalised. Regardless of these aspects it is still instructive to see how the two notions of choice relate.

Figure 3.1 illustrates Zermelo's form of the axiom. We have taken the liberty to mark the chosen "representative" for each X_k as the result of a metafunction D that is capable of selecting default elements for arbitrary, non-empty sets. The existence of such a D is what lies at the heart of the axiom and, as we will see shortly, arises from our type-theoretic choice principle.

We have introduced the empty set, $\emptyset$, above which guarantees that our type set is inhabited, i.e., we have $\emptyset : \mathsf{set} \vdash \mathsf{inhabits}\ \emptyset : \mathsf{inhabited\ set}$. With this we can define an ε-operator at the type set, namely $\varepsilon_{\mathsf{set}} \coloneqq \varepsilon\ (\mathsf{inhabits}\ \emptyset)$.

This choice operator at type set now enables us to define the metafunction $D : \mathsf{set} \to \mathsf{set}$ as follows:

$$D(X) \coloneqq \varepsilon_{\mathsf{set}}(\lambda x : \mathsf{set}.\ x \in X). \tag{3.3}$$

Additionally, whenever we have $\mathrm{inh}_{\mathsf{set}}\ X$, $\varepsilon\_\mathsf{spec}$ allows us to conclude $D(X) \in X$. Zermelo's choice set S_1 can then be defined with a replacement operation over T:

$$S_1 \coloneqq \{D(X) \mid X \in T\}.$$

We, of course, have to verify that we actually constructed the set requested by the original axiom. Since the following result is only established on paper, we take the liberty to write $X \cap S_1 = \{x\}$ as a short hand notation for $\forall y,\ y \in X \wedge y \in S_1 \leftrightarrow y = x$.

Theorem 4. *For any collection T of non-empty, disjoint sets, the set $S_1 \coloneqq \{D(X) \mid X \in T\}$, where D is defined in* (3.3)*, satisfies Zermelo's properties of the choice set, namely*

1. $S_1 \subseteq \bigcup T$,
2. $\forall X \in T,\ \exists x,\ X \cap S_1 = \{x\}$.

Proof. We show that $S_1 \coloneqq \{D(X) \mid X \in T\}$ has the required properties to be considered a choice set with respect to Zermelo's original formulation.

1. Assume an arbitrary $z \in S_1$, then ReplE tells us that there is an $X \in T$ satisfying $D(X) = z$. Since X is non-empty we have $D(X) \in X$ and therefore $z \in X$. UnionI now allows us to conclude $z \in \bigcup T$, as required.

vereinigen." (p. 266, [Zer08])

2. Here the existential witness can only be $D(X)$ reducing our goal to $X \cap S_1 = \{D(X)\}$. We show inclusion both ways. First, it is easy to establish that $D(X) \in X$ since X is non-empty and $D(X) \in S_1$ only relies on the given fact that $X \in T$, thus establishing $\{D(X)\} \subseteq X \cap S_1$.

   The other direction is slightly more involved. We assume some $x \in X \cap S_1$, i.e., $x \in X$ and $x \in S_1$ and have to establish $x = D(X)$. Since x is a member of the choice set we know that there has to be a $Y \in T$ satisfying $x = D(Y)$. We now establish $X = Y$ by contradiction. Assume $X \neq Y$ and also note that since Y is non-empty we have $D(Y) \in Y$ and therefore $x \in Y$. This, however, contradicts X and Y being disjoint, hence $X = Y$ and $x = D(X)$. We have established $X \cap S_1 \subseteq \{D(X)\}$ which completes the proof of the equality.

□

CHAPTER 4

Development of the TG set theory

There are a number of derived constructions that show up repeatedly or are in general considered to be an integral part of a set theory and hence deserve their own name and/or notation. Providing these and establishing that their associated properties hold will get us from a basic axiomatisation to a theory that is actually useful. The first few are fairly simple and are basically just short hand notations for useful sets, while the later constructions will become quite involved.

4.1 Singleton and Basic Sets

First of all we will look at singleton sets, i.e., sets containing exactly one element. When we introduced unordered pairs we never considered the scenario where the two components of the pair are equal. Any set theory requires that sets do not contain duplicates, hence informally $\{x, x\}$ and $\{x\}$ should denote the same singleton set containing exactly x. This basically motivates the following definition of singleton sets:

$$\{x\} \coloneqq \{x, x\},$$

or respectively in Coq

Definition Sing : set → set := λ X : set . UPair X X.

The defining property of singleton sets is

$$\forall x : \mathsf{set},\ x \in \{y\} \leftrightarrow x = y.$$

In the formal development we have these two corresponding lemmas:

Lemma SingI : **forall** X, X ∈ (Sing X).
Lemma SingE : **forall** X Y, Y ∈ (Sing X) → Y = X.

They are easily derivable as special instances of the respective rules for unordered pairs. It should be clear that it is equally straightforward to ascertain the closure of GU_N under singleton formation.

Apart from $\emptyset$ there are two other sets that will later play a special rôle in the theory and that we can now define, namely $\{\emptyset\}$ and $\{\emptyset, \{\emptyset\}\}$. The reader may recognise these as the common encoding of the first and second ordinal number (with $\emptyset$ being the zeroth) and we will accordingly refer to them as $\mathbf{1}$ and $\mathbf{2}$, respectively. We postpone a general discussion of ordinals to the point where we construct the natural numbers. We observe that $\emptyset \in \mathbf{1} \in \mathbf{2}$ (also $\emptyset \in \mathbf{2}$) as well as $\emptyset \subseteq \mathbf{1} \subseteq \mathbf{2}$. In fact it is no coincidence but a carefully balanced design decision to have $x \in y \leftrightarrow x \subsetneq y$ for $x, y \in \{\emptyset, \mathbf{1}, \mathbf{2}\}$. The reason for introducing them here is that $\mathbf{2}$ can be viewed as a type for propositions with $\emptyset$ and $\mathbf{1}$ representing the false and true proposition, respectively. We will come back to this idea when we deal with functions and function spaces. First however it is instructive to see how the various axioms and constructions behave with respect to these particular sets.

Let us look at inhabitance first. Here we quickly obtain $\mathrm{inh}_{\mathrm{set}}\,\mathbf{1}$ and $\mathrm{inh}_{\mathrm{set}}\,\mathbf{2}$. To prove these we can use $\emptyset$ as the required witness in both cases.

When we consider set inclusion it is useful to note that the subset of a singleton set is either $\emptyset$ or the singleton set itself, and in particular

$$A \subseteq \mathbf{1} \to A = \emptyset \vee A = \mathbf{1}. \tag{4.1}$$

It is worth pointing out that these results are classical, i.e., we crucially rely on the law of the excluded middle to prove this.

Meanwhile, with respect to $\mathbf{2}$, two somewhat related results are useful as well. Assume we know $S \in \mathbf{2}$ for some S. Then we can use an additional piece of information to establish that $S = \mathbf{1}$ or equivalently $\emptyset \in S$, depending on the circumstances. The two formal lemmas are

$$\forall E\ O : \mathsf{set},\ E \in O \to O \in \mathbf{2} \to E = \emptyset \tag{4.2}$$

and

$$\forall S \in \mathbf{2},\ \mathrm{inh}_{\mathrm{set}}\ S \to S = \mathbf{1}.$$

The somewhat asymmetric shapes of the two statements are due to the way they are usually employed. The first is mostly used in forward reasoning to augment an existing context with additional information, while the second is commonly used in a backwards fashion and attempts to produce the simplest possible subgoals for most scenarios.

What happens when we take the union over one of our predefined sets, namely, what are the sets $\bigcup \emptyset$, $\bigcup \mathbf{1}$, and $\bigcup \mathbf{2}$? As it turns out $\emptyset$, $\emptyset$ and $\mathbf{1}$ respectively but let us consider the cases carefully, one at a time.

Theorem 5. *We have the following properties of the union operation with respect to the defined sets* $\emptyset$, $\mathbf{1}$ *and* $\mathbf{2}$*:*

1. $\bigcup \emptyset = \emptyset$
2. $\bigcup \{X\} = X$
3. $\bigcup \mathbf{1} = \emptyset$
4. $X \in \mathbf{2} \longrightarrow \bigcup X = \emptyset$
5. $\bigcup \mathbf{2} = \mathbf{1}$

Proof. We prove the five results in order.

1. To show that some set is equal to the empty set it is sufficient to assume that we have a member of the set in question and then produce a contradiction. This technique is a one-sided form of set extensionality, the other half is always satisfied since $\emptyset$ is included in every set.

   Now we assume some $x \in \bigcup \emptyset$, but then UnionE tells us that we have, for some Y, $x \in Y$ and moreover $Y \in \emptyset$. The latter is the desired contradiction.

2. We use set extensionality for this. First we show $X \subseteq \bigcup \{X\}$, i.e., for a given $x \in X$ we establish $x \in \bigcup \{X\}$. From SingI we know that $X \in \{X\}$ and from this and $x \in X$ we can derive the desired result with the help of UnionI.

   For the other direction, $\bigcup \{X\} \subseteq X$, we start with the elimination principle for unions and obtain, for some Y and x, $x \in Y$ and $Y \in \{X\}$. Elimination of singletons allows us to conclude $Y = X$ from the latter, and therefore $x \in X$ as required.

3. This is simply a special case of the second property where we have instantiated X with $\emptyset$.

4. This fact is an immediate corollary of the definition of $\mathbf{2}$ and properties one and three.

5. This last one is slightly more involved. When exactly is a set equal to the set $\mathbf{1}$? It has to satisfy two criteria. Firstly, it must contain $\emptyset$ and secondly it should *only* contain $\emptyset$. The first is easy, as $\emptyset \in \mathbf{1}$ and $\mathbf{1} \in \mathbf{2}$, but what about the second? Assume we have some arbitrary $x \in \bigcup \mathbf{2}$, then from UnionE we obtain $x \in Y$ and $Y \in \mathbf{2}$ for some Y. At this point we can exploit result (4.2), to ensure that $x = \emptyset$, as required.

□

With respect to the powerset we can state and prove the following two interesting results.

Theorem 6. *We have the following properties of the powerset operation with respect to the defined sets* $\emptyset$, $\mathbf{1}$ *and* $\mathbf{2}$*:*

1. $\mathscr{P}(\emptyset) = \mathbf{1}$
2. $\mathscr{P}(\mathbf{1}) = \mathbf{2}$

Proof. We prove the results in order.

1. We follow the same scheme we used before to establish that a set is equal to $\mathbf{1}$. We noted before that $\emptyset$ is a subset of every set, and concretely $\emptyset \subseteq \emptyset$, from which $\emptyset \in \mathscr{P}(\emptyset)$ follows with PowerI. Next we assume an arbitrary element $X \in \mathscr{P}(\emptyset)$ or, via PowerE, $X \subseteq \emptyset$. This of course implies $X = \emptyset$, and together the two results tell us that $\mathscr{P}(\emptyset) = \mathbf{1}$.

2. For this proof we use set extensionality. The easier of the two is $\mathbf{2} \subseteq \mathscr{P}(\mathbf{1})$. $\mathbf{2}$ contains the sets $\emptyset$ and $\mathbf{1}$ and for each it is easy to establish that they are contained in $\mathscr{P}(\mathbf{1})$ using PowerI. For the first we again appeal to the fact that $\emptyset$ is a subset of everything, while for the second we can exploit the reflexivity of $\subseteq$.

   The more interesting direction is $\mathscr{P}(\mathbf{1}) \subseteq \mathbf{2}$ since it only holds classically. We assume an $X \in \mathscr{P}(\mathbf{1})$ and hence $X \subseteq \mathbf{1}$. Intuitionistically, this is all we can say, but classically we know that $X = \emptyset \vee X = \mathbf{1}$ (see result (4.1)) and from there the proof is trivial.

□

To complete this section we look at the axiom of replacement, where we have another interesting result. It is useful in several cases.

Theorem 7. *Applying the replacement operation with an arbitrary metafunction* F *on* $\emptyset$ *always yields* $\emptyset$*, formally*

$$\{F\ y \mid y \in \emptyset\} = \emptyset.$$

Proof. If we assume an $x \in \{F\ y \mid y \in \emptyset\}$ then ReplE tells us that we must have a $y \in \emptyset$ with $F\ y = x$, but the existence of such a y is a contradiction, which in turn makes it impossible for the x to exist. Hence the result of the replacement is $\emptyset$. □

$$\textsc{UnionI}\ \frac{a \in A \qquad \textsc{UPairI1}\ \dfrac{}{A \in \{A, B\}}}{a \in A \cup B} \qquad \textsc{UnionI}\ \frac{b \in B \qquad \textsc{UPairI2}\ \dfrac{}{B \in \{A, B\}}}{b \in A \cup B}$$

Figure 4.1: *Derivations of* BinUnionI1 *and* BinUnionI2

4.2 Binary Unions

We indicated before that we can have the binary union of two sets A and B, written $A \cup B$, as a derived construct in the present axiomatisation. We make this concrete here with the following definition:

$$A \cup B \coloneqq \bigcup \{A, B\}.$$

In Coq we have

Definition BinUnion : set → set → set := λ A . λ B . Union (UPair A B).

