The Eudoxus reals constructed in homotopy type theory

Fokma, S.J. (Anna)

Award date:
2021

The Eudoxus reals constructed in homotopy type theory

Author:
S.J. Fokma

Supervisor:
J.W. Portegies

Eindhoven
March 23, 2021

Eindhoven University of Technology
Department of Mathematics and Computer Science

Abstract

In this thesis we present a construction of the Eudoxus reals in homotopy type theory. We start by giving an overview of homotopy type theory and discuss generalizations of inductive types working our way up to higher inductive-inductive types. Next we construct the Eudoxus reals of A'Campo [1] as a quotient type. Unusual about this construction is that the reals are constructed directly from the integers, without first defining the rationals. We show that this quotient type is an Archimedean ordered field, but we cannot prove its completeness. Hence we take its completion as in the HoTT book [41] and generalized by Gilbert [25], to end up with a Cauchy complete Archimedean ordered field. We prove that the completion of the Eudoxus reals is an *initial* Cauchy complete Archimedean ordered field and hence equals the Cauchy reals from the HoTT book. We formalize most of the proof that the Eudoxus reals are an Archimedean ordered field in Coq.

Contents

1 Introduction

Homotopy type theory claims to be a (constructive) alternative to set theory in which almost all of mathematics can be represented [41, p. 6]. It extends ordinary type theory by adding the univalence axiom but also by defining a new kind of types, higher inductive types. The so-called HoTT book uses these higher inductive types to construct the Cauchy reals, which form a Cauchy complete Archimedean ordered field [41, Sec. 11.3]. The construction is similar to the usual definition of the reals as equivalence classes of Cauchy sequences of rationals.

Our goal is to gain a better understanding of these higher inductive types and therefore we will use them to construct the real numbers based on the Eudoxus reals. This is a construction of the reals from classical mathematics based on just the integers [1].

Our first approach will try to construct the reals using a higher inductive type known as a quotient type. The HoTT book [41, p. 381] namely claims that such a type cannot be used if we want to prove that the result is Cauchy complete and we want to understand why this is the case. Since we can indeed not show that the Eudoxus reals are Cauchy complete, we thereafter use Gilbert's generalization [25] of the construction of the Cauchy reals. This does result in a construction of the real numbers, of which we then want to know how it compares to the Cauchy reals. This second approach uses a higher *inductive-inductive* type and the HoTT book states that it is preferable to other approaches using for instance setoids, the axiom of choice and Dedekind complete reals [41, p. 381]. We would like to understand what aspect of this type justifies this statement.

Since the proof assistant Coq is based on type theory, we additionally want to formalize part of our results in Coq. However, Coq does not support higher inductive types natively, which means that we need to find a method which does enable this.

Below we will first take a short look at (homotopy) type theory to see how it differs from set theory. In particular we will discuss what it entails for homotopy type theory to be constructive. Then we will discuss how the reals have been formulated so far in homotopy type theory and give a general overview of the different formulations of the Eudoxus reals. We finish with an overview of this thesis.

Type theory

With type theory we will refer to specifically Martin-Löf type theory, which was developed in order to formalize the constructive mathematics of for instance Bishop [34]. It extended earlier type theories by allowing proofs to occur in propositions, so that types could depend on other types [34].

Type theory is an alternative to set theory and hence we will not work with sets as we are used to, but with types. Rules tell us how to construct a type and its terms, and these introduction rules imply in turn other rules which tell us how we can work with elements of the type.

A familiar example of an (inductive) type is the type of natural numbers. We can namely generate the natural numbers $\mathbb{N}$ from an element we denote by 0, and a successor function $\mathsf{succ} : \mathbb{N} \to \mathbb{N}$. This implies that the type of natural numbers consists of 0, $\mathsf{succ}\,0$, $\mathsf{succ}(\mathsf{succ}\,0)$ and so on. If we now denote $\mathsf{succ}\,0$ by 1 and the term $\mathsf{succ}(\mathsf{succ}\,0)$ by 2 we see our familiar natural numbers arise.

When we want to define a function f on the natural numbers we do so by the induction principle. Hence we should define $f(0)$ and, given $f(n)$ we should define $f(\mathsf{succ}\,n)$. In particular we can define functions recursively. If for instance we want to define multiplication by two, we state that $f(0) = 0$ and $f(\mathsf{succ}\,n) = \mathsf{succ}(\mathsf{succ}(f(n)))$. For any type such induction and recursion principles follow from its definition.

Homotopy type theory

In homotopy theory, two paths $p, q : [0,1] \to X$ are said to be homotopic if there exists a homotopy between the two paths. That is, if there exists a continuous collection of paths between $p(x)$ and $q(x)$ for all elements x of the unit-interval.

Homotopy theory is a branch of topology which is at first glance unrelated to type theory since it concerns itself with paths in topological spaces. However, in 2006 Voevodsky [42], and in 2009 Awodey and Warren [5], combined type theory and homotopy type theory to form homotopy type theory. In this homotopic interpretation of type theory, types are seen as spaces and terms as points in these spaces. The disjoint union from topology for instance corresponds to the type $A + B$ where there exists a point $\mathsf{inl}(a)$ for all a in A and a point $\mathsf{inr}(b)$ for all b in B.

In addition we can see equalities as paths, which means that if $a = b$ that there exists a path from a to b. Since there can exist multiple paths between two points a and b this also means that there can

exist multiple terms of $a = b$. Just as in homotopy theory, we can then wonder whether there exists a path, that is a homotopy, between these paths.

To resemble the case for points, we want functions to be equal if there exists a path, or homotopy, between them. We can assume so by the univalence axiom, which was added to homotopy type theory by Voevodsky. In general it roughly states that two types are isomorphic, or equivalent, if and only if they are equal. This enlarges the concept of being equal to being isomorphic.

It is a convention we already often apply when working with isomorphic structures and make no explicit difference anymore between the two structures. As an example, we almost never distinguish the natural numbers seen as subset of the integers and seen as subset of the rationals. The univalence axiom justifies this, because if two types A and B are equivalent, there exists a path $A = B$. Hence if a property holds for A we can use this path to transport the property to B.

Higher inductive types

When working with homotopy type theory we can extend the notion of an inductive type to higher inductive types. This means that we not only construct a type by stating what points it contains, but also what paths it contains. An example is the circle which consists of a point base and a path $\mathsf{loop} : \mathsf{base} = \mathsf{base}$, where we clearly see the resemblance with its construction as CW-complex.

These higher inductive types come just as ordinary types with induction principles. If for instance we want to define a function on the sphere, we should not only state where to send base but also state what happens with loop in a coherent way. Using for example the induction principle of the sphere, we can calculate its (higher) homotopy groups [41, Sec. 8.3]. This has lead to the development of synthetic homotopy theory [14, 13], but these higher inductive types can also be used to represent type theory within itself [3].

Higher inductive types and its generalizations are still an active field of research, which will be the subject of Section 3.

Constructivism in homotopy type theory

In type theory mathematical statements corresponds to types, which is known as 'propositions as types'. We can for instance capture the statement that 'if x, y are two natural numbers such that $x < y$ then there exists z such that $x + z = y$' in the type

$$\prod_{x,y:\mathbb{N}} \left(x < y \to \sum_{z:\mathbb{N}} x + z = y \right),$$

if we read $\prod$ as 'for all' and $\sum$ as 'there exists'. Hence, if A is a type then a can be considered an element of A, but also as a proof of the statement A. In particular this makes type theory a constructive form of mathematics, since to prove a statement we need to *construct* a term of the corresponding type.

We already mentioned earlier that type theory was developed in order to formalize Bishop's constructive mathematics, whose work [9] is seen as foundational in this field. Constructive mathematics is mentioned by Krebbers and Spitters [30] as one option to close the gap between (numerical) algorithms and their actual implementation. An example of this gap can be seen in the proof of Kepler conjecture by Hales [26], which used computer calculations and of which the referees were hence (only) 99% percent sure that the proof was correct.

The main idea of constructive mathematics is that in order to prove that a certain object exists, we need to construct such an object. Hence if we want to prove (P and Q) we should prove both P and Q, just like we are used to from classical mathematics. However, if we want to prove (P or Q) we should prove that either P holds or that Q holds.

An example where the difference between constructive and classical mathematics becomes clear is what Bishop calls the 'limited principle of omniscience' [10, p.11]. It states that for all functions $f : \mathbb{Z} \to \{0, 1\}$ there exists either an integer m such that $f(m) = 1$ or that $f(n) = 0$ for all integers n. In classical mathematics, this is certainly true because if there does not exists an element which maps to 1 and not all elements map to 0, then we clearly have a contradiction. This reasoning tells us however not how to decide for a function $\mathbb{Z} \to \{0, 1\}$ whether it is the zero function or which element maps to 1. Hence, such a proof is not allowed in constructive mathematics. We can also not expect that a constructive proof exists for this principle, and if it would then it would imply, among others, the Riemann hypothesis [10, p.7-8].

This might seem to limit mathematics, but Bishop has already shown that (large parts of) analysis can be carried out in a constructive manner. Moreover, the Univalent Foundations Program claims that 'essentially all mathematics' can be represented in homotopy type theory as well as it is normally represented in set theory [41, p. 6]. A nice introduction to constructive mathematics can also be found in a publication of Bauer [6].

Coq

Another option Krebbers and Spitters mention to close the gap between algorithms and their implementations are proof assistants [30]. These are computer programs which aid the user in constructing a valid proof by verifying the correctness of the proof. They can however also help by for instance enabling the user to search for existing theorems in a library. Type theory can be used as a basis for these assistants of which Agda, Coq and Lean are examples. In this project we will use Coq to formalize part of our results.

Reals

Now that we have given an impression of homotopy type theory, we can discuss how we can construct the real numbers. The real numbers can be defined as any complete Archimedean ordered field. In a classical setting the completeness can refer to both Dedekind and Cauchy completeness as these notions are then equivalent. However, working in a constructive setting without the axiom of countable choice these two are not [41, p. 419]. We are interested in constructive reals with respect to Cauchy completeness.

Usually the reals are defined as equivalence classes of Cauchy sequences of rational numbers, which we then call the completion of the rationals. To prove that this definition is indeed Cauchy complete, a Cauchy sequence in the quotient set is lifted to a sequence in the original set. Then in the original set, a representative of the limit is constructed. However, the lifting of a sequence requires the axiom of choice or the law of excluded middle [41, p. 381], which are both not available in homotopy type theory.

A first solution is to construct the reals as a setoid, which is a type where both an equivalence and equality relation are used. O'Conner [36] defines the completion of arbitrary premetric spaces using these setoids and hence in particular defines the reals. The disadvantage of setoids is however that all the theory on abstract algebra should explicitly be developed for setoids and that we cannot work with the equivalence relation as we would with the equality relation in homotopy type theory [25].

The HoTT book [41] avoids setoids by constructing the reals as a higher inductive-inductive type. There the reals itself are constructed simultaneously with the equivalence relation on them. This allows the identification of equivalent elements to occur immediately when defining them, instead of later in a quotient set. Hence the problem of lifting a sequence is avoided, and it can be shown that this construction is indeed Cauchy complete [41, Th. 11.3.49]. Moreover, the HoTT book show that their Cauchy reals are an initial Cauchy complete Archimedean ordered field [41, Th. 11.3.50].

At the time that the HoTT book was written, Coq did not yet support inductive-inductive types. Therefore, the Cauchy reals were only formalized in Coq in 2016 by Gilbert [25] after a workaround for this problem was posed by Sozeau.

Eudoxus reals

Instead of defining the real numbers based on the rationals, we can also directly construct them from the integers in a construction known as the Eudoxus reals. The construction is centered around what we will call slopes, which are functions $\mathbb{Z} \to \mathbb{Z}$ differing only finitely from linear functions. That is, a function $f : \mathbb{Z} \to \mathbb{Z}$ is a slope if there exists an integer d such that for all integers m and n we have $|f(m+n)-f(m)-f(n)| \leq d$. Addition is then defined by pointwise addition and multiplication is defined as the composition. Two slopes λ and μ are said to be equivalent if they differ only finitely in the sense that there exists an integer S such that for all integers n we have $|\lambda(n) - \mu(n)| \leq S$. The set of Eudoxus reals is then defined as equivalence classes of slopes. In a classical set-theoretic setting this results in a construction of the real numbers [39, 27, 1, 4, 35].

Over the years, the construction has popped up in multiple places as sometimes independent discoveries. The most recent publications are those by Arthan [4] and A'Campo [1], and more details of the construction by A'Campo can be found in the master's thesis of Mizrahi [35]. A'Campo independently discovered this construction, but Arthan based his results on earlier unpublished work by Stephen Schanuel. Based on the work of Schanuel, Street [39] had already published a short overview of the

construction and Harrison [27] published a more detailed account of the construction compatible with the HOL theorem prover.

The main definitions do not differ between the authors, but some aspects of their approaches did and we will shortly mention a few. First of all, not all completeness proofs were correct from the beginning [40, 20]. Moreover, A'Campo was the first to introduce well-adjusted slopes, for which the slope bound d equals 1, and prove that any slope is equivalent to such a slope. Harrison's approach differed from the others since he constructed the *positive* reals from the naturals and from there constructed all reals. The name 'Eudoxus reals' which we will use was introduced by Schanuel and Arthan [4], but not used by all other authors. Similarly, the term slopes is due to A'Campo and has earlier been referred to as quasi-homomorphisms (Street), almost homomorphisms (Arthan) and almost-additive functions (Harrison).

We will follow the construction of A'Campo and the detailed thesis from Mizrahi, because A'Campo's publication was the first we were aware of.

Overview of the thesis

In this project we will first discuss the necessary preliminaries from homotopy type theory in Section 2. Then we will discuss how we can generalize (higher) inductive types in Section 3. Since our final goal is to construct a Cauchy complete Archimedean ordered field using the Eudoxus reals, we will state in Section 4 what it entails in homotopy type theory to be such an ordered field.

We show in Section 5 that the Eudoxus reals are an Archimedean ordered field. Since we are working with a constructive notion of ordered field, some of our proofs differ from the ones given by A'Campo and Mizrahi. We also have to introduce the structure of a lattice and prove results related to the apartness relation. Moreover, as far as we know all previous constructions of the Eudoxus reals show its completeness by proving the least-upper-bound-property. We will however show in Section 6 why this is hard to work with in our constructive setting and instead show that the type of slopes is Cauchy complete. Unfortunately, we must indeed conclude that this does not imply that the Eudoxus reals itself are complete.