The defining property of the binary union is

$$\forall x : \mathsf{set},\ x \in A \cup B \leftrightarrow x \in A \vee x \in B,$$

which decomposes into a number of lemmas in the formalisation.

Lemma BinUnionI1 : **forall** A B a: set, a ∈ A → a ∈ BinUnion A B.
Lemma BinUnionI2 : **forall** A B b: set, b ∈ B → b ∈ BinUnion A B.
Lemma BinUnionE : **forall** A B x, x ∈ BinUnion A B → x ∈ A ∨ x ∈ B.

The first two are basically pieced together from the corresponding introduction axioms of the constituent components of the definition. The derivations are outlined in Figure 4.1. For the third lemma we reason similarly but the consequence of UnionE is an existentially quantified expression that we need to take apart formally. We consider an existing $Y \in \{A, B\}$ such that $x \in Y$. Using UPairE on the first component yields $Y = A \vee Y = B$, which can be combined with the second component to produce the desired goal.

The fact that our binary union is assembled from constructs under which our Grothendieck universes are closed suggests that this property might also hold for the new construct. Let us go ahead and prove this.

Lemma 3 (GU closed under Binary Union)**.** *Given any Grothendieck universe of the form* GU_N *and a two sets A and B contained in this universe, then their binary union* $A \cup B$ *is also contained in* GU_N*.*

Proof. We know that GU_N is closed under pairing, hence we have $\{A, B\} \in \mathsf{GU}_N$. From this and the closure property under taking the union we obtain $\bigcup\{A, B\} \in \mathsf{GU}_N$, that is $A \cup B \in \mathsf{GU}_N$ as required. □

$$\textsc{UnionI}\ \frac{\textsc{ReplI}\ \dfrac{x \in X}{F_x \in \{F\ x \mid x \in X\}} \qquad y \in F_x}{y \in \bigcup_{x \in X} F_x}$$

Figure 4.2: *Derivation of* FamUnionI

4.3 Union over a Family of Indexed Sets

We have seen the union over an arbitrary collection of sets. In many situations, however, these collections are not arbitrary but exhibit some underlying structure, i.e. all the elements of the collection are similar but depend on a particular index, which is again a set taken from a so-called *index set*. We are looking at set expressions of the form

$$F_x \cup F_y \cup F_z \cup \ldots,$$

where $x, y, z \ldots$ are elements in some set X. In this special case we call F a *family of sets indexed by* X and formally write the union over this family as

$$\bigcup_{x \in X} F_x = \bigcup \{F_x, F_y, F_z, \ldots\}.$$

The right hand side of this equation already suggests a way to define this useful construct with the help of our existing axiomatic constructions. There is nothing special about the index set but for the family we will use a metafunction, that is an F : set $\to$ set, which takes the index to the corresponding member of the family. In this setup, the set $\{F_x, F_y, F_z, \ldots\}$ can be produced with the replacement operation, mapping F over the index set X. Hence our formal definition is

$$\bigcup_{x \in X} F_x \coloneqq \bigcup \{F\ x \mid x \in X\}.$$

We are going to stick with the subscript notation for the indexing but we should keep in mind that under the hood we are working with function applications. The corresponding definition in the formalisation is

Definition FamUnion : set → (set → set) → set := λ X F . Union (Repl F X).

Mathematically we expect the new construct to satisfy its defining property, namely

$$\forall y : \text{set},\ y \in \bigcup_{x \in X} F_x \leftrightarrow \exists x \in X,\ y \in F_x,$$

which is achieved by proving the following lemmas in Coq:

Lemma FamUnionI : **forall** X F x y, x ∈ X → y ∈ (F x) → y ∈ (FamUnion X F).
Lemma FamUnionE : **forall** X F y, y ∈ (FamUnion X F) → **exists** x, x ∈ X ∧ y ∈ (F x).

For the derivation of FamUnionI we refer to Figure 4.2. The elimination lemma has an assumption which hides two existential quantifications, one for the union operation and one for replacement. The following proof sketch illustrates how we deal with these.

We assume some $y \in \bigcup_{x \in X} F_x$ and hence $y \in Y$ for some $Y \in \{F\ x \mid x \in X\}$. This in turn tells us that we have $Y = F_x$ for some $x \in X$. From this it follows that there is some $x \in X$ such that $y \in F_x$, justifying the conclusion of the lemma.

As with the binary union we have only used components for which our universes are closed, again suggesting a corresponding closure lemma.

Lemma 4 (GU closed under Union over Families of Indexed Sets)**.** *Given any Grothendieck universe of the form* GU_N*, a set* X *contained in this universe and an indexed family* F *satisfying* $\forall x \in X,\ F_x \in \mathsf{GU}_N$*, then the union over the family,* $\bigcup_{x \in X} F_x$ *is also contained in* GU_N*.*

Proof. The proof goes as before using GURepl and GUUnion. The given premises allow us to conclude $\{F\ x \mid x \in X\} \in \mathsf{GU}_N$ from which $\bigcup\{F\ x \mid x \in X\} \in \mathsf{GU}_N$ follows. The latter is the desired result, $\bigcup_{x \in X} F_x \in \mathsf{GU}_N$. □

For this slightly more involved construction we have a number of interesting results which derive from our knowledge about the underlying components.

Theorem 8. *Properties of the union over families of indexed sets.*

1. $\bigcup_{x \in \emptyset} F_x = \emptyset$
2. $(\forall x \in X,\ F_x \in \mathbf{2}) \longrightarrow (\exists x \in X,\ F_x = \mathbf{1}) \longrightarrow \bigcup_{x \in X} F_x = \mathbf{1}$
3. $\mathrm{inh}_{\mathrm{set}}\ X \longrightarrow (\forall x \in X,\ F_x = C) \longrightarrow \bigcup_{x \in X} F_x = C$
4. $(\forall x \in X,\ F_x = \emptyset) \longrightarrow \bigcup_{x \in X} F_x = \emptyset$
5. $(\forall x \in X,\ F_x \in \mathbf{2}) \longrightarrow \bigcup_{x \in X} F_x \in \mathbf{2}$

Proof. We prove the five results in order.

1. $\bigcup_{x \in \emptyset} F_x = \bigcup\{F_x \mid x \in \emptyset\} = \bigcup \emptyset = \emptyset$. The second to last equality follows from Theorem 7, while the last is Theorem 5, Part 1.

2. Here we again have to establish that a particular set equals $\mathbf{1}$ so we employ our tried and tested method for these cases. The second assumption tells us that there is an $x \in X$ such that $F_x = \mathbf{1}$, and therefore $\emptyset \in F_x$. From FamUnionI we conclude $\emptyset \in \bigcup_{x \in X} F_x$.

   Now consider some arbitrary $y \in \bigcup_{x \in X} F_x$, i.e., there is some $x \in X$ such that $y \in F_x$. For this particular x our first assumption tells us that $F_x \in \mathbf{2}$. Now the chain $y \in F_x \in \mathbf{2}$ is an instance of (4.2), which tells us that $y = \emptyset$ and completes the proof.

3. For this result itself we need set extensionality as C is arbitrary. First we assume some arbitrary $y \in \bigcup_{x \in X} F_x$, i.e., we have $y \in F_x$ for some $x \in X$. Our second premise now tells us that $F_x = C$ and hence $y \in C$, giving $\bigcup_{x \in X} F_x \subseteq C$. Conversely we have some $y \in C$ and can use the first premise to get hold of some arbitrary $x \in X$. For this x the second premise tells us that $F_x = C$. Hence we can conclude $y \in F_x$. At this point, all we need to do is use FamUnionI to obtain $y \in \bigcup_{x \in X} F_x$ and hence $C \subseteq \bigcup_{x \in X} F_x$.

4. We simply perform a case split on whether X is empty or inhabited. In the first case we have exactly Part 1 of this theorem, while in the second we have an instance of Part 3.

5. The premise tells us that all F_x are members of $\mathbf{2}$, i.e., they are all either $\emptyset$ or $\mathbf{1}$. Now we split on whether there is at least one $F_x = \mathbf{1}$. If this is the case we can use Part 2 to establish our goal. Otherwise we know that all $F_x = \emptyset$ which gives us an instance Part 4, again proving our claim.

□

The significance of these results comes to light when we view $\emptyset$ as logical falsity and $\mathbf{1}$ as truth, as indicated earlier. In this light, Part 2 and Part 4 reflect existential reasoning. The metafunction F plays the rôle of the predicate and the index set that of the argument domain. An existential quantification is true if there exists a witness, which is expressed by Part 2, and it is false if the predicate fails on all arguments, reflected in Part 4.

Finally, Part 5 will be relevant when we get to the construction of function spaces, where it plays a major role in establishing a desired closure property. This in turn will allow us to interpret the impredicative universe **Prop** correctly in our future model construction. We will continue this investigation once we define functions later.

4.4 Separation

In the introduction and in Section 3.12 we mentioned that the naive approach to set theory, i.e., in the tradition of Cantor [Can74], leads to sets which are "too big". In other words, there are classes P where the unrestricted comprehension $\{x \mid P\ x\}$ does not qualify as a set.

The canonical example is the Russell class $P \coloneqq \lambda z,\ z \notin z$. If we assume unrestricted comprehension we can form the set $R \coloneqq \{x \mid P\ x\} = \{x \mid x \notin x\}$, i.e., the set of all sets that do not contain themselves [Rus02]. The contradiction now arises when we ask whether R does contain itself, leading to the paradoxical

$$R \in R \leftrightarrow R \notin R.$$

Another class that cannot be encode is the *universal class*, i.e., there is no set of all sets. We will justify this statement later, once we have derived the correctness of a restricted form of comprehension.

To motivate the construction of separation let us first consider how we would go about deriving comprehension and where we get stuck. Assume we want to define a set Y in terms of a given defining predicate P with comprehension. That is, Y should be defined such that it exactly contains those elements satisfying P, i.e., mathematically

$$\forall x : \mathsf{set},\ x \in Y \leftrightarrow P\ x.$$

When we abstract over Y we can define a new predicate Q, thus

$$Q_P \coloneqq \lambda Z : \mathsf{set}.\ \forall x : \mathsf{set},\ x \in Z \leftrightarrow P\ x. \tag{4.3}$$

At this point we recall our choice operator ε_{set} from Section 3.12 and suggest the following definition:

$$Y \coloneqq \varepsilon_{\text{set}}\, Q_P,$$

where Q_P only depends on the defining property P and $\varepsilon_{\text{set}}\, Q_P$ is the comprehension construction. Summing up, we have defined the set Y to be one of those sets which satisfy the defining property. There may be none, so up to this point the construction is completely vacuous. What we are really after is $Q_P(Y)$, i.e., the fact that our newly defined set really does satisfy its specification. The means to prove this is ε_spec, which reduces the goal to

$$\exists Z : \mathsf{set},\ Q_P(Z) = \exists Z : \mathsf{set},\ \forall x : \mathsf{set},\ x \in Z \leftrightarrow P\ x.$$

This however will not be provable for all predicates P as the Russell predicate illustrates. We observe, though, that for those cases where it is provable, the defined set is unique due to extensionality.

There is a way to make a fairly similar construction work for all P if we assume the axioms of the empty set and the axiom of replacement. The idea is to strengthen the comprehension predicate Q_P from (4.3) with a bounding set X as follows:

$$Q'_{P,X} \coloneqq \lambda Z : \mathsf{set}.\ \forall x : \mathsf{set},\ x \in Z \leftrightarrow x \in X \wedge P\ x.$$

When we write $\varepsilon_{\text{set}}\, Q'_{P,X}$ in set builder notation we get $\{x \in X \mid P\ x\}$. This is commonly known either as restricted comprehension or as separation on sets and it allows us to generate a subset of X where all elements satisfy a given predicate P. In contrast to the unrestricted form, this is known to be well behaved and often taken as axiomatic (e.g., *Axiom III* in [Zer08]). In our case we are able to define the construction and follow the approach outlined above. This leads to the following definition in the formalisation:

```
Definition Sep : set → (set → Prop) → set := λ X P .
  ε (inhabits Empty) (λ Z .  forall x, x ∈ Z ↔ x ∈ X ∧ P x).
```

We have claimed that this construction "is ok" and we will now make this statement concrete, i.e., we prove that every set build with this set former actually satisfies its specification.

Theorem 9 (Definition of Separation is correct)**.** *For all bounding sets X and for all predicates on sets P, the set* Sep X P*, mathematically* $\{x \in X \mid P\ x\}$*, is exactly the subset of X where all elements satisfy P, formally*

$$\forall x : \mathsf{set},\ x \in \{x \in X \mid P\ x\} \leftrightarrow x \in X \wedge P\ x.$$

Proof. We have used ε_{set} to define $\{x \in X \mid P\ x\}$ and following the argument above it is sufficient to show

$$\exists Z,\ \forall x,\ x \in Z \leftrightarrow x \in X \wedge P\ x.$$

We will explicitly construct Z by mapping a function F over X that will act like the identity on those x which do satisfy the property and replace all others by a particular default element x_{def} for which the predicate P is known to hold. This already highlights the first problem, namely the case where no such x_{def} exists. We deal with this corner case first.

Case 1: $\neg\exists x \in X,\ P\ x$. It is clear that in this case $\emptyset$ is the witness we require, reducing the goal to

$$\forall x,\ x \in \emptyset \leftrightarrow x \in X \wedge P\ x,$$

where one direction is trivial while the other produces a contradiction with our assumption.

Case 2: For the remainder of the proof we can now assume that we have $x_{\text{def}} \in X$ available and also $P\ x_{\text{def}}$. All the ingenuity now goes into the construction of the appropriate function to map over X:

$$F\ x := \varepsilon_{\text{set}}\ (\lambda y \Rightarrow P\ x \wedge x = y \vee \neg P\ x \wedge x_{\text{def}} = y).$$

We have again employed the same trick that we used to define Sep for this auxiliary function, and of course we obtain a similar obligation before we can continue, namely

$$\forall x,\ P\ x \wedge x = F\ x \vee \neg P\ x \wedge x_{\text{def}} = F\ x \tag{4.4}$$

which, by the property of ε, again depends on an existence proof for an arbitrary x:

$$\exists y,\ P\ x \wedge x = y \vee \neg P\ x \wedge x_{\text{def}} = y.$$

At this point we can simply do a case split on whether $P\ x$ holds or not. If it does, x will satisfy the goal, and if it does not we can use x_{def} instead. We will shortly make use of two facts that follow from (4.4), namely

$$\forall x,\ P\ x \rightarrow x = F\ x \tag{4.5}$$

and

$$\forall x,\ \neg P\ x \to x_{\text{def}} = F\ x. \tag{4.6}$$

With this additional structure we can now get back to our main goal and use the set $\{F\ x \,|\, x \in X\}$ as our witness. The goal then becomes

$$\forall x,\ x \in \{F\ x \,|\, x \in X\} \leftrightarrow x \in X \wedge P\ x.$$

The backward direction is simple using ReplI and (4.5), while the forward direction requires yet another case split. Since x is a member of $\{F\ x \,|\, x \in X\}$ there must be an $x' \in X$ such that $x = F\ x'$. Now $P\ x'$ may or may not hold. In the first case we can use (4.5) to obtain $x' = F\ x'$ and hence $x = x'$. This case now follows from our assumptions. If it does not hold, it follows from (4.6) that $x_{\text{def}} = F\ x'$ and hence $x = x_{\text{def}}$, which again reduces the goal to known assumptions. This completes the proof.

□

At this point it is interesting to note that we deviated from our usual approach, i.e., we proved the defining property of the separation on sets directly. In general this tends to be the easier approach for definitions based on ε_{set}. In the following it is still easier though to work with separate introduction and elimination lemmas and we state them here without proof. They follow immediately from the correctness result we have just proved.

```
Lemma SepI : forall X, forall P: set → Prop, forall x,
  x ∈ X → P x → x ∈ (Sep X P).
Lemma SepE1 : forall X P x, x ∈ (Sep X P) → x ∈ X.
Lemma SepE2 : forall X P x, x ∈ (Sep X P) → P x.
Lemma SepE : forall X P x, x ∈ (Sep X P) → x ∈ X ∧ P x.
```

As before we also have a couple of additional facts about this construction which are useful in later developments.

First of all, it should be clear that any separation is a subset of its bounding set. This follows immediately from SepE1:

$$\{x \in X \,|\, P\ x\} \subseteq X, \tag{4.7}$$

which can also be expressed as

$$\{x \in X \,|\, P\ x\} \in \mathscr{P}(X). \tag{4.8}$$

At this point we can also extend our knowledge about the closure of our Grothendieck universes to sets formed by separation.

Lemma 5 (GU closed under Separation)**.** *Given any Grothendieck universe of the form* GU_N *and a set* X *contained in this universe, then any separation over* X *is also contained in* GU_N*.*

Proof. We know that GU_N is closed under taking the powerset of X, i.e., $\mathscr{P}(X) \in \mathsf{GU}_N$. Combining this with (4.8) and the fact that GU_N is a transitive set we obtain the desired result, $\{x \in X \mid P\ x\} \in \mathsf{GU}_N$ □

Next we have a few results about particular bounding sets, namely

$$\{x \in \emptyset \mid P\ x\} = \emptyset \tag{4.9}$$

and the somewhat more useful

$$\{x \in \mathbf{1} \mid P\ x\} = \emptyset \vee \{x \in \mathbf{1} \mid P\ x\} = \mathbf{1},$$

which follows from the fact that subsets of $\mathbf{1}$ are either $\emptyset$ or $\mathbf{1}$ (see (4.1)). Since $\mathbf{1}$ is just a special singleton set the following more general variant also is of some use:

$$(\{x \in \{x\} \mid P\ x\} = \{x\} \wedge P\ x) \vee (\{x \in \{x\} \mid P\ x\} = \emptyset \wedge \neg P\ x).$$

And finally, if we actually know that a particular separation yields the empty set, it is easy to argue that none of the members of the bounding satisfy the property:

$$\{x \in X \mid P\ x\} = \emptyset \rightarrow \forall x \in X, \neg P\ x.$$

Now that we know about separation, it is interesting to think about the universal class again, i.e., a class containing all sets, and whether it can be encoded as a set. We bring this up as it is relevant in the context of set intersection, the construction we will be dealing with next. To see why a universal set cannot exist, consider the following contradiction. Assume a universal set U exists, then the separation $\{x \in U \mid x \notin x\}$ over this universal set U is a set encoding the Russell class. As we argued above, however, there is no such set and therefore U cannot exist.

What does this have to do with intersection? Well, the class intersection of a set M is easily defined as

$$\bigcap^{\mathbf{C}} M \coloneqq \{x \mid \forall A \in M,\ x \in A\},$$

but when we consider the case for $M = \emptyset$ we observe that $\bigcap^{\mathbf{C}} \emptyset$ is the universal class: the defining condition is vacuously true for all sets. This poses a difficulty to encoding $\bigcap^{\mathbf{C}}$ as a set.

We can work around this issue with a separation over a sufficiently large bounding set, allowing correct behaviour of the intersection operation over any non-empty set M. To obtain this consider the following. Any element x that is part of the intersection is a member of all the sets A in the non-empty set M. We can weaken this specification to say that there is a set $A \in M$ which contains x. We recognise the latter as the union over M, which makes sense since we expect $\bigcap M \subseteq \bigcup M$. The idea is taken from Fraenkel's historical introduction in [Ber58] and can also be found in the recent work of Barras [Bar12]. Our actual definition is then

$$\bigcap M \coloneqq \{x \in \bigcup M \mid \forall A \in M,\ x \in A\},$$

which is formally given as

```
Definition Inter : set → set := λ M .
  Sep (Union M) (λ x : set . forall A : set, A ∈ M → x ∈ A).
```

The defining property is of course

$$\mathrm{inh}_{\mathrm{set}}\, M \longrightarrow (\forall x,\ x \in \bigcap M \leftrightarrow \forall A \in M,\ x \in A),$$

which is established by proving the following two results in Coq.

```
Lemma InterI : forall x M, inh_set M → (forall A, A ∈ M → x ∈ A) → x ∈ Inter M.
Lemma InterE : forall x M, x ∈ Inter M → inh_set M ∧ forall A, A ∈ M → x ∈ A.
```

We point out that the given definition leads to the following somewhat counterintuitive result:

$$\bigcap \emptyset = \emptyset.$$

It is an immediate combination of Theorem 5, Part 1 and Equation (4.9). This is technically unproblematic since intersection is now defined on all sets, but we expect correct behaviour only from the nonempty ones. Finally we look at universe closure.

Lemma 6 (GU closed under Intersection)**.** *Given any Grothendieck universe of the form* GU_N *and a set X contained in this universe, then the intersection over X is also contained in* GU_N.

Proof. Follows immediately from Lemma 5 and the closure under the union operation. □

As a last remark we could also consider binary intersection, which can be obtained from this arbitrary intersection along the same lines we employed for binary unions earlier. This, however, has not been formalised.

4.5 Ordered Pairs

In the context of set theory there are two notions of pairing, namely *unordered* pairs and *ordered* pairs.

The unordered version we have seen above is axiomatic in our development. They are explicitly not equipped with a notion of component ordering, i.e.,

$$\{a, b\} = \{b, a\}$$

and concepts like *first* and *second* component do not even make sense. Apart from being a fundamental building block of our theory, they are occasionally quite useful in higher-level developments. They should, however, not be confused with the concept most people tend to have in mind when talking about pairs.

Ordered pairs, on the other hand, are exactly what most mathematicians think of when talking about pairs and with these we are able to precisely identify the *first* and *second* component. Apart from the additional ordering, these pairs also come with projection operations which provide access to the two components of a pair. For the ordered pair of the sets a and b and its projections we write $\langle a, b\rangle$ and

$$\pi_1\langle a, b\rangle = a,$$
$$\pi_2\langle a, b\rangle = b.$$

This notation is used to clearly distinguish the two versions of pairs and to avoid confusing their different properties.

In most mathematics texts, ordered pairs are just assumed to exist without further elaboration, effectively turning them into axiomatic building blocks. Here, however, we are carefully building up the foundations and it is not necessary to assume the existence of ordered pairs. They can be a defined entity instead, given the machinery we already have. There are several ways to accomplish this. Regardless of the encoding we choose, we should be able to prove the *characteristic property of ordered pairs*:

$$\langle a, b\rangle = \langle c, d\rangle \leftrightarrow a = c \wedge b = d. \tag{4.10}$$

We use a construction which was suggested by Kuratowski in 1921 [Kur21]. It has the nice property that it is self-contained. It does not rely on some form of *marker sets* (as in $\{\{a, \mathrm{O}\}, \{b, \mathbf{1}\}\}$) to distinguish the components. Due to this and other favourable properties[1], these so-called Kuratowski Pairs have become the de facto standard construction in many texts on the foundation of mathematics and set theory.[2]

We define ordered pairs as

$$\langle a, b\rangle \coloneqq \{\{a\}, \{a, b\}\},$$

with the following projection operations:

$$\pi_1 p \coloneqq \bigcup\bigcap p$$
$$\pi_2 p \coloneqq \bigcup\{x \in \bigcup p \mid x \in \bigcap p \to \bigcup p = \bigcap p\}$$

The corresponding formal encoding is

```
Definition op_k : set → set → set := λ x y: set . UPair (Sing x) (UPair x y).
Definition π₁ : set → set := λ p . Union (Inter p).
Definition π₂ : set → set := λ p . Union (Sep (Union p)
  (λ x : set . x ∈ Inter p → Union p = Inter p)).
```

[1] Another self contained construction is $\{a, \{a, b\}\}$. This version, however, requires the Axiom of Regularity in order to obtain the characteristic property of pairs (4.10), while Kuratowski Pairs do not.

[2] See, for example, Chapter 2.1 in [HJ99], although they do not call them by this name.

Let us take a minute and motivate these definitions. First of all we remind ourselves that taking the union over a singleton set produces exactly that single element, i.e., speaking informally, this allows us to strip one layer of set braces. This is exactly the purpose of the outer union operation in both definitions. Then, for the definitions to make sense, the sets over which these unions are taken should be singletons sets, more precisely, when P is the ordered pair $\langle a, b\rangle$, we want to obtain the sets $\{a\}$ and $\{b\}$, respectively.

Furthermore, looking at the next building blocks in our definition, let us see what happens when we take the union or the intersection over an ordered pair.

$$\begin{aligned} \bigcup\langle a, b\rangle &= \{a\} \cup \{a, b\} = \{a, b\} \\ \bigcap\langle a, b\rangle &= \{a\} \cap \{a, b\} = \{a\} \end{aligned} \tag{4.11}$$

At this point it should be intuitively clear that the definition of the first projection works as intended, as long as we apply it to actual pairs.

Lemma 7. *For all sets x and y*

$$\pi_1\langle x, y\rangle = x$$

Proof. Trivial using (4.11). □

For the second projection the situation is slightly more complicated. The reader might expect something of the form $\pi_2 P \coloneqq \bigcup\{x \in \bigcup P \mid x \notin \bigcap P\}$ but this breaks down in the case where the two components are equal. Hence we need to express the fact that the component we are looking at is either not in the intersection, in which case we have our hands on the correct one, or, if it happens to be in there but the components are equal, then there effectively is only one component, precisely the one we are looking for. Note that inside the definition we do not have direct access to the two components (we are defining projections for generic set P), but stating that the union and the intersection coincide amounts to the same thing, at least for our definition of pairs.