Hence in Section 7 we consider the completion of the Eudoxus reals, which is defined similarly as the Cauchy reals from the HoTT book. That this type is Cauchy complete and Archimedean follows from respectively Gilbert [25] and its definition. We argue that this is an ordered field, and prove that it is an *initial* Cauchy complete Archimedean ordered field. This implies that the completion of the Eudoxus reals equals the Cauchy reals from the HoTT book.

We formalized most of Section 5 in Coq and some implementation details of this can be found in Section 9. In Section 8 we however first discuss how higher inductive types are defined in Coq.

In Section 10 we discuss what elements are missing in our construction and in the formalization in Coq. We finish this thesis with a conclusion in Section11.

2 Homotopy type theory

In this section we discuss the basics of homotopy type theory. Most of this is contained in the first chapter of the HoTT book [41], to which we also refer for more details. Starting in Section 2.4 we will also cover material from the second chapter of the HoTT book.

One of the two most fundamental notions, called judgments, in type theory is for a term a to be of type A, denoted $a : A$ and also stated as that 'a inhabits A'. This can be compared to the notion in logic of a proposition having a proof. Intuitively, especially in examples as $n : \mathbb{N}$ or $x : \mathbb{R}$, this corresponds to $a \in A$ from set theory. There is however one important difference here being that $a \in A$ is a proposition which could be true or false and hence can be proven. This is not the case for $a : A$ which is a judgment or atomic statement. We cannot speak of a term a on its own as it should always be introduced as a term of a certain type, unlike the situation in set theory.

Since the judgment $a : A$ is such a foundational statement, it will bind in notation more loosely than other operations we will introduce. Hence the notation in $p : (a = b)$ can be simplified to $p : a = b$. The reason that this causes no confusion is that $(p : a) = b$ does not have any meaning, because the left-hand side, a judgment, cannot be equal to anything else.

2.1 Equality

In type theory we distinguish two forms of equality. The first equality is definitional equality. We denote this by $\equiv$ and one can think of this as 'being equal by definition'. For example, when defining a function $f : \mathbb{N} \to \mathbb{N}$ as $f(x) :\equiv x^2$, we use $:\equiv$ to denote that the left and right hand side are equal by definition. It then follows that $f(2) \equiv 2^2$. To decide whether two terms are definitionally equal, we just expand definitional equations. Definitional equality is the second judgment in type theory and hence cannot be proven, just as $a : A$. Likewise, $\equiv$ binds more loosely than anything else. As an example $P \equiv (a = b)$ can be written as $P \equiv a = b$, because the statement $(P \equiv a) = b$ has no meaning.

The second form of equality is propositional equality and we denote it by $=$. This equality can be proven and can be part of an hypothesis, in contrast to the definitional equality. To continue our example, we can write $f(2) = 4$ after proving that $2^2 = 4$.

2.2 Universes

As stated at the beginning of this section, we cannot discuss an element in isolation as every term is a term of some type, in contrast to set theory. Similarly also a type must be an element of some type. Therefore we introduce *universes*, which are types whose terms are also types. Such a universe needs to be the term of a type as well, but to avoid contradictions we cannot assume a universe to include itself.

Hence we define a hierarchy of universes consisting of $\mathcal{U}_0, \mathcal{U}_1, \mathcal{U}_2, \ldots$, where $\mathcal{U}_i : \mathcal{U}_{i+1}$ and $A : \mathcal{U}_{i+1}$ if $A : \mathcal{U}_i$. In practice, the level of the universe is often omitted and so the statement that $A : \mathcal{U}$ means that there exists a universe $\mathcal{U}_i$ such that $A : \mathcal{U}_i$. Sometimes the statement $A : \mathcal{U}$ is also stated as that A is a type.

2.3 The definition of a type

So far we have only mentioned types, but not yet how we define a type. To define a type we have to state a certain set of rules which state how the type and its terms can be constructed and how to work with these terms. This set of rules consists of:

- the formation rule which specifies how we can define this new type given types we already have,
- the introduction rules (or constructors) which specify how we can introduce (or construct) terms of this type,
- the elimination rules which specify how to use terms of this type to construct terms of other types, and
- the computation rules which specify how eliminators act on a term given by the introduction rules.

In the next few section we will discuss a few standard types so that we can work with them later. There we will also see how these rules may look like.

2.3.1 Function types

The first type we will define is the type of *functions* and these are similar to the notion of functions we are used to. Given a type $A : \mathcal{U}$ and a type $B : \mathcal{U}$, we form the type $A \to B : \mathcal{U}$ of functions from A to B. Note that the notation we introduced for : implies that this should be read as $(A \to B) : \mathcal{U}$. Then there are two possible, and equivalent, introduction rules. The first is by direct definition, where we construct a function $f : A \to B$ by $f(x) :\equiv \Phi$. Here x is a free variable and Φ an expression involving x, such that if $x : A$ then $\Phi : B$. An example would be $f(x) :\equiv x^2$. The second option is to use λ-abstraction, where we construct the function, without giving it a name, as $(\lambda(x : A).\Phi) : A \to B$. Here again x is a variable and Φ an expression involving x such that if $x : A$ then $\Phi : B$. Often the λ notation is shortened to $\lambda x.\Phi$. The previous example would in this case be written as $\lambda x.x^2$.

The elimination rule states that if $a : A$ and $f : A \to B$ that $f(a) : B$. The computation rule states that $f(a) = \Phi'$ in the case of the direct definition, where Φ' is the expression Φ but with all x replaced by a. In the case of λ-abstraction it states that $(\lambda x.\Phi)(a) = \Phi'$, where Φ' is again the expression Φ with x replaced by a.

We use currying to define functions in multiple variables as iterated functions. If we for instance want to define a function with inputs in A and B to C, then we define it as a function which maps $a : A$ to a function in $B \to C$. Therefore we will denote the type of functions from A and B to C by $A \to (B \to C)$, although we will often forget about the brackets and write $A \to B \to C$. To limit the use of brackets we will write $f(a, b)$ instead of $(f(a))(b)$.

A special case of function types is when we choose the codomain to be a universe $\mathcal{U}$. In this case we often refer to a function as a *family of types*.

2.3.2 Dependent function types

As a generalization of functions, we can define *dependent functions*. We can form the type of dependent functions, which is denoted by $\prod_{x:A} B(x) : \mathcal{U}$, if we are given a type $A : \mathcal{U}$ and a type family $B : A \to \mathcal{U}$. The introduction rule is similar to the one of (independent) function types: given an expression $\Phi : B(x)$ which may depend on $x : A$, we define $f : \prod_{x:A} B(x)$ by either $f(x) :\equiv \Phi$ for $x : A$, or $f :\equiv \lambda x.\Phi$. When we apply a function $f : \prod_{x:A} B(x)$ to a term $a : A$, we obtain a term $f(a) : B(a)$. The term $f(a)$ can be computed as the expression Φ, where every occurrence of x is replaced by a.

Although it is unusual in many other parts of mathematics, in type theory a $\prod$ concerns the rest of the expression. This is a generalization of the notation we used for independent functions. Hence when we write $\prod_{x:A} B(x) \to C(x)$, this should be read as $\prod_{x:A}(B(x) \to C(x))$.

2.3.3 Product types

Next we define the *(cartesian) product type*, which is similar to a product of two sets. We form the product type $A \times B : \mathcal{U}$ from two types $A, B : \mathcal{U}$. Then from a term $a : A$ and a term $b : B$ we construct the pair $(a, b) : A \times B$. The elimination rule states that given a function $g : A \to B \to C$ we can define a function $f : A \times B \to C$ by using the computation rule $f((a, b)) :\equiv (g(a))(b)$. Another way of stating this elimination rule is to define the (independent) eliminator, which we call the recursor. The recursor, denoted $\mathsf{rec}_{A\times B}$, is of type

$$\mathsf{rec}_{A\times B} : \prod_{C:\mathcal{U}} (A \to B \to C) \to A \times B \to C$$

and is defined as

$$\mathsf{rec}_{A\times B}(C, g, (a, b)) :\equiv (g(a))(b),$$

where $C : \mathcal{U}$, $g : A \to B \to C$ and $(a, b) : A \times B$. In particular, the recursor enables us to define the projection functions

$$\mathsf{pr}_1 :\equiv \mathsf{rec}_{A\times B}(A, \lambda a.\lambda b.a) \text{ and } \mathsf{pr}_2 :\equiv \mathsf{rec}_{A\times B}(B, \lambda a.\lambda b.b)$$

We can generalize this independent eliminator to a dependent eliminator, which is called induction and denoted $\mathsf{ind}_{A\times B}$. It extends recursion by allowing the type C to be a function of $A \times B$ instead of a fixed type. Hence induction is of type

$$\mathsf{ind}_{A\times B} : \prod_{C:A\times B\to\mathcal{U}} \left(\prod_{x:A} \prod_{y:B} C((x, y)) \right) \to \prod_{z:A\times B} C(z)$$

and is defined for $C : A \times B \to \mathcal{U}$, $g : \prod_{x:A} \prod_{y:B} C((x,y))$ and $(a,b) : A \times B$ by

$$\mathsf{ind}_{A\times B}(C, g, (a,b)) :\equiv (g(a))(b).$$

At first these reduction and, especially, induction principles can seem quite intimidating, but over time one grows accustomed to this compact way of writing down how to define a function on a new type. Note that if we take C to be constant, that the induction operator reduces to the recursion operator, and that hence induction is indeed a generalization of recursion.

So far, we have used that all terms of the cartesian product $A \times B$ are indeed pairs and that it is hence enough to define a function on the product only on these pairs. Using the recursor, we can actually prove this by showing there exists an element of type

$$\prod_{x:A\times B} ((\mathsf{pr}_1(x), \mathsf{pr}_2(x)) =_{A\times B} x).$$

2.3.4 Dependent pair types types

The product type is an instance of the more general *dependent pair type* $\sum_{x:A} B(x)$ in which the type of the second element of the pair may depend on the first element. It can be constructed from a type $A : \mathcal{U}$ and a family $B : A \to \mathcal{U}$. Then given $a : A$ and $b : B(a)$ we can form the term $(a,b) : \sum_{x:A} B(x)$. We see that if B is constant, a dependent pair type indeed reduces to a product type. The recursion principle is of type

$$\mathsf{rec}_{\sum_{x:A} B(x)} : \prod_{C:\mathcal{U}} \Big(\prod_{x:A} B(x) \to C \Big) \to \Big(\sum_{x:A} B(x) \Big) \to C$$

and defined by

$$\mathsf{rec}_{\sum_{x:A} B(x)}(C, g, (a,b)) :\equiv (g(a))(b)$$

for $C : \mathcal{U}$, $g : \prod_{x:A} B(x) \to C$ and $(a,b) : \sum_{x:A} B(x)$. The induction principle is of type

$$\mathsf{ind}_{\sum_{x:A} B(x)} : \prod_{C:(\sum_{x:A} B(x))\to\mathcal{U}} \Big(\prod_{x:A} B(x) \to C((a,b)) \Big) \to \Big(\sum_{x:A} B(x) \Big) \to \prod_{p:\sum_{x:A} B(x)} C(p)$$

and defined by

$$\mathsf{ind}_{\sum_{x:A} B(x)}(C, g, (a,b)) :\equiv (g(a))(b)$$

for $C : (\sum_{x:A} B(x)) \to \mathcal{U}$, $g : \prod_{x:A} B(x) \to C((A,b))$ and $(a,b) : \sum_{x:A} B(x)$.

A $\sum$ scopes just as $\prod$ over the remaining part of the expression. Hence $\sum_{x:A} B(x) \to C(x)$ should be read as $\sum_{x:A}(B(x) \to C(x))$.

2.3.5 Coproduct types

Given the types $A, B : \mathcal{U}$ we can also form their *coproduct* $A + B$. Since this types behaves similarly as the disjoint union in set theory, it is also referred to as the *disjoint union* of A and B. There are two ways to introduce terms of $A + B$: firstly, if $a : A$ then $\mathsf{inl}(a) : A + B$, and secondly, if $b : B$ then $\mathsf{inr}(b) : A + B$. The recursion principle

$$\mathsf{rec}_{A+B} : \prod_{C:\mathcal{U}} (A \to C) \to (B \to C) \to A + B \to C$$

is defined by

$$\mathsf{rec}_{A+B}(C, g_0, g_1, \mathsf{inl}(a)) :\equiv g_0(a)$$

and

$$\mathsf{rec}_{A+B}(C, g_0, g_1, \mathsf{inr}(b)) :\equiv g_1(b)$$

for $C : \mathcal{U}$, $g_0 : A \to C$, $g_1 : B \to C$, $a : A$ and $b : B$. The induction principle

$$\mathsf{ind}_{A+B} : \prod_{C:(A+B)\to\mathcal{U}} \Big(\prod_{a:A} C(\mathsf{inl}(a)) \Big) \to \Big(\prod_{b:B} C(\mathsf{inr}(b)) \Big) \to \prod_{x:A+B} C(x)$$

is defined by

$$\mathsf{ind}_{A+B}(C, g_0, g_1, \mathsf{inl}(a)) :\equiv g_0(a)$$

and

$$\mathsf{ind}_{A+B}(C, g_0, g_1, \mathsf{inr}(b)) :\equiv g_1(b)$$

for $C : (A + B) \to \mathcal{U}$, $g_0 : \prod_{a:A} C(\mathsf{inl}(a))$, $g_1 : \prod_{b:B} C(\mathsf{inr}(b))$, $a : A$ and $b : B$.

2.3.6 Empty type

The next two types we will define do not need other types in order to formed, in contrast to the (co)product types. We define the *empty type* as the type $\mathbf{0} : \mathcal{U}$ without any introduction rule for terms. The recursion and induction principles are hence relatively short:

$$\mathsf{rec}_{\mathbf{0}} : \prod_{C:\mathcal{U}} \mathbf{0} \to C$$

and

$$\mathsf{ind}_{\mathbf{0}} : \prod_{C:\mathbf{0}\to\mathcal{U}} \prod_{z:\mathbf{0}} C(z).$$

They both do not have a defining equation since we did not introduce any terms of $\mathbf{0}$.