Lemma 8. *For all sets x and y*

$$\pi_2\langle x, y\rangle = y$$

Proof. We have to consider two cases here, namely whether x and y are equal or not. Let us deal with the inequality first.

Case 1: Take $x \neq y$, then $\bigcup\langle x, y\rangle = \{x, y\} \neq \{x\} = \bigcap\langle x, y\rangle$. It is clear that $(x \in \{x\} \to \{x, y\} = \{x\})$ is false, while $(y \in \{x\} \to \{x, y\} = \{x\})$ is vacuously true. The expression becomes $\pi_2\langle x, y\rangle = \bigcup\{y\} = y$, as required.

Case 2: Now assume $x = y$, i.e., we are dealing with pairs of the form $\langle y, y\rangle$ and note that $\bigcup\langle y, y\rangle = \{y\} = \bigcap\langle y, y\rangle$. In this scenario, the only instance of the property, namely $(y \in \{y\} \to \{y\} = \{y\})$, is trivially true. Hence we again obtain $\pi_2\langle y, y\rangle = \bigcup\{y\} = y$, as required.

□

Now that we have established that our projection operations behave as desired we can use this information to finally obtain a proof of (4.10)

Theorem 10 (Characteristic Property of Ordered Pairs)**.** *For all sets a, b, c and d,*

$$\langle a, b\rangle = \langle c, d\rangle \leftrightarrow a = c \wedge b = d.$$

Proof. Let us look at the two directions separately

Case $\leftarrow$ *:* Trivial using the substitution property of equality.

Case $\rightarrow$ *:* We have $\pi_1\langle a, b\rangle = a$ from Lemma 7 and therefore $\pi_1\langle c, d\rangle = a$ by assumption. Again from Lemma 7 we obtain the desired equality $c = a$ (up to symmetry). The proof of $b = d$ is analogue, using π_2 in place of π_1. □

Formally it is helpful to be able to classify, which of the sets in our universe of discourse are well-formed ordered pairs, and which are not. Among other things this helps us with the clean definition of functions later in Section 4.6.

Here is the corresponding extract from the formal development, where (x,y)k is just notation for op_k x y.

Definition is_pair : set → **Prop** := λ p . **exists** x, **exists** y, p = (x,y)k.

It is now easy to express properties which require their arguments to be pairs. Take eta expansion as an example:

Lemma op_k_eta : **forall** p, is_pair p ↔ p = (π_1 p, π_2 p)k.

We are normally only concerned with the forward direction of this result, but note that this provides us with an alternative classification of which sets are pairs, i.e., all those where forming a pair out of the first and second projection produces the original set.

A concept that is closely related to ordered pairs is the notion of a *Cartesian Product* of two sets, which provides a means to express sets of ordered pairs.[3] Using the axioms of replacement and the union over a family of sets we can define them as

$$A \times B := \bigcup_{a \in A} \{\langle a, b\rangle \mid b \in B\}.$$

Formally we have

Definition CProd : set → set → set :=
λ A B . FamUnion A (λ a . Repl (λ b . (a,b)k) B).

We prove

Lemma CProdI : **forall** a A b B, a ∈ A → b ∈ B → (a,b)k ∈ CProd A B.
Lemma CProdE1 : **forall** p A B, p ∈ CProd A B → π_1 p ∈ A ∧ π_2 p ∈ B.
Lemma CProdE2 : **forall** p A B, p ∈ CProd A B → is_pair p.

[3] For those familiar with the concept of typing: *a Cartesian Product is the type of an ordered pair*

and thus establish the defining property of the cartesian product, namely

$$\forall p : \mathsf{set},\ p \in A \times B \leftrightarrow \pi_1 p \in A \wedge \pi_2 p \in B \wedge \mathsf{is\_pair} p.$$

There are two simple properties that will be helpful in later developments. In both cases the proofs use extensionality of sets, resulting in contradictions of the form $a \in \emptyset$.

Lemma 9. *For all sets B,* $\emptyset \times B = \emptyset$.

Lemma 10. *For all sets A,* $A \times \emptyset = \emptyset$.

We have now constructed everything which is necessary to start talking about a notion of functions, where ordered pairs play a central rôle.

4.6 Functions

In this section we will discuss three interlocking constructions that equip our set theory with typed functions. Firstly there are abstractions or λ-terms which represent the functions themselves, written as $\lambda x \in X.e_x$. The subscript indicates the possibility of free occurrences of x in e. These abstractions will be applied to arguments so the next thing we require is an application mechanism $(f\ x)$ and finally we want to be able to express function spaces, or in other words, function types. In fact we will be dealing with dependent functions and for these one usually speaks of dependent products instead. These are written as $\Pi x \in X.Y_x$. For the special case where x does not occur free in Y we use the more common notation $X \dot{\longrightarrow} Y$.

Formally we want three new set formers at the following types,

$$\begin{aligned} \mathsf{ap} &: \mathsf{set} \to \mathsf{set} \to \mathsf{set}, \\ \mathsf{lam} &: \mathsf{set} \to (\mathsf{set} \to \mathsf{set}) \to \mathsf{set}, \\ \mathsf{Pi} &: \mathsf{set} \to (\mathsf{set} \to \mathsf{set}) \to \mathsf{set}. \end{aligned}$$

The Pi provides dependent function spaces but in several cases a non-dependent version is sufficient and easier to work with. We give the definition here, but the construction is formally not very interesting so we will not discuss it at any depth:

$$\mathsf{Arr} : \mathsf{set} \to \mathsf{set} \to \mathsf{set} := \lambda X\ Y.\ \mathsf{Pi}\ X\ (\lambda\_.\ Y).$$

What is interesting though are the types of the new constants. The simplest one is function application. It takes two sets and returns the set that is the result of applying the first to the second. We observe that it is technically possible to feed arbitrary sets to this set former but we only expect sensible behaviour if the first set actually encodes a function and the second is a suitable argument. We are going

to investigate to some degree what happens beyond these implicit boundaries as it helps us with some of the proofs, but we will get to that later.

For abstractions and products the types are slightly more involved since we have to deal with a variable binding operation. In both cases the first set given to the set formers is the argument domain of the encoded function or product. The second component is in both cases a metalevel function which encodes the body with a free occurrence of a variable. In other words, we use Coq's metalevel binding mechanism to encode the binding operation for abstractions and products.

We call these new constructions functions but to justify this name they should satisfy a number of properties. We state these properties loosely to get an intuition, i.e., we avoid being too pedantic about side conditions for the sake of a simple presentation. We will of course later formally prove that the properties hold in full detail for the actual definitions of our new constructs.

The key defining property we expect from functions is β-reduction which relates abstraction and application:

$$\mathsf{ap}\ (\mathsf{lam}\ X\ e)\ x = e_x.$$

In addition to that we have the dependent products which should behave like the type of abstractions. In the language of sets this is a statement about set containment for suitably related e and Y:

$$\mathsf{lam}\ X\ e \in \mathsf{Pi}\ X\ Y.$$

The last core property of interest is the fact that everything taken from a product can be used as a function with respect to application and return something sensible:

$$f \in \mathsf{Pi}\ X\ Y \to \mathsf{ap}\ f\ x \in Y_x.$$

If our only goal would be a formalisation of set theory we could simply use the well known *standard graph encoding* and finish the section at this point. We have however mentioned a number of times already that we plan to use this set theory for the construction of models for type theory, which poses an additional constraint and forces us to seek an alternative solution. To see where the default solution breaks down and to motivate both the extra constraint and the solution we have to take a small detour.

In the standard graph encoding, a function or abstraction is represented by a set of ordered pairs where the first components belong to the domain of the function and the second components to its co-domain. This is a binary relation which has to be *functional* to be considered a valid encoding. That is, for each element x in the domain there is exactly one pair of the form $\langle x, y\rangle$ in the encoding set, with y being a member of the co-domain. In fact, any functional binary relation (i.e., set of ordered pairs) represents some function and its domain can be reconstructed by

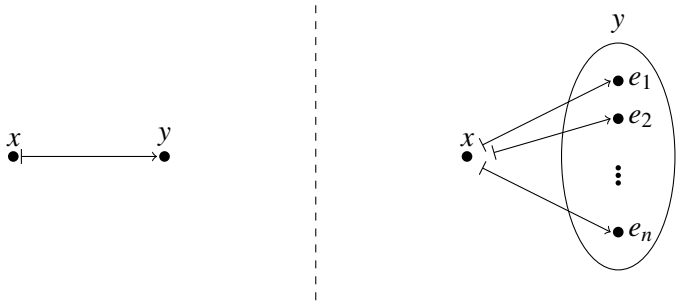

Figure 4.3: *A function mapping the value x to the value y as represented in the* standard graph encoding *(left) and in the* Aczel encoding *(right), respectively.*

just gathering up all first components.[4] The corresponding application operator is easy. We simply have to find the pair in the set where the first component matches the argument of the function application and project out the corresponding second component. This encoding is clean and simple but unfortunately insufficient for the following reason.

In Section 4.1 we mentioned that **2** can be viewed as the type of propositions[5] and in this light it is interesting to look at logical implications. We are going to build on Kolmogorov's idea that function types and logical implications are the same [Kol32, Hey74]. In particular this entails that functions inhabiting function spaces are proofs of the corresponding implications.

We would expect that a logical implication constructed from two propositions is itself a proposition, i.e., we are looking for a closure property of **2** under the construction of function spaces, formally[6]

$$\forall A\ B \in \mathbf{2},\ A \dot{\longrightarrow} B \in \mathbf{2}. \tag{4.12}$$

This, however, is not satisfied when we use the standard encoding. For a counter-example, take the function type $\mathbf{1} \dot{\longrightarrow} \mathbf{1}$. Read as a logical implication we would expect this to be true, i.e., equal to the set **1**. Looking at the (informal) definitions above it is easy to see that the constructed function space will contain exactly one function, namely the one with the single mapping $\emptyset \mapsto \emptyset$. Hence with the standard encoding we would have $\mathbf{1} \dot{\longrightarrow} \mathbf{1} = \{\{\langle \emptyset, \emptyset \rangle\}\} \notin \mathbf{2}$.

The solution is an encoding of functions which was envisioned by Aczel [Acz98], explicitly designed to satisfy (4.12) in the context of modelling impredicative type universes. The idea is to still encode abstractions as sets of ordered pairs but instead of representing the mapping $x \mapsto y$ with a single pair $\langle x, y \rangle$ we

[4] This is relevant since we do not have this property for the encoding we actually choose.

[5] To be precise, we intend to use **2** as the interpretation of ECC's lowest, impredicative type universe **Prop** in our planned model construction.

[6] We use the non-dependent version. The dependent version is mostly the same, but hides the concept behind a less clear expression.

include one pair $\langle x, e_i \rangle$ for each $e_i \in y$. The resulting relation can still be considered as a graph albeit in most cases a non-functional one. In general though, this analogy should be avoided since the semantics of an edge have changed considerably. Figure 4.3 illustrates the difference with a function that has as its only mapping $x \mapsto y$.

Clearly we have to adjust the application mechanism. Given an encoded function, i.e., a set of ordered pairs, and an argument set, we now have to collect all the elements associated with the argument and gather them up into a set. This set is then returned as the result of the application.

With this principle in mind we pose the following definitions:

$$\begin{aligned}
\mathsf{ap}\ f\ x &:= \{\pi_2\ p \mid p \in \{p \in f \mid \pi_1\ p = x \wedge \mathsf{is_pair}\ p\}\} \\
\mathsf{lam}\ X\ F &:= \bigcup_{x \in X} \{\langle x, y \rangle \mid y \in F\ x\} \\
\mathsf{Pi}\ X\ Y &:= \{f \in \mathscr{P}(X \times \bigcup(\bigcup_{x \in X} Y_x)) \mid \forall x \in X,\ \mathsf{ap}\ f\ x \in Y_x\}
\end{aligned}$$

At a first glance it is of course neither obvious that these definitions give us Aczel-encoded functions nor that they satisfy any of the properties we required in the beginning. In the remainder of this section we will work through a number of proofs that establish the necessary properties and further related results and thus show that we actually have a workable encoding of set-theoretic functions in our development. In comparison to Barras [Bar10] we construct these functions directly while he first produces the standard encoding and then proceeds to embed this into the Aczel representation. This primarily affects the definition of products while abstractions and applications are essentially defined in the same way we did.

To get started with the formal results we present a rather curious observation, namely that Cartesian products and constant abstractions coincide in our set theory. We were not able to find a real use case for this but think that it is interesting nonetheless.

Lemma 11. *The Cartesian product of sets A and B coincides with the constantly B abstraction on domain A:*

$$A \times B = \mathsf{lam}\ A\ (\lambda\_.\ B).$$

Proof. Immediate from the definitions. □

We begin our main analysis by showing that abstractions only contain pairs.

Lemma 12. *Given a $p \in \mathsf{lam}\ X\ F$ there exist sets $x \in X$ and $y \in F\ x$ such that $p = \langle x, y \rangle$.*

Proof. Unfolding the definition of abstractions reveals $p \in \bigcup_{x \in X} \{\langle x, y\rangle \mid y \in F\ x\}$. From the elimination principle FamUnionE we obtain an $x \in X$ for which $p \in \{\langle x, y\rangle \mid y \in F\ x\}$. At this point ReplE tells us that there exists a $y \in F\ x$ such that $p = \langle x, y\rangle$, completing the proof. □

A related result concerns the behaviour of application with arbitrary sets.

Lemma 13. *Given an arbitrary set f, then f contains a pair $\langle x, y\rangle$ iff applying f to x yields a set containing y, formally*

$$y \in \mathsf{ap}\ f\ x \leftrightarrow \langle x, y\rangle \in f.$$

Proof. We look at each of the implications in turn. For the forward direction we can use ReplE to obtain $y = \pi_2 p$ for some $p \in \{p \in f \mid \pi_1\ p = x \wedge \mathsf{is\_pair}\ p\}$. This tells us that $p \in f$ and $\pi_1 p = x$ and additionally that p is in fact a pair, which means that we can η-expand it to $\langle \pi_1 p, \pi_2 p\rangle$. We can now substitute to obtain the desired result.

For the other direction we have a pair $\langle x, y\rangle \in f$ and it is obvious that $\pi_1\langle x, y\rangle = x$ as well as $\mathsf{is\_pair}\ \langle x, y\rangle$. We can therefore deduce that $\langle x, y\rangle \in \{p \in f \mid \pi_1\ p = x \wedge \mathsf{is\_pair}\ p\}$. If we now replace everything in the latter with the second projection π_2 we have established that $\pi_2\langle x, y\rangle \in \mathsf{ap}\ f\ x$. The desired result then follows from Lemma 8. □

Now, before we look at any of the subtler results we are going to verify that our constructions exhibit the fundamental property of functions: an abstraction applied to an argument (of its domain) β-reduces.

Theorem 11 (β-reduction of Aczel-encoded functions)**.** *Given a domain set X and a metafunction on sets F, then for every $x \in X$, the following reduction equation holds:*

$$\mathsf{ap}\ (\mathsf{lam}\ X\ F)\ x = F\ x.$$

Proof. We show inclusion both ways. Assume we have a $z \in \mathsf{ap}\ (\mathsf{lam}\ X\ F)\ x$, then ReplE and Theorem 9 allow us to conclude that there is an ordered pair $p \in \mathsf{lam}\ X\ F$ for which $\pi_1 p = x$ and $\pi_2 p = z$. FamUnionE now tells us that p is obtained by replacing some $y \in F\ x$ with $\langle x, y\rangle$. Hence we can conclude $y = z$ and therefore $z \in F\ x$ as required.

For the other direction we take an arbitrary $y \in F\ x$, which we can equally express as $\pi_2\langle x, y\rangle \in F\ x$. To show that $\pi_2\langle x, y\rangle \in \mathsf{ap}\ (\mathsf{lam}\ X\ F)\ x$, it is sufficient to establish that $\mathsf{is\_pair}\ \langle x, y\rangle$ and $\pi_1\langle x, y\rangle = x$, both of which are trivial, as well as $\langle x, y\rangle \in \mathsf{lam}\ X\ F$. As we have seen several times now we can establish the membership in a union over an indexed family, if we can present a witnessing index, where we choose x of course, and establish that our target set, here $\langle x, y\rangle$, is contained in the indexed member, here $\{\langle x, y\rangle \mid y \in F\ x\}$. This follows from our assumption and completes the proof. □

The reader may have noticed that the result we just proved relies on the argument being contained in the declared domain of the abstraction. This immediately brings up the question of what happens if we we know that the argument is not in the domain. As it turns out we obtain the empty set.

Lemma 14. *Applying a given abstraction* $\mathsf{lam}X\ F$ *to an argument* x *outside of its domain* X*, that is* $x \notin X$*, always yields the empty set, formally*

$$x \notin X \longrightarrow \mathsf{ap}\ (\mathsf{lam}\ X\ F)\ x = \emptyset.$$

Proof. We assume that the redex is non-empty, i.e., we have a $w \in \mathsf{ap}\ (\mathsf{lam}\ X\ F)\ x$, and produce a contradiction. From Lemma 13 it follows that $\langle x, w\rangle \in \mathsf{lam}\ X\ F$ and from Lemma 12 we can further conclude that there is a pair $\langle z, y\rangle = \langle x, w\rangle$ such that $z \in X$. From the characteristic property of ordered pairs we deduce that $z = x$ and therefore $x \in X$ which contradicts our initial assumption and completes the proof. □

Another interesting form of redex involves the function encoded by the empty set, namely the constantly $\emptyset$ function, which is agnostic with respect to domains.

Lemma 15. *When we apply the empty set to any set, we always obtain* $\emptyset$*, formally*

$$\mathsf{ap}\ \emptyset\ x = \emptyset.$$

Proof. We have $\mathsf{ap}\ \emptyset\ x = \{\pi_2\ p \mid p \in \{p \in \emptyset \mid \pi_1\ p = x \wedge \mathsf{is\_pair}\ p\}\} = \{\pi_2\ p \mid p \in \emptyset\} = \emptyset$ where we have used the definition of ap, Equation (4.9) and Theorem 7 for the equalities, respectively. □

Now that we have looked at β-reduction and various associated results it is interesting to move to the issue of typing. We begin with the most important property first, namely that products are the types of abstractions. In the context of sets typing is represented by set membership, yielding the following lemma.

Theorem 12. *Given a domain set* X *and two metafunctions* F *and* Y *such that* $\forall x \in X,\ F\ x \in Y_x$*, we have*

$$\mathsf{lam}\ X\ F \in \mathsf{Pi}\ X\ Y.$$

Proof. To establish this result we have to show two things. First the abstraction has to be contained in the bounding set of the separation which defines Pi and secondly we must satisfy the condition of that separation. As it turns out, the second is the easier of the two and this is where we begin.

Our goal then is $\forall x \in X,\ \mathsf{ap}\ (\mathsf{lam}\ X\ F)\ x \in Y_x$. Since $x \in X$ we can β-reduce the redex to $F\ x$ which is contained in Y_x by assumption.

For the bounding set we have to show that $\mathsf{lam}X\ F \in \mathscr{P}(X \times \bigcup(\bigcup_{x\in X} Y_x))$ or equivalently $\mathsf{lam}X\ F \subseteq X \times \bigcup(\bigcup_{x\in X} Y_x)$. From Lemma 12 we know that everything contained in the abstraction is a pair $\langle x, y\rangle$ such that $x \in X$ and $y \in F\ x$. Establishing

that $\langle x, y\rangle$ is also a member of the given Cartesian product reduces to showing that the components of the pair are in their respective domains. For x this is trivial. For the second component we have to find a witness in $\bigcup_{x\in X} Y_x$ containing y. This is clearly going to be $F\ x$. But we still need $F\ x \in \bigcup_{x\in X} Y_x$ which again demands a witness, this time the index for the family Y. This obviously is x and $F\ x \in Y_x$ follows from our assumption. □

We have established that abstractions are contained in products. What else do they contain? In fact we are not really concerned about the exact shape of their elements but about their behaviour under application. Loosely speaking, a product should only contain functions and applying these functions to sensible arguments should produce well-typed results, i.e., things contained in the dependent result-type domain. Formally we want the following.

Theorem 13. *Given a set $f \in \mathsf{Pi}\ X\ Y$ and an argument $x \in X$, whenever we apply f to x we obtain something in the set Y_x, i.e.,*

$$\mathsf{ap}\ f\ x \in Y_x.$$

Proof. This one is actually rather straightforward as we simply have to use SepE2 to project out the condition of the separation that encodes the product, immediately yielding the desired result. □

We also have the non-dependent case as a corollary.

Corollary 2. *Given a set $f \in X \dot{\longrightarrow} Y$ and an argument $x \in X$, whenever we apply f to x we obtain something in Y, i.e.,*

$$\mathsf{ap}\ f\ x \in Y.$$

Proof. Trivial. □

The last of the typing results we will look at is also the closure property that forced us to use Aczel's non-standard function encoding. The result is proved here in its most general, dependent case.

Theorem 14 (The set **2** is closed under the construction of function spaces)**.** *Given a domain X and a co-domain Y, then the construction of their product* $\mathsf{Pi}\ X\ Y$ *satisfies the following closure property:*

$$(\forall x \in X,\ Y_x \in \mathbf{2}) \to \mathsf{Pi}\ X\ Y \in \mathbf{2}.$$

Proof. To prove this we assume $\forall x \in X,\ Y_x \in \mathbf{2}$ and recall Theorem 8 where Part 5 allows us to conclude $\bigcup_{x \in X} Y_x \in \mathbf{2}$. Then from Theorem 5, Part 4, we obtain $\bigcup \bigcup_{x \in X} Y_x = \emptyset$. This tells us that $\mathsf{Pi}\ X\ Y = \{f \in \mathscr{P}(X \times \emptyset) \mid \forall x \in X,\ \mathsf{ap}\ f\ x \in Y_x\} = \{f \in \mathbf{1} \mid \forall x \in X,\ \mathsf{ap}\ f\ x \in Y_x\}$. Now any separation is a subset of its bounding set (4.7), hence we know that $\mathsf{Pi}\ X\ Y \subseteq \mathbf{1}$. Using the introduction rule for the powerset construction we have $\mathsf{Pi}\ X\ Y \in \mathscr{P}(\mathbf{1})$, which we have shown earlier to be equivalent to $\mathsf{Pi}\ X\ Y \in \mathbf{2}$. This completes the proof. □

We have established a number of necessary results of our function representation to ensure that it behaves sensibly and we will now complete the picture with a number of equational properties. First we will see that both the abstraction and the product construction exhibit extensional equality when provided with metafunctions that are extensionally equal on a given domain.

Lemma 16. *Given a domain set X and two metafunctions F_1 and F_2 on sets which satisfy $\forall x \in X,\ F_1\ x = F_2\ x$, then the sets $\mathsf{lam}\ X\ F_1$ and $\mathsf{lam}\ X\ F_2$ are equal.*

Proof. This proof is trivial. We simply unfold the definitions and see that F_1 and F_2 are only ever applied to arguments from the domain set X. Since we know that F_1 and F_2 agree on X, we immediately obtain the desired equality. □

Lemma 17. *Given a domain set X and two dependent co-domain metafunctions on sets Y_1 and Y_2 which satisfy $\forall x \in X,\ Y_1\ x = Y_2\ x$, then the sets $\mathsf{Pi}\ X\ Y_1$ and $\mathsf{Pi}\ X\ Y_2$ are equal.*

Proof. The key idea is the same we used above. After unfolding definitions we observe two occurrences of Y_1, and respectively two of Y_2. They are again only applied to arguments from X so we know that the posed equality holds. □

These proofs look almost trivial from the mathematical point of view, while they are in fact rather involved in the Coq formalisation. The reason for this discrepancy is an inherent notion of extensionality used by mathematicians that is not guaranteed in type theory. The heart of the problem relates to the fact that we cannot proof-theoretically "see" that the function arguments in question all belong to the appropriate domain. This fact is only uncovered after disassembling all the involved set constructions. At that point the replacement can be performed and the sets have to be reassembled again. The full details can be found in the corresponding proof script.

To conclude we finally show that we have extensionality for all sets that behave functionally.

Theorem 15. *Given two functions $f,\ g \in \mathsf{Pi}\ X\ Y$, we have*

$$(\forall x \in X,\ \mathsf{ap}\ f\ x = \mathsf{ap}\ g\ x) \to f = g.$$

Proof. Since f and g are sets we use set extensionality to show inclusion in both directions. The proofs are symmetric so we only show $f \subseteq g$. Assume we have a set $p \in f$ and we want to establish that $p \in g$. The first step towards this goal is to show that p is a pair.

To achieve this we look at $f \in \mathsf{Pi}\ X\ Y$, from which we know that f is part of a separation and therefore also of the corresponding bounding set: $f \in \mathscr{P}(X \times \bigcup(\bigcup_{x \in X} Y_x))$. This is equivalent to $f \subseteq X \times \bigcup(\bigcup_{x \in X} Y_x)$. Now since $p \in f$ it follows that also $p \in X \times \bigcup(\bigcup_{x \in X} Y_x)$. At this point we can use CProdE2 to conclude that $p = \langle x, y \rangle$ for some $x \in X$ and $y \in \bigcup(\bigcup_{x \in X} Y_x)$.