The recursion principle is nevertheless important since it corresponds to an important logical principle called *the principle of explosion* or *ex falso quodlibet*. A classical formulation of this principle is that if both propositions P and $\neg P$ are true, then any statement is true. To see that the recursion principle corresponds to this principle, we first define for types $P : \mathcal{U}$ the type $\neg P$ as $P \to \mathbf{0}$. Let $C : \mathcal{U}$ now be a type representing any kind of statement and assume that we have both P and $\neg P$. This implies that we have an element of $\mathbf{0}$. Hence it then follows from the recursion principle that C holds.

2.3.7 Unit type

Next we define the *unit type* $\mathbf{1} : \mathcal{U}$ with the single introduction rule $\star : \mathbf{1}$. Its recursor is of type

$$\mathsf{rec}_{\mathbf{1}} : \prod_{C:\mathcal{U}} C \to \mathbf{1} \to C$$

and defined by $\mathsf{rec}_{\mathbf{1}}(C, c, \star) :\equiv c$ for $C : \mathcal{U}$ and $c : C$. Its induction principle

$$\mathsf{ind}_{\mathbf{1}} : \prod_{C:\mathbf{1}\to\mathcal{U}} C(\star) \to \prod_{x:\mathbf{1}} C(x)$$

is defined by $\mathsf{ind}_{\mathbf{1}}(C, c, \star) :\equiv c$. Just as for a product type we have a so-called uniqueness principle, which in this case is a term of $\prod_{x:\mathbf{1}} x = \star$.

2.3.8 Identity types

The identity type is a type in type theory which does not have an obvious equivalent in set theory and hence is specific to type theory. We will first state the formation and introduction rules, before we will make a short note on what this type represents.

Given a type $A : \mathcal{U}$ and two terms $a, b : A$, we can form the identity type $\mathsf{Id}_A(a, b) : \mathcal{U}$. This type is also denoted as $a =_A b$ or just $a = b$. For each $a : A$ we can form the term $\mathsf{refl}_a : a = a$. Hence, for each element a, the type $a = a$ is inhabited. If $a \equiv b$ the type $a = b$ is also inhabited, because if $a : A$ is judgmentally equal to b, the type $a = b$ is judgmentally equal to $a = a$. The recursion principle

$$\mathsf{rec}_{=_A} : \prod_{C:A\to A\to\mathcal{U}} \Big(\prod_{a:A} C(a, a)\Big) \to \prod_{a,b:A} a = b \to C(a, b)$$

is defined by

$$\mathsf{rec}_{=_A}(C, c, a, a, \mathsf{refl}_a) :\equiv c$$

for $C : A \to A \to \mathcal{U}$, $c : \prod_{a:A} C(a, a)$ and $a : A$.

The induction principle

$$\mathsf{ind}_{=_A} : \prod_{C:\prod_{a,b:A}(a=_A b)\to\mathcal{U}} \Big(\prod_{a:A} C(a, a, \mathsf{refl}_a)\Big) \to \prod_{a,b:A} \prod_{p:a=_A b} C(a, b, p)$$

is defined by

$$\mathsf{ind}_{=_A}(C, c, a, a, \mathsf{refl}_a) :\equiv c(a)$$

for $C : \prod_{a,b:A}(a =_A b) \to \mathcal{U}$, $c : \prod_{a:A} C(a, a, \mathsf{refl}_a)$ and $a : A$.

The (interpretation of) the identity type is closely related to Section 2.8. There we will discuss that a proposition can have multiple terms which all represent different proofs. The same holds for the identity type, since there can exists multiple ways to prove two terms are equal. In homotopy type theory specifically, the identity type $a = b$ can be seen as the space of all paths between a and b.

2.3.9 Natural numbers and integers

After defining these abstract types, we can now also define more concrete types as the natural numbers and the integers.

We start by defining the type of natural numbers, $\mathbb{N} : \mathcal{U}$. The elements are constructed by $0 : \mathbb{N}$ and the successor function $\mathsf{succ} : \mathbb{N} \to \mathbb{N}$. For convenience, we write $1 :\equiv \mathsf{succ}(0)$, $2 :\equiv \mathsf{succ}(\mathsf{succ}(0))$ and so forth as usual. Its recursion principle is of type

$$\mathsf{rec}_{\mathbb{N}} : \prod_{C:\mathcal{U}} C \to (\mathbb{N} \to C \to C) \to \mathbb{N} \to C,$$

defined by

$$\begin{aligned} \mathsf{rec}_{\mathbb{N}}(C, c_0, c_s, 0) &:\equiv c_0 \text{ and} \\ \mathsf{rec}_{\mathbb{N}}(C, c_0, c_s, \mathsf{succ}(n)) &:\equiv c_s(n, \mathsf{rec}_{\mathbb{N}}(C, c_0, c_s, n)), \end{aligned}$$

where $C : \mathcal{U}$, $c_0 : C$, $c_s : \mathbb{N} \to C \to C$ and $n : \mathbb{N}$. The induction principle is of type

$$\mathsf{ind}_{\mathbb{N}} : \prod_{C:\mathbb{N}\to\mathcal{U}} C(0) \to \left(\prod_{n:\mathbb{N}} C(n) \to C(\mathsf{succ}(n))\right) \to \prod_{n:\mathbb{N}} C(n),$$

and defined by

$$\begin{aligned} \mathsf{ind}_{\mathbb{N}}(C, c_0, c_s, 0) &:\equiv c_0 \text{ and} \\ \mathsf{ind}_{\mathbb{N}}(C, c_0, c_s, \mathsf{succ}(n)) &:\equiv c_s(n, \mathsf{ind}_{\mathbb{N}}(C, c_0, c_s, n)), \end{aligned}$$

where $C : \mathbb{N} \to \mathcal{U}$, $c_0 : C(0)$, $c_s : \prod_{n:\mathbb{N}} C(n) \to C(\mathsf{succ}(n))$ and $n : \mathbb{N}$. Although slightly hidden in the notation, we see in both the recursion and induction principle that we can define a function f on the naturals by defining $f(0)$ and $f(\mathsf{succ}(n))$ given $f(n)$. Hence these principles coincide with the usual induction on the integers from classical mathematics.

In addition we define the type $\mathbb{N}_+ :\equiv \sum_{n:\mathbb{N}} 0 < n$. Often we will be sloppy in the notation and write $n : \mathbb{N}_+$ when we actually mean $n :\equiv \mathsf{pr}_1\, p$ where $p : \mathbb{N}_+$.

We define the type of integers, $\mathbb{Z} : \mathcal{U}$, again following the HoTT book [41, Section 6.10], as the type

$$\sum_{z:\mathbb{N}\times\mathbb{N}} r(z) = z,$$

where $r : \mathbb{N} \times \mathbb{N} \to \mathbb{N} \times \mathbb{N}$ is defined for all $a, b : \mathbb{N}$ by

$$r(a, b) = \begin{cases} (a - b, 0) \text{ if } a \geq b \\ (0, b - a) \text{ otherwise.} \end{cases}$$

We will often be sloppy in the notation and write $z : \mathbb{Z}$ when we actually mean $(z, p) : \mathbb{Z}$ for some $p : r(z) = z$. Moreover, we will identify $n : \mathbb{N}$ with $(n, 0, q)$ where $q : r(n, 0) = (n, 0)$. For $n : \mathbb{N}$ we will also denote by $-n$ the triple $(0, n, s)$ where $s : r(0, n) = (0, n)$.

We will not state the induction principle which follows directly from this definition of $\mathbb{Z}$. However, we will state the following 'induction principle' which is implied by the original induction principle [41, Lem. 6.10.12], since this one is similar to induction on $\mathbb{N}$ and is of more use to us. This induction principle is of type

$$\mathsf{ind}_{\mathbb{Z}} : \prod_{C:\mathbb{N}\to\mathcal{U}} C(0) \to \left(\prod_{n:\mathbb{N}} C(n) \to C(\mathsf{succ}(n))\right) \to \left(\prod_{n:\mathbb{N}} C(-n) \to C(-\mathsf{succ}(n))\right) \to \prod_{n:\mathbb{N}} C(n),$$

and is defined by

$$\begin{aligned} \mathsf{ind}_{\mathbb{Z}}(C, c_0, c_s, c_p, 0) &:\equiv c_0, \\ \mathsf{ind}_{\mathbb{Z}}(C, c_0, c_s, c_p, \mathsf{succ}(n)) &:\equiv c_s(n, \mathsf{ind}_{\mathbb{Z}}(C, c_0, c_s, n)) \text{ and} \\ \mathsf{ind}_{\mathbb{Z}}(C, c_0, c_s, c_p, -\mathsf{succ}(n)) &:\equiv c_p(n, \mathsf{ind}_{\mathbb{Z}}(C, c_0, c_s, -n)), \end{aligned}$$

where $C : \mathbb{N} \to \mathcal{U}$, $c_0 : C(0)$, $c_s : \prod_{n:\mathbb{N}} C(n) \to C(\mathsf{succ}(n))$, $c_p : \prod_{n:\mathbb{N}} C(-n) \to C(-\mathsf{succ}(n))$ and $n : \mathbb{N}$.

Using the induction principles for $\mathbb{N}$ and $\mathbb{Z}$, we can define the usual operations as addition and multiplication, and prove properties about these operations. For this we refer however to the HoTT book [41, Section 1.9, 6.10]. The result is that we can work with the natural numbers and the integers as we are accustomed to.

2.4 Paths and transport

In the next sections we will discuss a few aspects of homotopy type theory building up to the univalence axiom. In homotopy type theory the theory of types is viewed from a homotopical perspective. The main idea is that a type can be seen as a space, a term as a point and an equality as a path. An identity type then coincides with the space of all paths between the two corresponding points and we can concatenate equalities as we would paths. This interpretation is developed in the second chapter of the HoTT book [41]. In this section we will state a few basic results on how to work with paths.

A first result states that paths can be reversed, or more formally that equality is symmetrical [41, Lem. 2.1.1].

Lemma 2.1. *Let $A : \mathcal{U}$ be a type and let $x, y : A$. Then there exists a function*

$$(x = y) \to (y = x).$$

We denote this function by $p \mapsto p^{-1}$ and the function satisfies $\mathsf{refl}_x = \mathsf{refl}_x^{-1}$ for all $x : A$.

We can also concatenate paths, by which we mean that equality is transitive [41, Lem. 2.1.2].

Lemma 2.2. *Let $A : \mathcal{U}$ be a type and let $x, y, z : A$. Then there exists a function*

$$(x = y) \to (y = z) \to (x = z).$$

We denote this function by $p \mapsto q \mapsto p \cdot q$ and the function satisfies $\mathsf{refl}_x \cdot \mathsf{refl}_x = \mathsf{refl}_x$ for all $x : A$.

To show that these operations behave as we would expect if we see equalities as paths, we have the following lemma [41, Lem. 2.1.4]. In topological terms, this lemma shows for instance that if we concatenate a path with its inverse that the resulting path is homotopic to the constant path.

Lemma 2.3. *Let $A : \mathcal{U}$ be a type and let $x, y, z, w : A$. Let $p : x = y$, $q : y = z$ and $r : z = w$ be paths. Then we have the following:*

- $\mathsf{refl}_x \cdot p = p = p \cdot \mathsf{refl}_y$
- $p \cdot p^{-1} = \mathsf{refl}_x$ *and* $p^{-1} \cdot p = \mathsf{refl}_y$
- $(p^{-1})^{-1} = p$
- $(p \cdot q) \cdot r = p \cdot (q \cdot r)$.

Note that the above lemma states the existence of certain paths between paths, which in topology we would call homotopies. The path $s : p = p \cdot \mathsf{refl}_y$ is for instance a paths between elements of the type $x = y$.

The next lemma shows that functions respect equalities. From a topological point of view, the lemma states that functions preserve paths and are hence 'continuous' [41, Lem. 2.2.1].

Lemma 2.4. *Let $A, B : \mathcal{U}$ and let $f : A \to B$ be a function. Then for all $x, y : A$ we have a function*

$$\mathsf{ap}_f(x =_A y) \to (f(x) =_B f(y)).$$

This function satisfies for all $x : A$ that $\mathsf{ap}_f(\mathsf{refl}_x) \equiv \mathsf{refl}_{f(x)}$.

The function ap_f is often denoted by f itself. Hence it depends on the input of f whether we refer to f itself or to ap_f, but this causes no confusion in general.

Lemma 2.4 is only concerned with independent functions $f : A \to B$ but we would like to have a similar result for dependent functions. However, we cannot simply replace $f : A \to B$ in the lemma by a dependent function $f : \prod_{x:A} B(x)$, because $f(x) : B(x)$ and $f(y) : B(y)$ are elements of different types. Therefore, we need the following function to transport an element in $B(x)$ to $B(y)$ given a path $x = y$ [41, Lem 2.3.1].

Lemma 2.5. *Let $A : \mathcal{U}$ be a type, let $B : A \to \mathcal{U}$ be a type family and let $p : x =_A y$. Then there is a function $p_* : P(x) \to P(y)$.*

This function p_* is called a transport function and is sometimes also denoted by $\mathsf{transport}^B(p, \cdot)$ to indicate the type family B in which the transport takes place.

Using the transport function we can define the dependent alternative to ap, which we denote by apd [41, Lem. 2.3.4].

Lemma 2.6. *Let $A : \mathcal{U}$ be a type and let $B : A \to \mathcal{U}$ be a type family. Then for all $x, y : A$ there exists a function*

$$\mathsf{apd}_f : \prod_{p:x=y} p_*(f(x)) =_{B(y)} f(y).$$

The HoTT book shows how ap and apd are related and proves more lemmas related to these functions. We will however have little use for all this theory and hence we refer an interested reader to the HoTT book [41, Sec. 2.3].

Later on, when dealing with higher inductive types, we will also define functions by how they act on paths. However, since the existence of ap is a lemma, we cannot use $:\equiv$ in these cases [41, Section 6.2]. Hence if p, q are paths and we want to define a function f which sends p to q, we write $f(p) := q$ (or $\mathsf{ap}_f(p) := q$). We will however not encounter these situations frequently.

2.5 Equivalences

In topology the role of equivalence is fulfilled by homotopy equivalences. These are, if A, B are topological spaces, functions $f : A \to B$ such that there exists $g : B \to A$ such that $f \circ g$ and $g \circ f$ are both homotopic to the respective identity functions. This can be shown by constructing a homotopy, which is in the first case a continuous collection of paths between $(f \circ g)(x)$ and x for all $x : B$. In our setting of homotopy type theory, we can define a similar notion of a homotopy between functions.

Definition 2.7. *Let $A : \mathcal{U}$ be a type, let $B : A \to \mathcal{U}$ be a type family and let $f_1, f_2 : \prod_{x:A} B(x)$ be two dependent functions. Then a homotopy from f_1 to f_2 is a function of type*

$$f_1 \sim f_2 :\equiv \prod_{x:A} (f_1(x) =_{B(x)} f_2(x)).$$

We can also define a quasi-inverse in homotopy type theory, which is similar to a homotopy equivalence in topology.

Definition 2.8. *Let $A, B : \mathcal{U}$ be types and let $f : A \to B$ be a function. A quasi-inverse of f is a triple (g, α, β) consisting of a function $g : B \to A$ and homotopies $\alpha : f \circ g \sim \mathrm{id}_B$ and $\beta : g \circ f \sim \mathrm{id}_A$.*

We will however not define a function as an equivalence if it is a quasi-inverse, which explains the name. The proof that a function is a homotopy equivalence is namely not unique and we want the statement that a function is an equivalence to not depend on the proof [41, Th. 4.1.3]. Intuitively, we want the statement that a function is an equivalence to only capture whether it is an equivalence and not the extra information such as how this is shown. This is exactly the idea behind the definition of a mere proposition, which we will discuss in Section 2.8.

There are multiple options to define an equivalence such that the statement that a function is an equivalence has at most one inhabitant. The HoTT book defines three in Chapter 4, of which we will state here the one which is most similar to the quasi-inverse.

Definition 2.9. *Let $A, B : \mathcal{U}$ be types and let $f : A \to B$ be a function. Then f is bi-invertible if there exists $g : B \to A$ such that $f \circ g \sim \mathrm{id}_B$ and $h : B \to A$ such that $h \circ f \sim \mathrm{id}_A$. If f is bi-invertible, we say that f is an equivalence and that A and B are equivalent, denoted $A \simeq B$.*

From the definitions we see that if a function f is a quasi-inverse then it is also an equivalence. If (g, α, β) is namely the quasi-inverse of f, then (g, α, g, β) shows that f is bi-invertible. Hence we can show that two types A, B are equivalent by constructing a function $A \to B$ and its quasi-inverse.

2.6 Characterization of paths

Up to equivalence we can characterize paths in basic types as products and dependent pairs of which we will discuss four examples here.

Intuitively we expect that, if $A, B : \mathcal{U}$ are types, a pair $(a_1, b_1) : A \times B$ is equal to $(a_2, b_2) : A \times B$ if and only if $a_1 = a_2$ and $b_1 = b_2$. By applying pr_1 and pr_2 to $(a_1, b_1) = (a_2, b_2)$ we see that this indeed implies $a_1 = a_2$ and $b_1 = b_2$. The following lemma shows that also the other implication holds [41, Th. 2.6.2].

Lemma 2.10. *Let $A, B : \mathcal{U}$ be types and let $x, y : A$. Then*

$$(x =_{A \times B} y) \simeq (\mathsf{pr}_1(x) =_A \mathsf{pr}_1(y)) \times (\mathsf{pr}_2(x) =_B \mathsf{pr}_2(y)).$$

Similarly we can also characterize paths in a dependent pair type. We have to be careful however, since if (a_1, b_1) and (a_2, b_2) are both elements of a dependent pair type $\sum_{x:A} B(x)$, then b_1 and b_2 are not necessarily elements of the same type. This is similar to the reason why we needed to introduce apd next to ap. Hence again we will use the transport function to be able to talk about equality [41, Th. 2.7.2].

Lemma 2.11. *Let $A : \mathcal{U}$ be a type and let $B : A \to \mathcal{U}$ be a type family. Let $w, w' : \sum_{x:A} B(x)$. Then there is an equivalence*

$$(w = w') \simeq \prod_{p:\mathsf{pr}_1(w)=\mathsf{pr}_1(w')} p_*(\mathsf{pr}_2(w)) = \mathsf{pr}_2(w').$$

For elements in a coproduct $A + B$, we would expect that they are equal if they are equal in A if they are of the form $\mathsf{inl}(a)$ or equal in B if they are of the form $\mathsf{inr}(n)$. This is indeed something we can show [41, Sec. 2.12].

Lemma 2.12. *Let $A, B : \mathcal{U}$ and let $a_1, a_2 : A$ and $b_1, b_2 : B$. Then*

$$\begin{aligned}(\mathsf{inl}(a_1) = \mathsf{inl}(a_2)) &\simeq (a_1 = a_2)\\ (\mathsf{inl}(b_1) = \mathsf{inl}(b_2)) &\simeq (b_1 = b_2)\\ (\mathsf{inl}(a_1) = \mathsf{inr}(b_1)) &\simeq \mathbf{0}.\end{aligned}$$

A next step would be to characterize paths between dependent functions just as we did for coproduct and dependent pair types. However, the type theory we have developed so far, only allows us to show that if $f, g : \prod_{x:A} B(x)$, where $A : \mathcal{U}$ and $B : A \to \mathcal{U}$, and $f = g$ that then for all $x : A$ we have $f(x) =_{B(x)} g(x)$. We cannot show the other direction at this point, but we will be able to after assuming the univalence axiom in the next section.

2.7 Univalence and functional extensionality

The univalence axiom was added to homotopy type theory by Voevodsky and is based on the observation that the transport function enables us to turn an equality into an equivalence [41, Lem. 2.10.1].

Lemma 2.13. *Let $A, B : \mathcal{U}$ be types. Then $(A =_{\mathcal{U}} B) \to (A \simeq B)$.*

The univalence axiom then states that this also hold in the other direction.

Definition 2.14 (Univalence)**.** *Let $A, B : \mathcal{U}$ be types. Then the univalence axiom states that*

$$(A =_{\mathcal{U}} B) \simeq (A \simeq B).$$

Hence if two types are equivalent, we can transport all properties which hold for one type to hold for the other type and vice versa. We will assume this axiom to hold in all universes we will be working in.

The univalence axiom implies in particular the characterization of paths between dependent functions, we were not able to show yet in Section 2.6. This characterization can also be assumed as a separate axiom, and then it is called the functional extensionality axiom. From a classical point of view this axiom states that two functions are equal if and only if they are equal in each point. From a homotopical point of view, the axiom states that a path between two functions is the same as a homotopy between the two functions. Since we will assume the univalence axiom to hold, the functional extensionality axiom will also be available.

Definition 2.15 (Functional extensionality)**.** *Let $A : \mathcal{U}$ be a type, $B : A \to \mathcal{U}$ a type family, and $f, g : \prod_{x:A} B(x)$ functions. Then the function extensionality axiom states that*

$$(f = g) \simeq \left(\prod_{x:A} f(x) =_{B(x)} g(x)\right).$$

2.8 Propositions as types

In type theory propositions are types and as a result to prove a proposition, we have to construct a term of the proposition. An example of a proposition and type would be $1+1 = 2$, just as $1+1 = 3$. However, we can show that the type $1+1 = 2$ is inhabited (that is: has a proof), whereas if the type $1+1 = 3$ would be inhabited we could conclude that $0 = 1$ which forms a contradiction. The terms of types representing

propositions are also referred to as evidence, witnesses or proofs of the proposition. A proposition can have multiple terms as multiple proofs for a proposition may exist.

When we prove a statement, we prefer to use natural language opposed to a formal construction of a term of the associated type. If we for instance want to prove that if $x = y$ then $y = x$ for all types $A : \mathcal{U}$ and elements $x, y : A$, which is Lemma 2.1, we could state as a proof the following term

$$\lambda A.\lambda x.\lambda y.\lambda p\, \mathsf{ind}_{=_A}((\lambda x.\lambda y.\lambda p.(y = x)), (\lambda x.\, \mathsf{refl}_x), x, y, p) : \prod_{A:\mathcal{U}} \prod_{x,y:A} (x = y) \to (y = x).$$

However, this can hardly be called a readable proof. We prefer instead a proof in the following form.

Proof. Let $A : \mathcal{U}$ an take $x : A$. Then by induction it suffices to prove the statement in the case where y is x and the path we want to define the inverse of is refl_x. Hence we should construct a term of $x = x$, for which we can just choose refl_x. □

Using the types we have constructed in earlier sections we can naturally identify types with logical operations as in Table 1. We need to be careful however when working with 'there exists' and 'or', because here classical and constructive mathematics differ. For example in the case of 'there exists', a constructive proof constructs this element and hence the statement remembers what element is a witness. However, in classical mathematics it might be the case that we know that there exists some element, but that we do not know what element.

Similarly, in the case of 'or', a proof of $A + B$ in type theory will construct either an element of A or B and so we know whether A or B holds. In classical mathematics if we prove $A \vee B$ we do not know which one of the two holds.

Logic	Types
False	$\mathbf{0}$
True	$\mathbf{1}$
$A \vee B$	$\|A + B\|$
$A \wedge B$	$A \times B$
$A \Rightarrow B$	$A \to B$
$\neg A$	$A \to \mathbf{0}$
$\exists(x : A)B(x)$, $x : A$ such that $B(x)$	$\|\sum_{x:A} B(x)\|$
$\forall(x : A)B(x)$	$\prod_{x:A} B(x)$

Table 1: The correspondence between logic and types.

In both examples the type theoretic statement contains more information than the classical statement. The solution for this is a type which only contains the information whether a type is inhabited and not how. Such a type we call a mere proposition.

Definition 2.16. *A type P is a mere proposition if for all $x, y : P$ we have $x = y$. If P is a mere proposition, we write $P : \mathsf{Prop}$.*

We can construct from any type a mere proposition by truncating the type as follows. The second constructor forces $\|A\|$ to be a mere proposition.

Definition 2.17. *If $A : \mathcal{U}$ is a type, then we introduce the type $\|A\|$ using two constructors:*

- *If $a : A$, then $|a| : \|A\|$*
- *If $x, y : \|A\|$, then $x = y$.*

Its recursion principle states that if B is a mere proposition and $f : A \to B$, that there exists a function $g : \|A\| \to B$ such that $g(|a|) :\equiv f(a)$ for all $a : A$.

We will follow the convention as in the HoTT book [41] and use the symbols $\vee$ and $\exists$ to indicate the use of truncation. Our standard will however be to use untruncated logic. If propositional truncation is needed, we will indicate the use of this by the adverb merely.

We introduced truncation to lessen the difference between the constructive logic from type theory and the classical logic. It is useful to note that other classical reasonings also do not hold. Another example is that in general we do not have

$$\prod_{A,B:\mathcal{U}} ((A \times B) \to \mathbf{0})) \to (A \to \mathbf{0}) + (B \to \mathbf{0}).$$

Proofs by contradiction are also allowed in classical mathematics, but in type theory only if the law of excluded middle holds. We shortly discuss this law in Section 2.9 together with the axiom of choice, but refer for more extensive discussions of constructive mathematics to an interesting article by Bauer [6] and the HoTT book [41, Sec. 3.4, 3.10].

2.9 Axiom of choice, law of excluded middle

When introducing the univalence axiom and functional extensionality we mentioned that these must be assumed as an axiom and cannot be proven within type theory. The axiom of choice is often assumed in set theory but we can prove it within type theory in the following form.

$$\left(\prod_{x:X}\sum_{a:A(x)}P(x,a)\right)\to\sum_{a:\left(\prod_{x:X}A(x)\right)}\prod_{x:X}P(x,a(x)).$$

However, the axiom of choice can also be formalized differently, closer to the meaning of the statement in set theory. For this we will need the definition of a set in type theory.

Definition 2.18. *A type A is a set if for all $x, y : A$ and all $p, q : x = y$, we have $p = q$. If A is a set, we will write $A : \mathsf{Set}$.*

Then we can formalize the axiom of choice as follows.

Definition 2.19. *Let $X : \mathsf{Set}$ be a set and let $A : X \to \mathcal{U}$ and $P : \prod_{x:X} A(x) \to \mathcal{U}$ be type families. Then the axiom of choice states that*

$$\left(\prod_{x:X}\left\|\sum_{a:A(x)}P(x,a)\right\|\right)\to\left\|\sum_{g:\prod_{x:X}A(x)}\prod_{x:X}P(x,g(x))\right\|.$$

Although this form of the axiom of choice does not follow from type theory, it can be assumed as an axiom without making the theory inconsistent. The HoTT book however encourages to avoid using the axiom of choice to keep the computational character of type theory [41, p. 10] and hence we will not assume it.

In contrast to the axiom of choice and the situation in classical mathematics, the law of excluded middle cannot be assumed in type theory without causing inconsistencies [41, Cor. 3.2.7].

Lemma 2.20. *It is not the case that for all $A : \mathcal{U}$ we have $A + (\neg A)$.*

However, we can avoid contradictions if we restrict ourselves to mere propositions.

Definition 2.21. *Let $A : \mathsf{Prop}$. Then the law of excluded middle states that $A + (\neg A)$.*

We will however not assume this law for the same reason as we do not assume the axiom of choice. There are however mere propositions for which the law of excluded middle holds and we will call these types decidable. We can extend the notion of a decidable type to decidable families and decidable equalities.

Definition 2.22. *We say that*

- *a type A is decidable if $A + (\neg A)$*
- *a type family $B : A \to \mathcal{U}$, where $A : \mathcal{U}$, is decidable if $\prod_{a:A} B(a) + (\neg B(a))$.*
- *a type A has decidable equality if $\prod_{a,b:A}(a = b) + (\neg(a = b))$.*

Since the law of double negation $\prod_{A:\mathsf{Prop}} \neg\neg A \to A$ is equivalent to the law of excluded middle, we will also not be able to use this result in general.

3 Inductive types

The types we discussed in Section 2, except for the universes and function types, were all examples of inductive types. These are types that are generated by a certain list of generators. In this section we will discuss some generalizations of these inductive types.

However, first we note that for the resulting inductive type to make sense, we have to pose some restrictions on the generators. In the HoTT book [41, Sec. 5.6] there are some examples of invalid definitions. The main restriction is that the type which we try to define, is only allowed to occur strictly positively in the domain of the generator, although some relaxations are possible [41, p. 166]. This means that if we try to define the type A, that an argument of a constructor should not contain A at all, or that A should be the codomain if the argument is an (iterated) function type. Another restriction is that the list of generators of an inductive type should be finite. This is however a milder restriction because if we would for instance have a countable amount of generators, we could parameterize them using $\mathbb{N}$, reducing the list to just one generator.

Not all interesting constructions can be captured by ordinary inductive types. Therefore multiple ways to generalize inductive types have been introduced over the years of which most are discussed, or at least mentioned, in the HoTT book [41, Sec. 5.7, 6]. In this section we will go through some generalizations in order to understand the concept of a higher inductive-inductive definition, which we will eventually use in Section 7. We will give an example of all these generalizations.