Now that we have $\langle x, y \rangle \in f$, we can exploit Lemma 13 to derive $y \in \mathsf{ap}\ f\ x$. With the initial assumption we therefore also know that $y \in \mathsf{ap}\ g\ x$. Using Lemma 13 again we obtain $\langle x, y \rangle \in g$ which is the same as $p \in g$, completing the proof. □

4.7 Finite Ordinals & Natural Numbers

To conclude the development we describe the construction of natural numbers in our set theory. In Coq, the natural numbers are built up inductively from a constant $\mathrm{O} : \mathbb{N}$ and a successor function $\mathrm{S} : \mathbb{N} \to \mathbb{N}$ according to the following rules:

$$\textsc{NatO}\ \frac{}{\mathrm{O} : \mathbb{N}} \qquad\qquad \textsc{NatS}\ \frac{n : \mathbb{N}}{\mathrm{S}\ n : \mathbb{N}}$$

As such, these natural numbers are part of our metatheory as well as the type theory we eventually intend to model. The former means that we can (and will) use natural numbers in the process of the set constructions, while the latter suggests a blueprint for the shape our set construction will require. The fact that we can construct natural numbers entails two things. Firstly it is an example for a structured set of infinite size, i.e., we will in principle be able to model infinite types. The type of natural numbers is considered the canonical, non-trivial inductive type in CiC and the fact that we can model it suggests that we will be able to model a large number of "simple" inductive types.

We will first construct the finite ordinals in our set theory and then define an (almost) isomorphism between the finite ordinals we have constructed and Coq's type $\mathbb{N}$. This ensures that our finite ordinals are a suitable representation of natural numbers.

Before we start with the ordinals we recall that our sets are well founded since we have the induction principle Ind. We will make use of the principle to establish that so-called *one-cycles* ($X \in X$) and *two-cycles* ($X \in Y \wedge Y \in X$) are impossible. We use the latter to prove that the successor ordinal construction we will see below is *injective*.

Lemma 18. *The TG set theory does neither contain one-cycles nor two-cycles, formally*

$$\forall X,\ X \notin X, \tag{4.13}$$

$$\forall X\ Y,\ X \in Y \to Y \notin X. \tag{4.14}$$

Proof. We start with (4.13) and immediately apply Ind, which reduces the goal to

$$\forall X,\ (\forall x \in X,\ x \notin x) \to X \notin X$$

We proceed by assuming an X such that $X \in X$ as well as $\forall x \in X,\ x \notin x$ and produce a contradiction by instantiating the universal quantification with X, yielding $X \notin X$. This immediately contradicts our assumption of $X \in X$ and completes the proof of (4.13).

For (4.14) we again immediately apply the induction principle on the membership in set X to reduce the goal to

$$\forall X,\ (\forall x \in X,\ \forall Y,\ x \in Y \to Y \notin x) \to \forall Y,\ X \in Y \to Y \notin X$$

The proof proceeds by contradiction under the following assumptions. We have some X and Y such that $X \in Y$ and $Y \in X$ as well as our induction hypothesis

$$\forall x \in X,\ \forall Y,\ x \in Y \to Y \notin x$$

The trick now is to instantiate x with Y (which we know is in X) and Y with X, which allows us to produce a contradiction from $Y \in X$ and $X \in Y$, both of which we have assumed. This completes the proof. □

We need one more auxiliary construct before we can start the construction proper, namely the iterator function on Coq's type $\mathbb{N}$. It is recursively defined as

```
Fixpoint iter (n:nat) {X:Type} (f: X → X) (x: X) :=
  match n with
  | O ⇒ x
  | S n' ⇒ f (iter n' f x)
  end.
```

The iterator takes a natural number n and applies a given function f n times to a base value x. If we instantiate f and x with the constructors S and O respectively, we are just reconstructing the original natural number. We use this idea to mirror the structure of a natural number with our constructors for ordinals. That is, we will now pose a base ordinal and a successor function, and feed them to the iterator.

We basically require three things, the two constructors we already mentioned and the set of all finite ordinals. The latter is an infinite set and we will describe it with separation so we require a sufficiently large bounding set. As it turns out,

the set of all hereditarily finite sets, in our development $\mathsf{GU}_\emptyset$, does the trick. This leads us to the following definitions.

$$\begin{array}{lcl} \mathsf{ord}_\mathsf{O} & := & \emptyset \\ \mathsf{ord}_\mathsf{S}\ N & := & N \cup \{N\} \\ \mathsf{FinOrd} & := & \{N \in \mathsf{GU}_\emptyset \mid \exists n : \mathbb{N},\ \mathsf{iter}\ n\ \mathsf{ord}_\mathsf{S}\ \mathsf{ord}_\mathsf{O} = N\} \end{array}$$

We have defined the sets $\emptyset$, **1** and **2** as the first three ordinals and it should now be clear how their definitions arise from applying ord_S to ord_O 0, 1 and 2 times respectively.

Before we go any further, we should briefly discuss the definition of FinOrd, as it makes use of a new component in our metatheory, namely the inductive type $\mathbb{N}$. This makes several of the following proofs and constructions quite convenient, but one may justly ask whether we could have done it without resorting to inductive types. In fact we can, forming the least set closed under ord_O and ord_S by representing all sets satisfying the necessary closure properties and then intersecting over them. Further details can be found in the introduction of [Ber58]. Establishing an equivalence between the two encodings should be straightforward but we have not included it in our formalisation.

We are now going to establish a few results about these definitions which ascertain their suitability to encode the inductive type of natural numbers. For a start we point out that our Grothendieck universes are closed under taking the successor ordinal, which follows immediately from the closure results for singleton sets and binary unions. Next we want to look at some "typing" results. For each of the constructs above we show in which set they are contained. For ord_O and $\mathsf{ord}_\mathsf{S}\ N$ this will be FinOrd, while the latter itself lives in $\mathsf{GU}_{\mathsf{GU}_\emptyset}$.

For $\mathsf{ord}_\mathsf{O} = \emptyset$, the proof reduces to showing that $\emptyset \in \mathsf{GU}_\emptyset$, which is trivial, and to provide a natural number witness that collapses the iterator to ord_O, which of course is 0.

Lemma 19. *For any finite ordinal $N \in \mathsf{FinOrd}$, its successor ordinal is also in* FinOrd*, $\mathsf{ord}_\mathsf{S}\ N \in \mathsf{FinOrd}$.*

Proof. From our assumption we know that $N \in \mathsf{GU}_\emptyset$ and via the closure property outlined above $\mathsf{ord}_\mathsf{S}\ N \in \mathsf{GU}_\emptyset$. Furthermore there is a natural number n such that $\mathsf{iter}\ n\ \mathsf{ord}_\mathsf{S}\ \mathsf{ord}_\mathsf{O} = N$. To establish that $\mathsf{ord}_\mathsf{S}\ N \in \mathsf{FinOrd}$ we again have to provide two things, namely that $\mathsf{ord}_\mathsf{S}\ N \in \mathsf{GU}_\emptyset$, which we have already established, and we have to provide a witness for the iterator, which of course is $\mathrm{S}\, n$. □

With a simple induction on natural numbers[7] and the results we have just shown we can also establish that

$$\forall n : \mathbb{N},\ \mathsf{iter}\ n\ \mathsf{ord}_\mathsf{S}\ \mathsf{ord}_\mathsf{O} \in \mathsf{FinOrd}. \tag{4.15}$$

[7]Technically we only require the induction to establish that $\mathsf{iter}\ n\ \mathsf{ord}_\mathsf{S}\ \mathsf{ord}_\mathsf{O} \in \mathsf{GU}_\emptyset$, but since we have the preceding results both the base case and the induction step follow immediately if we perform the induction as suggested.

Finally, we have $\mathsf{FinOrd} \subseteq \mathsf{GU}_\emptyset$ and therefore $\mathsf{FinOrd} \in \mathscr{P}(\mathsf{GU}_\emptyset)$. From $\mathsf{GU}_\emptyset \in \mathsf{GU}_{\mathsf{GU}_\emptyset}$ it additionally follows that $\mathscr{P}(\mathsf{GU}_\emptyset) \in \mathsf{GU}_{\mathsf{GU}_\emptyset}$ and using transitivity we obtain $\mathsf{FinOrd} \in \mathsf{GU}_{\mathsf{GU}_\emptyset}$ as claimed above.

This leads us to one of the more relevant results especially since we want to encode an inductive type with our ordinals. The set ord_O will never coincide with a set that has ord_S as its topmost set former, i.e., they produce distinct sets. Since ord_O is $\emptyset$ it is sufficient to show that everything produced with ord_S is inhabited. From the fact that nothing can be both inhabited and not inhabited at the same time it then follows as a simple corollary that the sets cannot coincide.

Lemma 20. *For any set N, $\mathsf{ord}_\mathsf{S}\ N$ is inhabited.*

Proof. We observe that $\mathsf{ord}_\mathsf{S}\ N = N \cup \{N\}$ and pose N as the witness of the inhabitance. $N \in N \cup \{N\}$ then follows from $N \in \{N\}$ by BinUnionI2, while the latter is an instance of SingI, completing the proof. □

Corollary 3. *Sets produced by ord_O and ord_S are distinct, formally*

$$\forall N : \mathsf{set},\ \mathsf{ord}_\mathsf{S}\ N \neq \mathsf{ord}_\mathsf{O}.$$

Proof. Trivial. □

When we look at our ordinals in the light of an inductive type we also require injectivity of the constructors, which in our case means that we want ord_S to be injective.

Lemma 21. *The set former for the successor ordinal ord_S is injective,*

$$\mathsf{ord}_\mathsf{S}\ N = \mathsf{ord}_\mathsf{S}\ M \to N = M.$$

Proof. We assume $\mathsf{ord}_\mathsf{S}\ N = \mathsf{ord}_\mathsf{S}\ M$ and can easily establish that $N \in \mathsf{ord}_\mathsf{S}\ N$ and hence $N \in \mathsf{ord}_\mathsf{S}\ M$. And similarly $M \in \mathsf{ord}_\mathsf{S}\ N$. Now $N \in \mathsf{ord}_\mathsf{S}\ M$ entails $N = M \vee N \in M$ and symmetrically $M = N \vee M \in N$. For the left disjuncts respectively the result is immediate, so we are left with the case where $N \in M$ and $M \in N$. This however contradicts Lemma 18, completing the proof. □

At this point we can present the isomorphism between our finite ordinals and Coq's natural numbers. An isomorphism consists of two mappings, one from Coq's $\mathbb{N}$ to set and one in the other direction. To be an isomorphism these two mappings have to be inverses of each other, i.e., composing the two in either way should yield the identity. At this point we have to be a bit careful. The forward function will always be applied to natural numbers, and by definition, will only yield sets in FinOrd. For the other direction, however, we observe that our set construction is inherently untyped and hence the function will be defined on the whole type set. We do however only expect correct behaviour for those sets taken from FinOrd. We should kept this in mind when we call the pair of functions an isomorphism.

Given an arbitrary set, then projecting out a natural number and embedding that again as a set is certainly not guaranteed to behave as the identity. Without further ado, here are the definitions:

```
Definition FinOrd_embed (n: nat) : set := iter n ordS ordO.
Definition FinOrd_proj (N: set) : nat :=
  match dit {n: nat | iter n ordS ordO = N} with
    | inl (exist n _) ⇒ n
    | inr _ ⇒ 0
  end.
```

We use the mathematical notations embed n and proj N for FinOrd_embed n and FinOrd_proj N, respectively.

While the embedding operation is straightforward, the projection requires some discussion. Morally we know that we only want to project natural numbers out of sets that are actual embeddings of these but technically the Coq argument type does not provide this information. To get our hands on this information we first construct the strong sum {n: nat | iter n ordS ordO = N}, which is a form of informative existential quantification. This states that there exists a natural number n which yields the embedding N. This is what we expect, but N could have come from anywhere else, in which case the sum type would be empty. We can now perform a case analysis on the inhabitance of the constructed sum with the help of the assumed decision procedure outlined in Section 2.5. When the sum is inhabited and we were given a sensible set N, we are also given the witness n as well as a proof of the body of the sum. We simply return the witness and ignore the proof. If the sum turns out to be empty we just default to 0. We require this branch of the case split to have a fully defined function, but the reader should be aware that in the remainder we never expect to actually enter it.

Before we can show that these two functions yield an isomorphism we require the following inversion lemma of the embedding.

Lemma 22. *The embedding of the natural numbers n and m is injective, formally*

$$\mathsf{embed}\ n = \mathsf{embed}\ m \to n = m.$$

Proof. We proof this by induction on n and a case analysis on m, yielding four cases.

The first case is trivial since $0 = 0$. For the second case we have $\mathsf{embed}\ 0 = \mathsf{embed}\ (\mathrm{S}\, m')$ which is equivalent to $\mathsf{ord_O} = \mathsf{ord_S}\ (\mathsf{iter}\ m'\ \mathsf{ord_S}\ \mathsf{ord_O})$. This is contradictory as we have seen in Corollary 3.