The following overview covers by no means all possible generalizations as different generalizations can be combined to make new kinds of definitions. Examples are the inductive-inductive-recursive definitions of Forsberg [23, Sec. 6.1] or the indexed inductive-recursive definitions of Dybjer and Setzer [22].

In general there is not just one kind of definition suitable to define a certain type. As an example, we construct $\mathsf{SortedList}(\mathbb{N})$ in Section 3.4 as an inductive-inductive type, but it can also be constructed as an ordinary inductive type or as an inductive-recursive type [24].

3.1 Inductive families

The first generalization of inductive types we want to discuss is the one of inductive families. Here not just a single inductive type B is defined, but a whole inductive family $B : A \to \mathcal{U}$ simultaneously, based on an already constructed type $A : \mathcal{U}$. The definition of one $B(a)$ for $a : A$ may depend on other $B(a')$, where $a' : A$.

As an example we will define lists of length $n : \mathbb{N}$ with elements of C, denoted $\mathsf{Vec}_n(C)$, as in the HoTT book [41, p. 168]. The inductive family $\mathsf{Vec}(C) : \mathbb{N} \to \mathcal{U}$ is generated by the constructors:

- $\mathsf{nil} : \mathsf{Vec}_0(C)$, which is the list of length zero
- $\mathsf{cons} : \prod_{n:\mathbb{N}} C \to \mathsf{Vec}_n(C) \to \mathsf{Vec}_{\mathsf{succ}(n)}(C)$, which concatenates a term of C to a list of length $n : \mathbb{N}$ with as result a list of length $\mathsf{succ}(n)$.

By defining the $\mathsf{Vec}_n(C)$ for $n : \mathbb{N}$ like this, the separate types are not an inductive types by themselves, because they all depend on lists of shorter lengths.

We could also see the above definition as a definition of the type family $\mathsf{Vec} : \mathcal{U} \to \mathbb{N} \to \mathcal{U}$ since the above definition depends on the type C. This is similar to the definition of the coproduct $+$ where we do not define $C_1 + C_2$ for just two fixed types $C_1 : \mathcal{U}$ and $C_2 : \mathcal{U}$, but define $+$ for arbitrary types C_1 and C_2. In cases like these, we refer to C (or C_1 and C_2) as a parameter.

3.2 Mutual inductive definitions

A mutual inductive definition can be seen as a special case of an inductive family where the indexing family is finite. We discuss an example of this, again from the HoTT book [41, p. 169-170], where the types even and odd of respectively even and odd natural numbers are defined. Firstly, even is generated by the constructors:

- $0 : \mathsf{even}$
- $\mathsf{esucc} : \mathsf{odd} \to \mathsf{even}$

and secondly, odd is generated by the constructor:

- $\mathsf{osucc} : \mathsf{even} \to \mathsf{odd}$.

From these definitions it is not immediately clear that even and odd are part of an inductive family. However, we can introduce the family $\mathsf{paritynat} : \mathbf{2} \to \mathcal{U}$ and let $\mathsf{paritynat}(0_{\mathbf{2}})$ represent the even numbers and $\mathsf{paritynat}(1_{\mathbf{2}})$ the odd numbers. Then we can define $\mathsf{paritynat}(0_{\mathbf{2}})$ to be generated by:

- $0 : \mathsf{paritynat}(0_{\mathbf{2}})$
- $\mathsf{esucc} : \mathsf{paritynat}(1_{\mathbf{2}}) \to \mathsf{paritynat}(0_{\mathbf{2}})$

and $\mathsf{paritynat}(1_{\mathbf{2}})$ by

- $\mathsf{osucc} : \mathsf{paritynat}(0_{\mathbf{2}}) \to \mathsf{paritynat}(1_{\mathbf{2}})$.

Formulated like this, we now see clearly that even and odd can be constructed as an inductive family. A similar reduction to inductive families can be applied for all mutual inductive definitions.

3.3 Inductive-recursive definitions

Whereas we only defined one type or type family in the generalizations discussed so far, we can use an inductive-recursive definition to define both a type and type family simultaneously. In this kind of definition, the type A is defined inductively and the type family $B : A \to \mathcal{U}$ is defined recursively. As they are defined simultaneously, the constructor of A may refer to $B(a)$'s for $a : A$. Since B is defined recursively, it is defined using the recursion principle of A and so immediately when a constructor of A is stated, the value of B on that constructor should also be given.

An example of this construction is the set of lists with distinct elements of a type C, denoted $\mathsf{DistinctList}$, together with the recursive relation $\mathsf{Fresh} : \mathsf{DistinctList} \to C \to \mathsf{Set}$, due to Dybjer and Setzer [21]. Note that this is already a slight generalization compared to the standard case where B depends only on A, since Fresh depends on both $\mathsf{DistinctList}$ and C. Let $\# : C \to C \to \mathsf{Set}$ be a (predefined) relation, where $a\#b$ means that a and b are distinct elements of C. Then $\mathsf{DistinctList}$ is generated by the constructors:

- $\mathsf{nil} : \mathsf{DistinctList}$
- $\mathsf{cons} : \prod_{b:C} \prod_{u:\mathsf{DistinctList}} \mathsf{Fresh}(u, b) \to \mathsf{DistinctList}$, which creates a new list by adding $b : C$ to a list $u : \mathsf{DistinctList}$ if b is 'fresh' to u

and we can define Fresh by

- $\mathsf{Fresh}(\mathsf{nil}, c) = \mathbf{1}$, which states that an element $c : C$ is trivially different from all elements of the empty list
- $\mathsf{Fresh}(\mathsf{cons}(b, u, b'), c) = b\#c \times \mathsf{Fresh}(u, c)$, which states that if $c : C$ is different from $b : C$ and c is different from all elements in $u : \mathsf{DistinctList}$, that c is different from all elements in the list $\mathsf{cons}(b, u, b')$.

3.4 Inductive-inductive definitions

In an inductive-inductive definition, again both a type A and a type family $B : A \to \mathcal{U}$ are simultaneously defined, but now the type family is inductively defined. In comparison to two separate definitions of a type and type family, here a constructor of A may additionally depend on $B(a)$'s for $a : A$, and a constructor of $B(a)$ for some $a : A$ may additionally depend on A. We will compare this definition to an inductive-recursive definition in the next section.

An example of an inductive-inductive definition are sorted lists, due to Forsberg [24]. Here the type of sorted lists with elements of a type C, denoted $\mathsf{SortedList}(C) : \mathsf{Set}$, is inductively defined together with the relation $\leq_L$: $C \times \mathsf{SortedList}(C) \to \mathsf{Set}$. We want $n \leq_L l$ to be true for $n : C$ and $l : \mathsf{SortedList}$ if n is smaller or equal to every element in the list l. Therefore we let $\mathsf{SortedList}(C)$ be generated by the constructors:

- $\mathsf{nil} : \mathsf{SortedList}(C)$, which is the trivially sorted list of length zero
- $\mathsf{cons} : \prod_{n:C} \prod_{l:\mathsf{SortedList}(C)} n \leq_L l \to \mathsf{SortedList}$, which constructs a new sorted list by putting n in front of l if n is smaller or equal to every element in l.

The relation $\leq_L$ is generated by the constructors:

- $\mathsf{triv} : \prod_{m:C} m \leq_L \mathsf{nil}$, which states that every $m : C$ is trivially smaller than every element of the empty list
- $\ll \cdot \gg : \prod_{m,n:C} \prod_{l:\mathsf{SortedList}(C)} \prod_{p:n\leq_L l} (m \leq n) \to (m \leq_L l) \to (m \leq_L \mathsf{cons}(n,l,p))$, which states that if $m \leq n$, $m \leq_L l$ and $n \leq_L l$ then $m \leq_L \mathsf{cons}(n,l,p)$.

Note that if the order $\leq$ on C is transitive, then the second constructor of $\leq_L$ does not need a proof $m \leq_L l$.

3.5 Comparison

It is useful here to make a comparison between the definitions we have seen so far to make the distinction between inductive-inductive definitions and the other generalizations clear.

First of all, an inductive-inductive definition is not a mutual inductive definition, because the type family $B : A \to \mathcal{U}$ is indexed by A. Moreover, an inductive-inductive definition is also not a definition of an inductive family as the index A may refer to B.

An inductive-inductive definition and an inductive-recursive definition both define a type A and type family $B : A \to \mathcal{U}$ simultaneously and the difference is only in the way that B is defined: if we define B inductively we have to state for all $a : A$ constructors for $B(a)$, whereas if we define B recursively, we should state the value of B on constructors of A.

Both the inductive and recursive definitions of B have their advantages [23, p. 7-8]. On the one hand an inductive-inductive definition gives more freedom to define B as multiple constructors can add to the definition of B on a constructor of A. However, since B is inductively defined in an inductive-inductive definition, the family B is only allowed to occur positively in the constructors of both A and B. If B on the other hand is defined recursively, then B can also occur negatively in a constructor of A and so the inductive-recursive definition gives more freedom. Lastly, from a practical perspective it is possible to define $B : A \to C$ recursively for any codomain C. However defining $B : A \to \mathbf{2}$ inductively does not make sense. If we want to map $B(a)$ for example to $0_{\mathbf{2}} : \mathbf{2}$ for some $a : A$ then we would have to inductively define $0_{\mathbf{2}}$ and this is not possible as $0_{\mathbf{2}}$ has no elements. Therefore we can only inductively define functions with codomains which have types as terms, such as $\mathcal{U}$, Set and Prop.

We end this comparison with the note that this discussion is specific to homotopy type theory, since in other variants of type theory the differences between the previous generalizations may not exist [23, Th. 5.39].

3.6 Higher inductive types (HIT)

A higher inductive type generalizes an ordinary inductive type in a different way than the previous examples we have seen. In the case of a higher inductive type not only constructors for the elements of a type are given, but also constructors for its (iterated) equality types. We will refer with point constructors to constructors defining terms of the type and with path constructors to generators of equality types.

Just as for ordinary types, the type being defined may only occur strictly positively in the input of the constructors of that same type. In particular this excludes the use of the identity type of the type being defined in the constructor [41, Sec. 6.13]. Hence if we define a type A, we cannot use a constructor of the form $(x = y) \to A$ for some $x, y : A$. A path constructor is allowed to have other constructors of the type as end points, but these constructors should then be listed before this path constructor. The end points of the path constructed by a path constructor are also bounded by some requirements as not every possible option leads to a sensible definition. However, the exact requirements are still part of current research [41, Sec. 6.13]. So far there is no good, consistent scheme general enough to cover all interesting cases [33].

As an example we give here the example of the circle S^1, which is generated by

- a point $\mathsf{base} : S^1$
- a path $\mathsf{loop} : \mathsf{base} =_{S^1} \mathsf{base}$.

3.7 Higher inductive-inductive types (HIIT)

A higher inductive-inductive type is a combination of an inductive-inductive type and a higher inductive type. Hence a type and type family are simultaneously and inductively defined, whereby both point and path constructors are allowed.

A first formalization of higher inductive-inductive types has been given by Kaposi and Kovács [28], but no general scheme has yet been proposed. More is published on the case where the HIITs are set-truncated to end up with quotient inductive-inductive types (QIITs) [2, 19, 29]. There are two main reasons for doing so. The first is that it makes the analysis and formalization easier as no higher paths have to be taken into account. A second reason is that all known and interesting higher inductive-inductive types such as the Cauchy reals are sets [2]. A disadvantage is of course that not all ordinary higher inductive types are included, of which the circle is an example.

Just as for the ordinary inductive type and other generalizations, the QIIT constructed should occur strictly positive in its constructors [19]. Although it might be the case, just as for HITs, that other occurrences can also be allowed for a consistent theory [41, Section 6.13].

We will not give an example of a HIIT (or QIIT) here, but refer to the construction of the Cauchy reals in the HoTT book [41, Section 11.3] or to Section 7 where we will work with a HIIT ourselves.

4 Ordered field

Now that we have covered the aspects of homotopy type theory we will need, we can start with our goal to construct a Cauchy complete Archimedean ordered field. In this section we state the requirements for a type to be an ordered field. We do not state the definitions of being Archimedean and Cauchy complete here, because these two definitions are not really different from their definitions in classical mathematics. Hence we just state their definitions when we work with them in respectively Section 5.7 and Section 6.3.

The HoTT book [41, Def. 11.2.7] defines an ordered field in a constructive setting. This formulation of an ordered field differs from the definition in the HoTT library in Coq, but the definitions are equivalent[1] We will work with the definition as stated in the HoTT library, since eventually we also want to formalize that the Eudoxus reals form an Archimedean ordered field using this library.

In the definition of an ordered field we distinguish four main components. To avoid an immense definition we will first state the definition of an ordered field without further specifying these components.

Definition 4.1. *An ordered field is a set F together with constants 0, 1, operations $+$, $-$, $\cdot$, $/$, $\min$, $\max$ and mere relations $\leq$, $<$, $\#$ such that*

- $(F, 0, 1, +, -, \cdot, /, \#)$ *is a field*
- $(F, \min, \max, \leq)$ *is a lattice*
- $\#$ *is a tight apartness relation*
- $(F, 0, 1, +, \cdot, \leq, <, \#)$ *is a full pseudo semiring order*

In Section 5 we will define the type of Eudoxus reals and prove that this is an Archimedean ordered field. We show in Section 5.8 that we can define a tight apartness relation on the Eudoxus reals and we show that the Eudoxus reals form a lattice in Section 5.10. That the Eudoxus reals are a field and can be endowed with a full pseudo semiring order will however take multiple sections. Hence we will state here the definitions of a field and full pseudo semiring order, but will postpone the definitions of an apartness relation and lattice to respectively Section 5.8 and 5.10.

First of all, the constructive definition of a field is similar to its classical definition. The main difference is however the occurrence of the apartness relation. This is a stronger notion than inequality and is in general needed in constructive mathematics to define a multiplicative inverse. We slightly simplified the definition of a field here to only list the right-identity for addition and multiplications and right-inverse of addition, since both operations are required to be commutative.

Definition 4.2. *A field is a set F together with constants 0, 1, operations $+$, $-$, $\cdot$, $/$ and mere relation $\#$ such that*

1. *$(F, 0, 1, +, -, \cdot)$ is a commutative ring, that is for all $a, b, c : F$ we have*
   - *(associativity)* $a + (b + c) = (a + b) + c$
   - *(commutativity)* $a + b = b + a$
   - *(additive identity)* $a + 0 = a$
   - *(additive inverse)* $a + (-a) = 0$
   - *(associativity)* $a \cdot (b \cdot c) = (a \cdot b) \cdot c$
   - *(commutativity)* $a \cdot b = b \cdot a$
   - *(multiplicative identity)* $a \cdot 1 = a$
   - *(distributivity)* $a \cdot (b + c) = a \cdot b + a \cdot c$
2. *if $a \# 0$ then $1/a$ is defined and $a/a = 1$ for all $a : F$*
3. *$0 \# 1$*
4. *(strong binary extensionality) $a \cdot b \# c \cdot d$ implies merely $a \# c$ or $b \# d$ for all $a, b, c, d : F$*
5. *(strong binary extensionality) $a + b \# c + d$ implies merely $a \# c$ or $b \# d$ for all $a, b, c, d : F$*

A full pseudo semiring order was first introduced by Krebbers and Spitters [30] and does not have an obvious classical alternative. The requirements are however not very different than we are used to for real numbers. Exceptions are however again the two statements related to apartness, of which the strong binary extensionality of the multiplication was already a requirement for a field. Another difference is the appearance of cotransitivity which is a constructive alternative to trichotomy, which would normally state that either $a < b$ or $a = b$ or $b < a$ for all $a, b : F$.

[1] The definition of an ordered field can be found in `Classes/interfaces/orders` as `IsOrderedField`, where it is also stated that the definition coincides with the definition from the HoTT book.

Definition 4.3. *A full pseudo semiring order is a set F together with constants 0, 1, operations $+$, $\cdot$ and mere relations $\leq$, $<$, $\#$ such that for all $a, b, c : F$ it holds that*

1. *$a\#b$ if and only if merely $a < b$ or $b < a$*
2. *(cotransitivity) $a < b$ implies merely $a < c$ or $c < b$*
3. *$\neg(a < b \wedge b < a)$*
4. *$0 < a$ and $0 < a$ imply $0 < a \cdot b$*
5. *(strong binary extensionality) $a \cdot b \# c \cdot d$ implies merely $a\#c$ or $b\#d$*
6. *$a < b$ if and only if $c + a < c + b$*
7. *$\neg(b < a)$ implies the existence of $c : F$ such that $b = a + c$*
8. *$a \leq b$ if and only if $\neg(b < a)$*

We would like to note that requirement 7 may seem trivial, but that this requirement is explained by the observation that a full pseudo semiring order does not require an additive inverse. The requirement is for instance useful when implementing the natural numbers [30].

Since the requirements for a field and full pseudo semiring order will be proven throughout Section 5, we list in Table 2 where these proofs can be found.

Requirement	Proven in/on
Field 1	Section 5.3
Field 2	Lemma 5.53
Field 3	Page 38 (end of Section 5.8)
Field 4	Corollary 5.65
Field 5	Page 38 (end of Section 5.8)
FPSRO 1	Lemma 5.52
FPSRO 2	Lemma 5.50
FPSRO 3	Lemma 5.24
FPSRO 4	Lemma 5.31
FPSRO 5	Corollary 5.65
FPSRO 6	Page 30
FPSRO 7	Page 32
FPSRO 8	Lemma 5.29 and Lemma 5.51

Table 2: The location of the proofs of the requirements for a field and full pseudo semiring order (FPSRO).

5 The Eudoxus reals

In this section we will construct the Eudoxus reals as an Archimedean ordered field based on the construction of A'Campo [1]. In Section 6 we check whether it is Cauchy complete. Not all details of A'Campo's construction can be found in his article, but they are often present in the master's thesis of Mizrahi [35].

The construction of A'Campo differs from other constructions of the real numbers since he constructs the real numbers directly from the integers, instead of from the rational numbers. Firstly, A'Campo defines slopes as maps $\mathbb{Z} \to \mathbb{Z}$ which differ only finitely from linear maps. Then he defines the Eudoxus reals as slopes module some equivalence relation. He shows that the resulting quotient set is indeed a construction of the real numbers by proving that the quotient is a complete totally ordered field. Odd slopes and well-adjusted slopes play an important part in this. The former are odd functions and the latter are slopes which are close to being linear.

Our construction of the Eudoxus reals will follow this approach of A'Campo, but there are differences. First of all, since we are working in homotopy type theory we have to adjust some proofs to make them constructive. The proof that the Eudoxus reals are Archimedean is in particular completely different. We also have to define an absolute value, an apartness relation, and a meet and join, which can be found in respectively Section 5.5, 5.8 and 5.10. Lastly, the requirements for the ordering differ between a constructive full pseudo semiring order and a classical total order on a field, but often our requirements follow quite quickly. Two exceptions to this are the cotransitivity in Section 5.8 and the strong binary extensionality of the multiplication we prove in Section 5.11.

Our goal was to formalize this section in Coq, but we were not able to formalize all results yet. Hence in this section we indicate whether we have formalized the result or not using respectively (✓) and (✗). We use (✓) if the reasoning of the proof itself is formalized, even when the proof refers to non-formalized results. Sometimes, a result consists of two statements or directions. In cases where only one case is formalized we indicate this by the use of /, so (✓/✗) indicates for instance that the first statement (or left-to-right direction) is formalized but the second statement (or right-to-left direction) not. We omit these indications for standard definitions such as the definition of surjectivity, since the library we use in Coq already includes these definitions.

5.1 Slopes and the Eudoxus reals

To define slopes and the Eudoxus reals, we start from the type of integers $\mathbb{Z}$ equipped with the addition operation $+$. A'Campo defines a map $\lambda : \mathbb{Z} \to \mathbb{Z}$ as a slope if the set $\{\lambda(m+n) - \lambda(m) - \lambda(n) | m, n \in \mathbb{Z}\}$ is finite. To be able to work with the notion of finiteness in our constructive setting, we will work with the existence of some bound on the set.

Definition 5.1 (✓). *A map $\lambda : \mathbb{Z} \to \mathbb{Z}$ is a slope if there exists $d_\lambda : \mathbb{Z}$ such that for all $m, n : \mathbb{Z}$ we have $-d_\lambda \leq \lambda(m+n) - \lambda(m) - \lambda(n) \leq d_\lambda$. We define the type of slopes, denoted $\mathcal{S}$, as*

$$\mathcal{S} :\equiv \sum_{\lambda:\mathbb{Z}\to\mathbb{Z}} \sum_{d_\lambda:\mathbb{Z}} \prod_{m,n:\mathbb{Z}} -d_\lambda \leq \lambda(m+n) - \lambda(m) - \lambda(n) \leq d_\lambda.$$

Note that since inequalities of integers are mere propositions, the type of slopes is a set [41, Ex. 3.1.5].

Given a function $\lambda : \mathbb{Z} \to \mathbb{Z}$ it is not correct to say that this function is a slope. An element of the type of slopes namely consists of a function $\mathbb{Z} \to \mathbb{Z}$ together with a bound in $\mathbb{Z}$ and proof that this bound is indeed a bound. Hence a function λ can be associated with multiple distinct slopes. We will however refer to a function $\lambda : \mathbb{Z} \to \mathbb{Z}$ as a slope in which case we implicitly assume the existence of a bound d_λ and a proof that this is indeed a bound. This also entails that we will write $\lambda : \mathcal{S}$ when actually $(\lambda, d_\lambda, p_\lambda) : \mathcal{S}$ would be the correct statement. This is similar to the convention of stating that an integer $n : \mathbb{N}$ is an element of $\mathbb{N}_+$, because if p is a proof that $0 < n$ then actually $(n, p) : \mathbb{N}_+$.

We can see $\mathbb{Z}$ as a subtype of the type of slopes by identifying $a : \mathbb{Z}$ with the linear map $\lambda : \mathbb{Z} \to \mathbb{Z}$ defined by $\lambda(n) = an$ for all $n : \mathbb{Z}$. We can also define the subtype $\mathcal{Q} \subset \mathcal{S}$, which is similar to the rational numbers.

Definition 5.2 (✓). *We define the subtype $\mathcal{Q} \subset \mathcal{S}$ as the type of slopes $\mu : \mathcal{S}$ for which there exist $p : \mathbb{Z}$ and $q : \mathbb{N}_+$ such that either $p \geq 0$ and $\mu(n) = \min\{k : \mathbb{N} | qk \geq pn\}$ for $n > 0$, or $p < 0$ and $\mu(n) = -\min\{k : \mathbb{N} | qk \geq -pn\}$ for $n > 0$. Additionally we require that $\mu(-n) = -\mu(n)$ for all $n : \mathbb{Z}$. We will refer to such slopes as rational slopes.*

First of all we note that this minimum exists for all $p : \mathbb{Z}$ and $q : \mathbb{N}_+$. In order to see this, take $n : \mathbb{N}_+$ and assume without loss of generality that $0 < p$. Then we know that $q \cdot (pn) \geq pn$ and therefore this

minimum lies in the interval $[0, pn]$. Hence to determine the minimum we can just check all elements in this interval starting from 0 to determine the first element k for which $qk \geq pn$ holds.

Even though we will call these slopes rational, this subtype is not a construction of the rational numbers, since it is not closed under operations as addition. Our definition of the rationals in Definition 5.5 will however heavily rely on this subtype. Since we can identify however a rational slope r with a pair $(p, q) : \mathbb{Z} \times \mathbb{N}_+$, we can define for instance $1/r$ as the slope $(q, p) : \mathcal{Q}$. We will formalize this in Section 5.12. Similar to $\mathbb{N}_+$, we define $\mathcal{Q}_+$ as the type $\sum_{(p,q):\mathcal{Q}} p > 0$. Also for $\mathcal{Q}_+$, we will be sloppy in the notation and write $p : \mathcal{Q}_+$ to denote $(p, s) : \mathcal{Q}_+$ for some $s : p > 0$.

Two slopes are said to be equivalent if their function values differ only finitely, where we again will work with the existence of a bound on the set. That is,

Definition 5.3 (✓). *Two slopes $\lambda, \kappa : \mathcal{S}$ are equivalent, denoted $\lambda \sim \kappa$, if there merely exists $S : \mathbb{Z}$ such that for all $n : \mathbb{Z}$ we have $-S \leq \lambda(n) - \kappa(n) \leq S$.*

That that this relation is indeed an equivalence relation is proven by Mizrahi [35]. The construction of A'Campo is then based on the quotient set of slopes and this equivalence relation. We can achieve a similar structure in type theory by defining the Eudoxus rules as a (set-)quotient type. This type requires the equivalence relation to be a mere relation, which is the reason for the truncation in Definition 5.3. More on this type can be found in the HoTT book [41, Section 6.10].

Definition 5.4. *The quotient type of $\mathcal{S}$ by $\sim$, denoted $\mathcal{E} : \mathsf{Set}$ (or $\mathcal{S}/\sim : \mathsf{Set}$), is generated by:*

- $\mathsf{classof} : \mathcal{S} \to \mathcal{E}$,
- $\mathsf{classeq} : \prod_{\lambda,\mu:\mathcal{S}} (\lambda \sim \mu) \to (\mathsf{classof}\ \lambda = \mathsf{classof}\ \mu)$,
- $\mathsf{setTrunc} : \prod_{a,b:\mathcal{E}} \prod_{r,s:a=b} r = s$, *which is the 0-truncation constructor.*

When we compare this definition with the quotient set in set theory we see that the map $\mathsf{classof}$ corresponds to the quotient function and $\mathsf{classeq}$ states that two equivalent elements are in the same equivalence class.

To define for instance the operations of addition and multiplication we will need the recursion and induction principle for the Eudoxus reals. The recursion principle states that given a type $B : \mathsf{Set}$ together with

- a function $g : \mathcal{S} \to B$,
- and for each $\lambda, \mu : \mathcal{S}$ such that $\lambda \sim \mu$ a path $p : g(\lambda) = g(\mu)$

we have a function $f : \mathcal{E} \to B$ such that for all $\lambda, \mu : \mathcal{S}$ we have $f(\mathsf{classof}\ \lambda) \equiv g(\lambda)$, and if $H : \lambda \sim \mu$ and $p : g(\lambda) = g(\mu)$ then $f(\mathsf{classeq}\ H) = p$.

The induction principle states that given a function $B : \mathcal{E} \to \mathsf{Set}$ together with

- a function $g : \prod_{\lambda:\mathcal{S}} B(\mathsf{classof}\ \lambda)$
- and for all $\lambda, \mu : \mathcal{S}$ and $H : \lambda \sim \mu$, a path $p : \mathsf{transport}^{\lambda \mapsto B(\mathsf{classof}\ \lambda)}(\mathsf{classeq}\ H, g(\lambda)) = g(\mu)$

we have a function $f : \prod_{a:\mathcal{E}} B(a)$ such that for all $\lambda, \mu : \mathcal{S}$ we have $f(\mathsf{classof}\ \lambda) = g(\lambda)$ and if $H : \lambda \sim \mu$ and $p : \mathsf{transport}^B(\mathsf{classeq}\ H, g(\lambda)) = g(\mu)$, then $\mathsf{apd}_f(\mathsf{classeq}\ H) = p$.

When defining the type of rational slopes we stated it was not a construction of the rational numbers. Instead we will define the type of rational numbers as subset of the Eudoxus reals as follows. In this definition of $\mathbb{Q}$ we truncated the existence of a representative rational slope, since for $\mathbb{Q}$ to be a field it should in particular be a set. The truncation enforces this. We will discuss this type in more detail in Section 5.12.

Definition 5.5 (✗). *Define the subset $\mathbb{Q}$ of the Eudoxus reals as*

$$\mathbb{Q} :\equiv \sum_{a:\mathcal{E}} \left\| \sum_{\lambda:\mathcal{Q}} \mathsf{classof}\ \lambda = a \right\|.$$

5.2 Concentration lemma

Slopes can differ, albeit only finitely, still quite a lot from linear maps. Hence it is often useful to work with odd slopes or, the even more regular, well-adjusted slopes.

Definition 5.6 (✓). *A slope $\lambda : \mathcal{S}$ is odd if $\lambda(-n) = -\lambda(n)$ for all $n \in \mathbb{Z}$.*

Definition 5.7 (✓). *A slope $\lambda : \mathcal{S}$ is well-adjusted if it is odd and it satisfies $-1 \leq \lambda(m+n) - \lambda(m) - \lambda(n) \leq 1$ for all $m, n : \mathbb{Z}$.*

We note that we do not require $d_\lambda = 1$ for a slope $\lambda : \mathcal{S}$ to be well-adjusted, which is a stronger requirement. As examples of well-adjusted slopes, we mention the slopes representing the integers and the rational slopes in $\mathcal{Q}$.