We now look at the inductive step. When $m = 0$ we have the same contradiction as in the second case so we are left with the following scenario. Our induction hypothesis states $\forall m : \mathbb{N}$, $\mathsf{embed}\ n = \mathsf{embed}\ m \to n = m$ and we also know for some m' that $\mathsf{ord_S}\ (\mathsf{embed}\ n) = \mathsf{ord_S}\ (\mathsf{embed}\ m')$. What we have to show is $\mathrm{S}\, n = \mathrm{S}\, m'$. For the latter it is sufficient to show $n = m'$. Lemma 21 allows

us to conclude $\mathsf{embed}\ n = \mathsf{embed}\ m'$. The desired result then follows from our induction hypothesis (with m instantiated to m'), completing the proof. □

We now show that embedding and projection form an isomorphism,

Lemma 23. *The function* $\mathsf{proj} \circ \mathsf{embed}$ *is the identity at* $\mathbb{N}$*, formally*

$$\forall n : \mathbb{N},\ \mathsf{proj}\ (\mathsf{embed}\ n) = n.$$

Proof. We unfold the definition of proj and perform a case split on the underlying inhabitance test. In the positive case the function collapses to the witness k, reducing the goal to $k = n$. While the function ignored the embedded proof of the equality $\mathsf{iter}\ k\ \mathsf{ord_S}\ \mathsf{ord_O} = \mathsf{embed}\ n$, we still have that knowledge as an assumption from the case analysis. From this we conclude the desired result with the help of Lemma 22.

If the sum is empty however, it is sufficient to show inhabitance and produce a contradiction. This means that we have to find a k such that $\mathsf{iter}\ k\ \mathsf{ord_S}\ \mathsf{ord_O} = \mathsf{embed}\ n$. This is simply n, completing the proof. □

The shape of the proof we have just seen illustrates the dummy rôle of the second branch of the projection function. We split on whether the particular strong sum was inhabited or not. In the first case the proof could go through cleanly. Assuming the sum was not inhabited led to a contradiction.

Lemma 24. *The function* $\mathsf{embed} \circ \mathsf{proj}$ *is the identity on sets encoding finite ordinals, formally*

$$\forall N \in \mathsf{FinOrd},\ \mathsf{embed}\ (\mathsf{proj}\ N) = N.$$

Proof. From the definition of FinOrd we know that there is a natural number k such that $N = \mathsf{embed}\ k$. Substituting for N reduces the claim to $\mathsf{embed}\ (\mathsf{proj}\ (\mathsf{embed}\ k)) = \mathsf{embed}\ k$. Using Lemma 23 we reduce this to the trivial $\mathsf{embed}\ k = \mathsf{embed}\ k$. □

We complete this section with a recursion principle for our finite ordinals that is closely modelled on the recursion principle we have for Coq's natural numbers. There are a number of properties of our projection and embedding functions that are used in the correctness proof of this principle. We state these here without proof but the inclined reader can find the details in the corresponding proof scripts.

$$\mathsf{proj}\ \mathsf{ord_O} = 0 \tag{4.16}$$

$$N = \mathsf{embed}\ n \to \mathsf{ord_S}\ N = \mathsf{embed}\ (\mathrm{S}\, n) \tag{4.17}$$

$$N \in \mathsf{FinOrd} \to \mathsf{proj}\ (\mathsf{ord_S}\ N) = \mathrm{S}\ (\mathsf{proj}\ N) \tag{4.18}$$

$$N \in \mathsf{FinOrd} \to 0 = \mathsf{proj}\ N \to N = \mathsf{ord_O} \tag{4.19}$$

$$N \in \mathsf{FinOrd} \to \mathrm{S}\, m = \mathsf{proj}\ N \to \exists M \in \mathsf{FinOrd},\ N = \mathsf{ord_S}\ M \tag{4.20}$$

We pose the following definition where nat_rect is the recursion principle for Coq's type $\mathbb{N}$ with the following type.

```
nat_rect : forall P : nat → Type,
  P 0 → (forall n : nat, P n → P (S n)) → forall n : nat, P n

Definition FinOrd_rect (z f N: set) : set :=
  nat_rect (λ _ . set) z
    (λ n: nat . λ R: set . ap (ap f (FinOrd_embed n)) R ) (FinOrd_proj N).
```

The mathematical notations for these two recursion principles are $\mathsf{rect}_{\mathsf{nat}}$ and $\mathsf{rect}_{\mathsf{ord}}$ respectively.

This definition is designed to outwardly appear to only operate on sets so we have to use our function encoding from Section 4.6 to represent and use the binary step function f. Its first argument is the current natural number, while the second represents the recursive argument. The actual recursion is performed with Coq's natural numbers. Since the numerical argument is an ordinal though we first have to project out the corresponding number. Then at every step we are given a natural number which we have to embed into the ordinals to provide the first argument to f in the correct format.

Our recursion principle comes with suitable computation rules that are easy to show with the properties of our projection function mentioned above.

$$\mathsf{rect}_{\mathsf{ord}}\ z\ f\ \mathsf{ord}_{\mathsf{O}} = z$$
$$N \in \mathsf{FinOrd} \longrightarrow \mathsf{rect}_{\mathsf{ord}}\ z\ f\ (\mathsf{ord}_{\mathsf{S}}\ N) = \mathsf{ap}\ (\mathsf{ap}\ f\ N)\ (\mathsf{rect}_{\mathsf{ord}}\ z\ f\ N)$$

For the first we simply unfold the definition of $\mathsf{rect}_{\mathsf{ord}}$. This reveals an occurrence of $\mathsf{proj}\ \mathsf{ord}_{\mathsf{O}}$ which we know to be equal to 0. The instance of $\mathsf{rect}_{\mathsf{nat}}$ then just collapses down to z as required. For the second we again unfold $\mathsf{rect}_{\mathsf{ord}}$ on the left and then exploit (4.18) to pull the $\mathsf{ord}_{\mathsf{S}}$ through the projection operation. Applying the computation rules of $\mathsf{rect}_{\mathsf{nat}}$ once leads to the desired equality.

Lastly we show a typing result for our recursor on ordinals and thus complete the correctness of the construct.

Theorem 16. *For any unary metafunction A on sets and encoded ordinal $N \in \mathsf{FinOrd}$, and for any $z \in A\ \mathsf{ord}_{\mathsf{O}}$ and $f \in \Pi N \in \mathsf{FinOrd}$, $A\ N \dot{\longrightarrow} A\ (\mathsf{ord}_{\mathsf{S}}\ N)$ the definition of $\mathsf{rect}_{\mathsf{ord}}$ satisfies*

$$\mathsf{rect}_{\mathsf{ord}}\ z\ f\ N \in A\ N.$$

Proof. In principle we would like to prove this result by induction on N but we do not even have a suitable induction principle for our ordinals. The use of $\mathsf{rect}_{\mathsf{nat}}$ in the definition of $\mathsf{rect}_{\mathsf{ord}}$ however suggests that we may delegate the induction to Coq's natural numbers instead. Unfolding the definition of $\mathsf{rect}_{\mathsf{ord}}$ yields

$$\mathsf{rect}_{\mathsf{nat}}\ (\lambda\_.\mathsf{set})\ z\ (\lambda n.\lambda R.\mathsf{ap}\ (\mathsf{ap}\ f\ (\mathsf{embed}\ n))\ R)\ (\mathsf{proj}\ N) \in A\ N.$$

Before we can perform the induction we have to represent N as the corresponding natural numbers and we have to take care of both occurrences. Since $N \in \mathsf{FinOrd}$ we can expand the second occurrence of N to $\mathsf{embed}\ (\mathsf{proj}\ N)$. We now have two occurrences of $\mathsf{proj}\ N$ which we are going to call k. Our goal now is

$$\mathsf{rect_{nat}}\ (\lambda\_.\mathsf{set})\ z\ (\lambda n.\lambda R.\mathsf{ap}\ (\mathsf{ap}\ f\ (\mathsf{embed}\ n))\ R)\ k \in A\ (\mathsf{embed}\ k)$$

and we can finally perform the induction on k.

The base case is fairly straightforward. For $k = 0$, the instance of $\mathsf{rect_{nat}}$ collapses down to simply z, reducing the goal to $z \in A\ (\mathsf{embed}\ 0)$ or equivalently $z \in A\ \mathsf{ord_O}$, which is one of our assumptions.

The step with $k = \mathrm{S}\,k'$ is more involved. The induction hypothesis is

$$\mathsf{rect_{nat}}\ (\lambda\_.\mathsf{set})\ z\ (\lambda n.\lambda R.\mathsf{ap}\ (\mathsf{ap}\ f\ (\mathsf{embed}\ n))\ R)\ k' \in A\ (\mathsf{embed}\ k')$$

and our goal reduces to

$$\mathsf{ap}\ (\mathsf{ap}\ f\ (\mathsf{embed}\ k'))\ (\mathsf{rect_{nat}}\ (\lambda\_.\mathsf{set})\ z\ (\lambda n.\lambda R.\mathsf{ap}\ (\mathsf{ap}\ f\ (\mathsf{embed}\ n))\ R)\ k') \in A\ (\mathsf{embed}\ (\mathrm{S}\,k'))$$

At this point we recall that $f \in \Pi N \in \mathsf{FinOrd},\ A\ N \dot{\longrightarrow} A\ (\mathsf{ord_S}\ N)$ and clearly $\mathsf{embed}\ k' \in \mathsf{FinOrd}$. We apply f to $(\mathsf{embed}\ k')$ and obtain $(\mathsf{ap}\ f\ (\mathsf{embed}\ k')) \in A\ (\mathsf{embed}\ k') \dot{\longrightarrow} A\ (\mathsf{ord_S}\ (\mathsf{embed}\ k'))$. We can push the one occurrence of $\mathsf{ord_S}$ into the embedding and then apply the result to the induction hypothesis, yielding the final result. □

We are still missing one fact to be sure that our construction of FinOrd can serve as the interpretation of the inductive type $\mathbb{N}$, namely analysis by cases. In our case this means that we know that a set in FinOrd is always either $\mathsf{ord_O}$ or $\mathsf{ord_S}\ M$ for some $M \in \mathsf{FinOrd}$.

Theorem 17. *Every set $N \in \mathsf{FinOrd}$ is a well-formed ordinal, i.e.,*

$$N = \mathsf{ord_O} \vee \exists M \in \mathsf{FinOrd},\ N = \mathsf{ord_S}\ M.$$

Proof. Since $N \in \mathsf{FinOrd}$ we know that we have an $n : \mathbb{N}$ such that $N = (\mathsf{iter}\ n\ \mathsf{ord_S}\ \mathsf{ord_O})$. We can now perform a case analysis on n. For $n = 0$ the iterator reduces to $\mathsf{ord_O}$, in which case the goal holds. If instead we have $n = \mathrm{S}\,m$ for some $m : \mathbb{N}$. This entails that $N = (\mathsf{iter}\ (\mathrm{S}\,m)\ \mathsf{ord_S}\ \mathsf{ord_O}) = \mathsf{ord_S}\ (\mathsf{iter}\ m\ \mathsf{ord_S}\ \mathsf{ord_O})$. We will now prove the right disjunct and pose $(\mathsf{iter}\ m\ \mathsf{ord_S}\ \mathsf{ord_O})$ as the required witness. We have just established that the requested equality holds, while $(\mathsf{iter}\ m\ \mathsf{ord_S}\ \mathsf{ord_O}) \in \mathsf{FinOrd}$ follows from (4.15), completing the proof. □

With the knowledge that we will be able to interpret basic inductive types we conclude the presentation of the technical development. As mentioned in the beginning and throughout the text, any technical omissions and further details can be found in the proof scripts available on the author's web page[8].

[8] `http://www.ps.uni-saarland.de/~jkaiser/`

CHAPTER 5

CONCLUSION

In this work we have presented the formal development of a set theory. We began with an introduction to the metatheory, a considerably strengthened variant of the Calculus of Constructions, equipped with classical reasoning and a powerful choice principle. On top of this foundation we have layered an axiomatisation of Tarski-Grothendieck set theory and then developed the set theory to include a number of interesting constructs. The most prominent among these is likely our development of typable functions and function spaces.

We would like to point out the development presented here is not a general purpose set library. Consider for example the absence of binary intersection or set difference. Instead, what we present here is a tool box developed with a particular purpose in mind. As with any formal development there is always the question about how to set up the intermediate lemmas and, more generally, how to structure the overall development. Depending on the overall goal there are a number of ways to address these issues. We do not claim that the approach chosen here is optimal with respect to our plans but it is the result of several iterations and so far it appears to have reached a stable point that can hold up to our needs.

We have omitted a number of smaller intermediate constructions. The reader would not have gained additional insights from these but they were necessary nonetheless. We want to mention one of these here, namely strong sums and dependent ordered pairs, which will become important in the long run. The interested reader can find the development in the Coq source files.