We want to prove that every slope is equivalent to such a well-adjusted slope. The first step is to show that every slope is equivalent to an odd slope [35, Pro. 4.26].

Lemma 5.8 (✓). *Let $\lambda : \mathcal{S}$ and define $\kappa : \mathcal{S}$ by*

$$\kappa(n) = \begin{cases} \lambda(n), & \text{if } n > 0 \\ 0, & \text{if } n = 0 \\ -\lambda(-n), & \text{if } n < 0. \end{cases}$$

Then $\lambda \sim \kappa$.

From this we can conclude that for two slopes to be equivalent, we do not need to bound their pointwise difference on the whole of $\mathbb{Z}$, but only on $\mathbb{N}$, since the equivalence relation is in particular transitive. To define a similar result for well-adjusted slopes, we need the concept of the optimal Euclidean divisor.

Definition 5.9 (✗). *Let $p, q : \mathbb{Z}$ such that $q \# 0$. The optimal Euclidean divisor is the integer $r : \mathbb{Z}$ that satisfies the inequalities $2p - |q| \leq 2qr < 2p + |q|$. The result of optimal Euclidean division of p by q is denoted as $p : q$.*

The construction of the optimal Euclidean divisor relies heavily on Euclidean division [35, Pro. 4.36]. Since the optimal Euclidean divisor of $p \in \mathbb{Z}$ and $q \in \mathbb{Z} \setminus \{0\}$ satisfies $|p/q - p : q| \leq 1/2$, we see that it corresponds to a rounded division.

Then the concentration lemma states that every slope is equivalent to a well-adjusted slope and hence every Eudoxus real can merely be represented by a well-adjusted slope.

Lemma 5.10 (✓, Concentration Lemma). *Let $\lambda : \mathcal{S}$ and let $d : \mathbb{N}_+$ be such that*

$$-d \leq \lambda(m+n) - \lambda(m) - \lambda(n) \leq d.$$

Define the map $\kappa : \mathbb{Z} \to \mathbb{Z}$ by $\kappa(n) = \lambda(3nd) : 3d$ for all $n \in \mathbb{Z}$. Then κ is a well-adjusted slope and equivalent to λ.

The Concentration Lemma is proven by A'Campo [1, Lemma 4] and by Mizrahi in greater detail [35, Lemma 4.44].

5.3 Commutative ring

Now we want to define operations like addition, multiplication and negation on the Eudoxus reals to establish the Eudoxus reals as a commutative ring. Before we state the definitions, we would first like to point out the difference in defining a function on a quotient *type* and a quotient *set*.

Both in set theory and in homotopy type theory we can show that the quotient function classof is surjective. We refer for a proof of the latter to the HoTT book [41, Lemma 6.10.2]. However, it is important to note that in homotopy type theory the definition of surjective uses a truncation.

Definition 5.11. *Let $A, B : \mathcal{U}$ and let $f : A \to B$. Then f is surjective if for every $b : B$ there merely exists $a : A$ such that $f(a) = b$.*

In set theory, assuming we have defined a negation for slopes, we would define the negation of an Eudoxus real a as $-a = [-\lambda]$ if $a = [\lambda]$. Here the existence of $\lambda : \mathcal{S}$ such that $[\lambda] = a$ uses the surjectivity of the quotient function. Then we would have to check that this definition does not depend on our choice of representative of a. We cannot follows this procedure exactly with a quotient type, but the induction

principle does capture this idea. For instance, to define the negation of an Eudoxus real we can use the recursion principle. Hence we have to define a function $\mathcal{S} \to \mathcal{E}$, for which we choose $\lambda \mapsto \mathsf{classof}(-\lambda)$. Then we have to check that if $\lambda \sim \mu$ that $\mathsf{classof}(-\lambda) = \mathsf{classof}(-\mu)$. By applying $\mathsf{classeq}$, this reduces to showing that if $\lambda \sim \mu$ then $-\lambda \sim -\mu$, which coincides with the check in set theory that the definition is independent of the choice of representative.

Now to define an operation on slopes, we need to state the resulting function $\mathbb{Z} \to \mathbb{Z}$, a bound and proof that the function is a slope with the given bound. Stating the proof in a definition will make the definitions quite long, even though it is more important that there is a proof than what the exact formulation of the proof is. Hence we will state the proof in the subsequent proof section following a definition, and only state the function and bound in the actual definition.

Keeping this in mind, we define the operations of addition and multiplication as follows.

Definition 5.12 (✓). *Let $\lambda, \mu : \mathcal{S}$ be slopes. Then we define the sum $\lambda+\mu : \mathcal{S}$ by $(\lambda+\mu)(n) = \lambda(n)+\mu(n)$ for all $n : \mathbb{Z}$ with bound $d_{\lambda+\mu} :\equiv d_\lambda + d_\mu$. By the recursion principle, this induces an addition on $\mathcal{E}$ such that if $a, b : \mathcal{E}$ then $a + b : \mathcal{E}$.*

Proof. Mizrahi [35, Pro. 4.7] proves that the pointwise addition of two slopes results in another slope with the stated bound. To apply the recursion principle it suffices by $\mathsf{classeq}$ to prove that for all $\lambda_1, \lambda_2, \mu_1, \mu_2 : \mathcal{S}$ if $\lambda_1 \sim \lambda_2$ and $\mu_1 \sim \mu_2$ that $\lambda_1 + \mu_1 \sim \lambda_2 + \mu_2$. This is also proven by Mizrahi [35, Pro. 4.8]. □

Definition 5.13 (✓). *Let $\lambda, \mu : \mathcal{S}$ be slopes. Then we define the product $\lambda \cdot \mu : \mathcal{S}$ as the composition $\lambda \circ \mu : \mathbb{Z} \to \mathbb{Z}$ with bound $d_{\lambda\cdot\mu} :\equiv 2d_\lambda + \max\{\lambda(n) | n : \mathbb{Z}, -d_\mu \leq n \leq d_\mu\}$. By the recursion principle, this induces a multiplication on $\mathcal{E}$ such that if $a, b : \mathcal{E}$ then $a \cdot b : \mathcal{E}$.*

Proof. Mizrahi [35, Pro. 4.19] proves that the composition of two slopes results in another slope with the stated bound. To apply the recursion principle it suffices by $\mathsf{classeq}$ to prove that for all $\lambda_1, \lambda_2, \mu_1, \mu_2 : \mathcal{S}$ if $\lambda_1 \sim \lambda_2$ and $\mu_1 \sim \mu_2$ that $\lambda_1 \cdot \mu_1 \sim \lambda_2 \cdot \mu_2$. This is also proven by Mizrahi [35, Pro. 4.20]. □

Before we define the identities for addition and multiplication, we note that two slopes are equal, if and only if the associated functions and slope bounds are equal. Intuitively this is clear, but we can also prove this using the characterization of paths in dependent pairs.

Lemma 5.14 (✓). *Let $(\lambda, d_\lambda, p_\lambda), (\mu, d_\mu, p_\mu) : \mathcal{S}$ be two slopes. Then $(\lambda, d_\lambda, p_\lambda) =_{\mathcal{S}} (\mu, d_\mu, p_\mu)$ if and only if $\lambda =_{\mathbb{Z}\to\mathbb{Z}} \mu$ and $d_\lambda =_{\mathbb{Z}} d_\mu$.*

Proof. Take $(\lambda, d_\lambda, p_\lambda), (\mu, d_\mu, p_\mu) : \mathcal{S}$ and assume $s : \lambda = \mu$ and $d_\lambda = d_\mu$. Then to prove the equality between the slopes, it remains by Lemma 2.11 to show that $s_*(d_\lambda, p_\lambda) = (d_\mu, p_\mu)$. The second elements of these two pairs are equal since they are both terms of a mere proposition. The first element of the pair $s_*(d_\lambda, p_\lambda)$ is $\mathsf{transport}^{\lambda\mapsto\mathbb{Z}}(s, d_\lambda)$ [41, Th. 2.7.4]. However, since this transportation occurs in the constant type family which send every element of $(\mathbb{Z} \to \mathbb{Z})$ to $\mathbb{Z}$, also the transportation is constant and so $s_*(d_\lambda) = d_\lambda$[41, Lem 2.3.5]. Hence $s_*(d_\lambda) = d_\mu$, which implies that $s_*(d_\lambda, p_\lambda) = (d_\mu, p_\mu)$.

The other direction follows the same line of reasoning. □

Now we can use this lemma to easily conclude that we can define an additive identity on slopes and hence on the Eudoxus reals. However, in the case of a multiplicative identity we only have the right-identity up to equivalence. Hence we decided to formulate the multiplicative identity for slopes completely up to equivalence, so that for both left and right multiplication with the identity we have an equivalence and not in the one case an equality and in the other case an equivalence.

Lemma 5.15 (✓). *The additive identity for slopes is the zero slope $0 : \mathcal{S}$ given by $0(n) = 0$ for all $n : \mathbb{Z}$ with bound $d_0 :\equiv 0$. The additive identity for the Eudoxus reals is $\mathsf{classof}\, 0 : \mathcal{E}$.*

Proof. That the zero slope is the additive identity for slopes follows from Lemma 5.14.

For the Eudoxus reals we can prove for all $a : \mathcal{E}$ that $a + \mathsf{classof}\, 0 = a$ using the induction principle. This reduce the statement to the corresponding statement for slopes. Hence we have to show that for all $\lambda : \mathcal{S}$ that $\lambda + 0 = \lambda$, which we just concluded. Since the Eudoxus reals are a set, the statement '$a + \mathsf{classof}\, 0 = a$ for all $a : \mathcal{E}$' is a mere proposition. Therefore the path constructors are respected when applying induction. Similarly, we prove that the zero slope is the left identity. □

Lemma 5.16 (✓). *Define the identity slope $1 : \mathbb{Z} \to \mathbb{Z}$ as $1(n) = n$ for all $n : \mathbb{Z}$ with bound $d_1 :\equiv 0$. Then for all $\lambda : \mathcal{S}$ we have $1 \cdot \lambda \sim \lambda \sim \lambda \cdot 1$. The multiplicative identity for the Eudoxus reals is $\mathsf{classof}\, 1 : \mathcal{E}$.*

Proof. That we have $1 \cdot \lambda \sim \lambda \sim \lambda \cdot 1$ for all $\lambda : \mathcal{S}$ follows immediately since $1(n) \cdot \lambda(n) = \lambda(n) = \lambda(n) \cdot 1(n)$ for all $n : \mathbb{Z}$.

For the Eudoxus reals we can prove for all $a : \mathcal{E}$ that $a \cdot \mathsf{classof}\, 1 = a$ using the induction principle. This reduces the statement to the corresponding statement for slopes. Hence, using $\mathsf{classeq}$, we have to show that for all $\lambda : \mathcal{S}$ that $\lambda \cdot 1 \sim \lambda$, which we just concluded. Since the Eudoxus reals are a set, the statement '$a \cdot \mathsf{classof}\, 1 = a$ for all $a : \mathcal{E}$' is a mere proposition. Therefore the path constructors are respected when applying induction. Similarly, we prove that the identity map is the left identity. □

The associativity, commutativity and distributivity of both the addition and multiplication for the Eudoxus reals can be found in Mizrahi's thesis [35, Pro. 4.10, 4.22, 4.23, 4.51]. For slopes not all of these results are valid as not only the function $\mathbb{Z} \to \mathbb{Z}$ should be equal, but also the slope bounds. However, in the case of slopes we do always have an equivalence. As an example, if $\lambda, \mu : \mathcal{S}$ then in general $\lambda \cdot \mu = \mu \cdot \lambda$ does not hold, but we do have $\lambda \cdot \mu \sim \mu \cdot \lambda$.

The additive inverse can also straightforwardly be defined for both slopes and the Eudoxus reals. However, for slopes this is only the inverse up to equivalence.

Definition 5.17 (✓)**.** *Let $\lambda : \mathcal{S}$ be a slope. Then we define its additive inverse up to equivalence $-\lambda : \mathcal{S}$ by $(-\lambda)(n) = -\lambda(n)$ for all $n : \mathbb{Z}$ with bound $d_{-\lambda} :\equiv d_\lambda$. It hence satisfies $\lambda + (-\lambda) \sim 0 \sim (-\lambda) + \lambda$. By the recursion principle, this induces for all $a : \mathcal{E}$ an additive inverse $-a : \mathcal{E}$ such that $a + (-a) = 0 = (-a) + a$.*

Proof. It is trivial to check that the additive inverse of a slope is still a slope with the given bound and that for all $\lambda : \mathcal{S}$ it holds that $\lambda + (-\lambda) \sim 0$.

For the Eudoxus reals we can define the additive inverse using the reduction principle. By $\mathsf{classeq}$ it suffices to check that for all $\lambda, \mu : \mathcal{S}$ if $\lambda \sim \mu$ that $-\lambda \sim -\mu$. The proof of this is trivial. To show that for all $a : \mathcal{E}$ it holds that $a + (-a) = 0 = (-a) + a$, we use induction and $\mathsf{classeq}$ to reduce it to the equivalence statement for slopes which we already proved. The path constructors are respected since the goal is a mere proposition. □

The definition of the multiplicative inverse takes some more effort in our setting, since we need an apartness relation. Hence we will postpone this to Section 5.9. We have however already shown now that the Eudoxus reals form a commutative ring.

Corollary 5.18 (✓)**.** *The type of Eudoxus reals together with the constants $0, 1$ and operations $+, \cdot, -$ is a commutative ring.*

5.4 Ordering

Next we define an ordering on slopes. The definitions of A'Campo and Mizrahi differ here slightly as A'Campo first defines $0 < a$ by defining $0 < \lambda$ for $\lambda : \mathcal{S}$ and then continues to define $b < a$ for $a, b : \mathcal{E}$. Mizrahi starts with a more general strict ordering for slopes and then shows this induces a strict ordering on the Eudoxus reals. Their definitions are however equivalent.

We will follow the approach by Mizrahi and first define an ordering for slopes and then for the Eudoxus reals, without the intermediate step of comparing a slope or real to zero. We will however replace the notion of a set being (in)finite in Mizrahi's definition by the existence of a bound on the set. Furthermore, Mizrahi states two requirements for her strict ordering of which one corresponds to the slopes in question not being equivalent. Hence we will first define a non-strict ordering without the condition that the slopes are not equivalent and then define a strict ordering by adding this condition. We will truncate these orderings to be mere relations as required for an ordered field.