This development can be seen as a larger exercise in formal classical reasoning. Up to this point, the inclusion of the law of the excluded middle did not lead to ground breaking short cuts or improvements in the formal derivations but all in

all it appears that the proofs could be kept reasonably small and at a level that is easy to grasp. It is rather unclear whether this would have been possible in a purely intuitionistic setting. Constructions which benefited in particular were the separation or restricted comprehension over sets, as well as the construction of ordinals and the isomorphism to Coq's natural numbers. The latter relied on the decidability of inhabitance of every type, which in turn is a consequence of classical reasoning in the presence of choice.

5.1 Related Work

Before we close we want to briefly discuss other results in the field of this work.

First and foremost is the work of Barras who recently constructed a formalised, intuitionistic model of CC_ω (which is ECC without strong sums) [Bar10] and his recent extensions to CiC [Bar12]. He constructed the model in Coq as well and our work was largely motivated by his developments. In contrast to his work we are of course classical which at times allows us to reason more directly.

With respect to the formalisation of set theory in general we of course have to mention Paulson's extensive work on Isabelle/ZF [Pau93, Pau95, Pau08] as well as the Mizar project [Rud92]. The latter employs Tarski-Grothendieck set theory as its basic formalism, albeit axiomatised with Tarski's *Axiom A* instead of Grothendieck universes [Try90]. In addition to these large developments there are various set libraries available in Coq, each of which was designed with a particular application in mind. And of course each of the formalised model constructions we look at next, comes with its own inbuilt set theory as well.

One of the early insights with respect to the construction of set-theoretic models is Reynolds' discovery that an impredicative, polymorphic universe like **Prop** only admits trivial classical models [Rey84]. With respect to such proof-irrelevant models we have the ongoing work of Werner et al. [Wer97, MW02, LW11]. They discovered that the combination subtyping and impredicativity in theories like ECC result in unforeseen complications. Part of their work has been formalised in Coq and partial solutions have been suggested. Similar to our work, Gordon sketches an embedding of ZFC into higher-order logic and vice versa, albeit with the goal of combining the benefits of both formalisms [Gor96]. And finally we have a rigorous approach by Aczel which relates type and set theories with respect to proof-theoretic strength [Acz98].

5.2 Future Work

As we have mentioned early on, we would like to construct a formal, set-theoretic model for type theories. In this light the development presented here forms the foundation of this goal. The type theories we aim for are Luo's ECC [Luo94] and the Calculus of (co)inductive Constructions [PPM89] underpinning Coq. We are

going to use the ordinal **2** to interpret the lowest type universe **Prop**, which leads to proof irrelevant models, similar to the models in [Acz98] and [MW02]. With this in mind we have chosen Aczel's function representation which exhibits a certain closure property for function spaces mapping into **2**. This step should allow us to interpret the function spaces of the type theory directly with our set-theoretic functions. For the latter we proved a number of results that will closely map to the typing rules of the target type theory. With respect to the non-propositional type universes we are going to rely on our set-theoretic Grothendieck universes. As of yet it is still unclear though which set-theoretic universe will be used to interpret which type universe. There seems to be a certain degree of freedom in this selection and preliminary observations hint at a number of possible independence results we could obtain from this.

All in all we have taken some initial steps towards the construction of an ECC model and there should not be too many surprises along the way. With respect to CiC it is still unclear, though, how to best deal with inductive types and in particular inductive propositions. For inductive (non-propositional) types the ordinal construction presented here suggests at least one possible approach, while inductive propositions are somewhat mysterious. In most models they are tacitly ignored and it is not clear for what reason. Given that the impredicativity of **Prop** is known to complicate the construction of set-theoretic models, it appears worthwhile to investigate their nature.

BIBLIOGRAPHY

[Ack25] Wilhelm Ackermann. Begründung des "tertium non datur" mittels der Hilbertschen Theorie der Widerspruchsfreiheit. *Mathematische Annalen*, 93:1–36, 1925.

[Acz78] Peter Aczel. The type theoretic interpretation of constructive set theory. In Angus Macintyre, Leszek Pacholski, and Jeff Paris, editors, *Logic Colloquium '77*, volume 96 of *Studies in Logic and the Foundations of Mathematics*, pages 55–66. Elsevier, 1978.

[Acz98] Peter Aczel. On relating type theories and set theories. In *TYPES*, pages 1–18, 1998.

[Bar10] Bruno Barras. Sets in Coq, Coq in sets. *Formalized reasoning*, 3(1), 2010.

[Bar12] Bruno Barras. Semantical investigations in intuitionistic set theory and type theories with inductive families. Habilitation Thesis manuscript, received in private communication, 2012.

[BDL06] Sandrine Blazy, Zaynah Dargaye, and Xavier Leroy. Formal verification of a C compiler front-end. In Jayadev Misra, Tobias Nipkow, and Emil Sekerinski, editors, *FM*, volume 4085 of *Lecture Notes in Computer Science*, pages 460–475. Springer, 2006.

[Ber58] Paul Bernays. *Axiomatic set theory : with a historical introduction by Abraham A. Fraenkel*. Amsterdam : North-Holland Publ. Co., 1958.

[Bro23] Luitzen E. J. Brouwer. *On the significance of the principle of excluded middle in mathematics, especially in function theory*, pages 334–345. In van Heijenoort [vH00], 1923.

[Can74] Georg Cantor. Über eine Eigenschaft des Inbegriffs aller reellen algebraischen Zahlen. *Journal für die reine und angewandte Mathematik*, 77:258–262, 1874.

[CF58] Haskell B. Curry and Robert Feys. *Combinatory logic*, volume 1. North-Holland Publ. Co., 1958.

[CH85] Thierry Coquand and Gérard P. Huet. Constructions: A higher order proof system for mechanizing mathematics. In Bruno Buchberger, editor, *European Conference on Computer Algebra (1)*, volume 203 of *Lecture Notes in Computer Science*, pages 151–184. Springer, 1985.

[CH88] Thierry Coquand and Gérard P. Huet. The calculus of constructions. *Information and Computation*, 76(2/3):95–120, 1988.

[Fra22] Abraham A. Fraenkel. Zu den Grundlagen der Cantor-Zermeloschen Mengenlehre. *Mathematische Annalen*, 86:230–237, 1922.

[Fre79] Gottlob Frege. *Begriffsschrift, a formula language, modeled upon that of arithmetic, for pure thought*, pages 1–82. In van Heijenoort [vH00], 1879.

[Gab62] Pierre Gabriel. Des catégories abéliennes. *Bulletin de la Société Mathématique de France*, 90:323–448, 1962.

[Gir72] Jean-Yves Girard. *Interprétation fonctionelle et élimination des coupures de larithmétique dordre supérieur*. PhD thesis, Université Paris VII, 1972.

[Gon07] Georges Gonthier. The four colour theorem: Engineering of a formal proof. In Deepak Kapur, editor, *ASCM*, volume 5081 of *Lecture Notes in Computer Science*, page 333. Springer, 2007.

[Gor96] Mike Gordon. Set theory, higher order logic or both? In Gerhard Goos, Juris Hartmanis, Jan Leeuwen, Joakim Wright, Jim Grundy, and John Harrison, editors, *Theorem Proving in Higher Order Logics*, volume 1125 of *Lecture Notes in Computer Science*, pages 191–201. Springer Berlin / Heidelberg, 1996.

[Gri90] Timothy G. Griffin. A formulae-as-type notion of control. In *Proceedings of the 17th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, POPL '90, pages 47–58, New York, NY, USA, 1990. ACM.

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

[GV72] Alexander Grothendieck and Jean-Louis Verdier. Exposé I: Prefaisceaux. In Michael Artin, editor, *Théorie des Topos et Cohomologie Etale des Schémas*, volume 269 of *Lecture Notes in Mathematics*. Springer Berlin / Heidelberg, 1972.

[Hey74] Arend Heyting. *Mathematische Grundlagenforschung Intuitionismus Beweistheorie*. Springer Berlin / Heidelberg, 1974.

[HJ99] Karel Hrbacek and Thomas J. Jech. *Introduction to set theory*. Pure and applied mathematics. CRC Press, 3. ed., rev. and expanded edition, 1999.

[How80] William A. Howard. The formulae-as-types notion of construction. In J. Roger Hindely and Jonathan P. Seldin, editors, *To H. B. Curry: Essays on Combinatory Logic*. Academic Press, Boston, MA, 1980.

[Hur95] Antonius J.C. Hurkens. A simplification of Girard's paradox. In Mariangiola Dezani-Ciancaglini and Gordon Plotkin, editors, *Typed Lambda Calculi and Applications*, volume 902 of *Lecture Notes in Computer Science*, pages 266–278. Springer Berlin / Heidelberg, 1995.

[Kol32] Andrej N. Kolmogorov. Zur Deutung der intuitionistischen Logik. *Mathematische Zeitschrift*, 35:58–65, 1932.

[Kru65] Arthur H. Kruse. Grothendieck universes and the super-complete models of Shepherdson. *Compositio Mathematica*, 17:96–101, 1965.

[Kur21] Casimir Kuratowski. Sur la notion de l'ordre dans la théorie des ensembles. *Fundamenta Mathematicae*, 2(1), 1921.

[Ler09a] Xavier Leroy. Formal verification of a realistic compiler. *Communications of the ACM*, 52(7):107–115, 2009.

[Ler09b] Xavier Leroy. A formally verified compiler back-end. *Journal of Automated Reasoning*, 43(4):363–446, 2009.

[Luo94] Zhaohui Luo. *Computation and reasoning: a type theory for computer science*. Oxford University Press, Inc., New York, NY, USA, 1994.

[LW11] Gyesik Lee and Benjamin Werner. Proof-irrelevant model of CC with predicative induction and judgmental equality. *Logical methods in computer science*, 7(4), 2011.

[ML75] Per Martin-Löf. An intuitionistic theory of types: Predicative part. *Studies in Logic and the Foundations of Mathematics*, 80:73–118, 1975.

[MW02] Alexandre Miquel and Benjamin Werner. The not so simple proof-irrelevant model of CC. In *TYPES*, pages 240–258, 2002.

[Pau93] Lawrence C. Paulson. Set theory for verification: I. from foundations to functions. *Journal of Automated Reasoning*, 11(3):353–389, 1993.

[Pau95] Lawrence C. Paulson. Set theory for verification. II: Induction and recursion. *Journal of Automated Reasoning*, 15(2):167–215, 1995.

[Pau08] Lawrence C. Paulson. The relative consistency of the axiom of choice–mechanized using Isabelle/ZF. *Logic and Theory of Algorithms*, pages 486–490, 2008.

[PPM89] Frank Pfenning and Christine Paulin-Mohring. Inductively defined types in the calculus of constructions. In Michael G. Main, Austin Melton, Michael W. Mislove, and David A. Schmidt, editors, *Mathematical Foundations of Programming Semantics*, volume 442 of *Lecture Notes in Computer Science*, pages 209–228. Springer, 1989.

[Rey84] John C. Reynolds. Polymorphism is not set-theoretic. In Gilles Kahn, David B. MacQueen, and Gordon Plotkin, editors, *Semantics of Data Types*, volume 173 of *Lecture Notes in Computer Science*, pages 145–156. Springer Berlin / Heidelberg, 1984.

[Rud92] Piotr Rudnicki. An overview of the Mizar project. In *Proceedings of the 1992 Workshop on Types for Proofs and Programs*, pages 311–330, 1992.

[Rus02] Bertrand Russell. *Letter to Frege*, pages 124–125. In van Heijenoort [vH00], 1902.

[Sko22] Thoralf Skolem. *Some remarks on axiomatized set theory*, pages 290–301. In van Heijenoort [vH00], 1922.

[Sko29] Thoralf Skolem. *Über einige Grundlagenfragen der Mathematik*. Skrifter utgitt av Det Norsk videnskaps-akademi i Oslo, I. Matematisk-naturvidenskapelig klasse, no. 4. I kommisjon hos J. Dybwad, Oslo, 1929.

[Tar38] Alfred Tarski. Über unerreichbare Kardinalzahlen. *Fundamenta Mathematicae*, 30(1):68–89, 1938.

[Try90] Andrzej Trybulec. Tarski Grothendieck set theory. *Journal of Formalized Mathematics*, 1(1):9–11, 1990.

[vH00] Jean van Heijenoort, editor. *From Frege to Gödel: a sourcebook in mathematical logic*. toExcel, Lincoln, NE, USA, 2000.

[Wag85] Stan Wagon. *The Banach-Tarski Paradox*. Cambridge University Press, Cambridge, UK, 1985.

[Wer97] Benjamin Werner. Sets in types, types in sets. In *TACS*, pages 530–346, 1997.

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

[Zer08] Ernst Zermelo. Untersuchungen über die Grundlagen der Mengenlehre. I. *Mathematische Annalen*, 65:261–281, 1908.

[Zer30] Ernst Zermelo. Über Grenzzahlen und Mengenbereiche. *Fundamenta Mathematicae*, 16(1):29–47, 1930.