Definition 5.19 (✓)**.** *Let $\lambda, \mu : \mathcal{S}$ be slopes. Then*

- *λ is less than or equal to μ, denoted $\lambda \leq \mu$, if there merely exists $S : \mathbb{Z}$ such that for all $n : \mathbb{N}$ we have $S \leq \mu(n) - \lambda(n)$, and*
- *λ is strictly less than μ, denoted $\lambda < \mu$, if $\lambda \leq \mu$ and for all $T : \mathbb{Z}$ there merely exists $n : \mathbb{N}$ such that $T \leq \mu(n) - \lambda(n)$.*

The second condition of the strict inequality implies that the two slopes are not equivalent because their pointwise difference is unbounded. We only require the values $\mu(n) - \lambda(n)$ for $n : \mathbb{N}$ to be unbounded from above and not also from below, because the latter is impossible since $\lambda \leq \mu$.

The strict inequality for slopes can also be characterized pointwise as in the following lemma. This is shown by Mizrahi [35, Pro. 4.50], but her left-to-right implication is not constructive. We can only prove

this direction if we truncate the goal, so our formulation is slightly different from Mizrahi. Therefore we mention the lemma here although the proof follows almost directly from Mizrahi and the definition of $<$.

Lemma 5.20 (✓/✗). *Let $\lambda, \mu : \mathcal{S}$ be slopes. Then $\lambda < \mu$ if and only if there merely exists $n : \mathbb{N}_+$ such that $\mu(n) - \lambda(n) > d_\lambda + d_\mu$.*

Proof. Let $\lambda, \mu : \mathcal{S}$ and assume $\lambda < \mu$. Since the goal is a mere proposition, we can apply the induction principle for truncations to remove the truncation in the definition of $<$. Then we know that there exists $n : \mathbb{N}$ such that $\mu(n) - \lambda(n) > d_\lambda + d_\mu$. It follows that n is strictly positive since the triangle inequality implies that

$$|\mu(0) - \lambda(0)| \leq |\mu(0) - \mu(0) - \mu(0)| + |\lambda(0) - \lambda(0) - \lambda(0)| \leq d_\mu + d_\lambda.$$

For the other direction we refer to Mizrahi [35, Lem. 4.58]. □

Mizrahi [35, Pro. 4.56] proves that these orderings respects equivalences. She first shows that $\leq$ is preserved by equivalences, after which she shows the same for $<$. The first part of this proof is constructive, but the second part is not. Hence we will need to prove the second part ourselves, but since we also believe we can prove the first part easier, we will also prove this.

Lemma 5.21 (✓). *Let $\lambda_1, \lambda_2, \mu_1, \mu_2 : \mathcal{S}$ and assume $\lambda_1 \sim \lambda_2$ and $\mu_1 \sim \mu_2$. Then $\lambda_1 \leq \mu_1$ holds if and only if $\lambda_2 \leq \mu_2$. Likewise, $\lambda_1 < \mu_1$ holds if and only if $\lambda_2 < \mu_2$.*

Proof. Let $\lambda_1, \lambda_2, \mu_1, \mu_2 : \mathcal{S}$ be such that $\lambda_1 \sim \lambda_2$ and $\mu_1 \sim \mu_2$. Assume that $\lambda_1 \leq \mu_1$, then the other direction follows similarly. Since $\lambda_2 \leq \mu_2$ is a mere proposition we can apply the induction principle for truncations to $\lambda_1 \sim \lambda_2$, $\mu_1 \sim \mu_2$ and $\lambda_1 \leq \mu_1$. Then we know the following

$$\begin{aligned} \sum_{S_\mu : \mathbb{Z}} \prod_{n:\mathbb{N}} S_\mu &\leq \mu_2(n) - \mu_1(n), \\ \sum_{S_\lambda : \mathbb{Z}} \prod_{n:\mathbb{N}} S_\lambda &\leq \lambda_1(n) - \lambda_2(n), \\ \sum_{S_{\mu,\lambda} : \mathbb{Z}} \prod_{n:\mathbb{N}} S_{\mu,\lambda} &\leq \mu_1(n) - \lambda_1(n). \end{aligned} \tag{1}$$

Hence for all $n : \mathbb{N}$ we can bound $\mu_2(n) - \lambda_2(n)$ from below as follows

$$\begin{aligned} \mu_2(n) - \lambda_2(n) &= \mu_2(n) - \mu_1(n) + \mu_1(n) - \lambda_1(n) + \lambda_1(n) - \lambda_2(n) \\ &\geq S_\mu + S_{\mu,\lambda} + S_\lambda, \end{aligned} \tag{2}$$

which implies that $\lambda_2 \leq \mu_2$.

Now assume that $\lambda_1 < \mu_1$. Again we only prove one direction, since the other can be proven similarly. We again have all bounds in (1) available, since also $\lambda_2 < \mu_2$ is a mere proposition. However, this time we also know that

$$\prod_{T:\mathbb{Z}} \sum_{a:\mathbb{N}} T \leq \mu_1(a) - \lambda_1(a).$$

Let $T : \mathbb{Z}$ be given and let $a : \mathbb{N}$ be such that $T - S_\lambda - S_\mu \leq \mu_1(a) - \lambda_1(a)$. Then, similarly as in (2), we show that

$$\begin{aligned} \mu_2(n) - \lambda_2(n) &= \mu_2(n) - \mu_1(n) + \mu_1(n) - \lambda_1(n) + \lambda_1(n) - \lambda_2(n) \\ &\geq S_\mu + (T - S_\lambda - S_\mu) + S_\lambda \\ &= T, \end{aligned}$$

which implies that $\lambda_2 < \mu_2$. □

Hence we can define a non-strict and strict-ordering on the Eudoxus reals as follows.

Definition 5.22 (✓). *Let $a, b : \mathcal{E}$. Then define the order $\leq$ as*

$$a \leq b :\equiv \left(\prod_{\lambda,\mu:\mathcal{S}} \mathsf{classof}\, \lambda = a \to \mathsf{classof}\, \mu = b \to \lambda \leq \mu \right)$$

and, similarly, the order $<$ as

$$a < b :\equiv \left(\prod_{\lambda,\mu:\mathcal{S}} \mathsf{classof}\, \lambda = a \to \mathsf{classof}\, \mu = b \to \lambda < \mu \right).$$

To make working with these ordering easier we can replace the 'for all $\lambda, \mu : \mathcal{S}$' by a truncated 'there exists $\lambda, \mu : \mathcal{S}$'. In the following version of the lemma we replaced both 'for all' by 'there exists', but we can also just replace one of them. This is particularly useful if for instance $a, b, c : \mathcal{E}$ and we want to convert the statements $a < b$ and $b < c$, to statements for slopes and want the same slope representing b in both inequalities.

Note that since we are working with mere propositions, we could also have used univalence to conclude an equality.

Lemma 5.23 (✓). *Let $a, b : \mathcal{E}$ then*

$$a \leq b \Leftrightarrow \left\| \sum_{\lambda,\mu:\mathcal{S}} (\mathsf{classof}\, \lambda = a) \times (\mathsf{classof}\, \mu = b) \times (\lambda \leq \mu) \right\|$$

and similarly

$$a < b \Leftrightarrow \left\| \sum_{\lambda,\mu:\mathcal{S}} (\mathsf{classof}\, \lambda = a) \times (\mathsf{classof}\, \mu = b) \times (\lambda < \mu) \right\|.$$

Proof. This follows from Lemma 5.21 and the observation that $\mathsf{classof}\, \lambda = \mathsf{classof}\, \mu$ if and only if $\lambda \sim \mu$ for all $\lambda, \mu : \mathcal{S}$ [41, Lem. 10.1.8]. □

Now we have to check that this ordering on the Eudoxus reals is a full pseudo semiring order. We will however not be able to prove this completely within this section. First of all since we have not yet defined an apartness relation, but we will also need some other theory we have yet to develop in future sections. We will however be able to show a few requirements for a full pseudo semi ring order in this section, and some other useful lemmas.

We will prove all following inequalities for slopes, but Lemma 5.23 and the surjectivity of $\mathsf{classof}$ can be used to convert all of these statements to the Eudoxus reals. We start with the observations that the strict-ordering is irreflexive and the non-strict order is reflexive.

Lemma 5.24 (✓). *For all $\lambda : \mathcal{S}$ we have $\lambda \leq \lambda$ and $\neg(\lambda < \lambda)$.*

Proof. Note that for all $n : \mathbb{N}$ we have $\lambda(n) - \lambda(n) = 0$, which implies $\lambda \leq \lambda$. If $\lambda < \lambda$ then this would imply $1 < 0$ for integers, which is a contradiction. □

Next we show that addition respects both the non-strict and strict ordering, although we actually prove a slightly stronger statement in case of the strict-ordering. That addition preserves the strict ordering, together with the existence of an additive inverse, implies that for all $a, b, c : \mathcal{E}$ we have $a < b$ if and only if $c + a < c + b$. This is requirement 6 for a full pseudo semiring order.

Lemma 5.25 (✓). *For all $\lambda_1, \lambda_2, \mu_1, \mu_2 : \mathcal{S}$ such that $\lambda_1 \leq \lambda_2$ and $\mu_1 \leq \mu_2$, we have $\lambda_1 + \mu_1 \leq \lambda_2 + \mu_2$.*

Proof. Let $\lambda_1, \lambda_2, \mu_1, \mu_2 : \mathcal{S}$ be satisfying the two hypotheses. Since $\leq$ is a mere relation, we can strip the truncations in both hypotheses and the goal. Then we know that

$$\begin{aligned} \sum_{S_\lambda:\mathbb{Z}} \prod_{n:\mathbb{N}} S_\lambda &\leq \lambda_2(n) - \lambda_1(n), \\ \sum_{S_\mu:\mathbb{Z}} \prod_{n:\mathbb{N}} S_\mu &\leq \mu_2(n) - \mu_1(n). \end{aligned} \tag{3}$$

Then it holds that for all $n : \mathbb{N}$ that

$$\lambda_2(n) + \mu_2(n) - \lambda_1(n) - \mu_2(n) \geq S_\lambda + S_\mu,$$

implying $\lambda_1 + \lambda_2 \leq \mu_1 + \mu_2$. □

Lemma 5.26 (✓). *For all $\lambda_1, \lambda_2, \mu_1, \mu_2 : \mathcal{S}$ such that $\lambda_1 \leq \lambda_2$ and $\mu_1 < \mu_2$, we have $\lambda_1 + \mu_1 < \lambda_2 + \mu_2$.*

Proof. Let $\lambda_1, \lambda_2, \mu_1, \mu_2 : \mathcal{S}$ be satisfying the two hypotheses. By Lemma 5.25 we have $\lambda_1 + \mu_1 \leq \lambda_2 + \mu_2$. In addition to (3) it now holds, after stripping the truncation, that

$$\prod_{T:\mathbb{Z}} \sum_{a:\mathbb{N}} T \leq \mu_2(a) - \mu_1(a).$$

To see that $\lambda_1 + \mu_1 < \lambda_2 + \mu_2$, let $T : \mathbb{Z}$ and take $a : \mathbb{N}$ such that $T - S_\lambda \leq \mu_2(a) - \mu_1(a)$. Then we have

$$\lambda_2(a) + \mu_2(a) - \lambda_1(a) - \mu_1(a) \geq (T - S_\lambda) + S_\lambda = T.$$

□

The next lemma about the transitivity of the orderings is not directly required for a full pseudo semiring order, but will however be useful in other proofs.

Lemma 5.27 (✓). *For all $\lambda, \mu, \kappa : \mathcal{S}$ such that $\lambda \leq \mu$ and $\mu \leq \kappa$ we have $\lambda \leq \kappa$. If either $\lambda < \mu$ or $\mu < \kappa$, then we have $\lambda < \kappa$.*

Proof. Let $\lambda, \mu, \kappa : \mathcal{S}$ be satisfying the two hypotheses of the first statement. Since $\leq$ is a mere relation, we can strip the truncations in both hypothesis and the goal. Then we have

$$\sum_{S_{\mu,\lambda}:\mathbb{Z}} \prod_{n:\mathbb{N}} S_{\mu,\lambda} \leq \mu(n) - \lambda(n),$$
$$\sum_{S_{\kappa,\mu}:\mathbb{Z}} \prod_{n:\mathbb{N}} S_{\kappa,\mu} \leq \kappa(n) - \mu(n).$$

Then it holds that for all $n : \mathbb{N}$ that

$$\kappa(n) - \lambda(n) = \kappa(n) - \mu(n) + \mu(n) - \lambda(n) \geq S_{\kappa,\mu} + S_{\mu,\lambda},$$

implying $\lambda \leq \kappa$.

For the second statement, assume $\lambda < \mu$ since the other case is analogous. Strip again the truncations, then we now know in addition that

$$\prod_{T:\mathbb{Z}} \sum_{a:\mathbb{N}} T \leq \mu(a) - \lambda(a).$$

To see that this implies a strict inequality, let $T : \mathbb{Z}$ and take $a : \mathbb{N}$ such that $T - S_{\kappa,\mu} \leq \mu(a) - \lambda(a)$. Then we have

$$\kappa(a) - \lambda(a) = \kappa(a) - \mu(a) + \mu(a) - \lambda(a) \geq S_{\kappa,\mu} + (T - S_{\kappa,\mu}) = T.$$

□

A first use of the previous lemma is the next lemma, which is a requirement for a full pseudo semiring order.

Lemma 5.28 (✓). *For all $\lambda, \mu : \mathcal{S}$ we have $\neg(\lambda < \mu \wedge \mu < \lambda)$.*

Proof. Let $\lambda, \mu : \mathcal{S}$ be such that $\lambda < \mu$ and $\mu < \lambda$. Then by Lemma 5.27 we have $\mu < \mu$, which is a contradiction by Lemma 5.24. □

We immediately continue with another requirement for a full pseudo semiring order. However, we can only prove one direction at this moment, because we need to use sequences of slopes for the other direction. Hence we postpone the proof of the reverse direction to Lemma 5.51.

Lemma 5.29 (✓). *For all $\lambda, \mu : \mathcal{S}$ such that $\lambda \leq \mu$, we have $\neg(\mu < \lambda)$.*

Proof. Let $\lambda, \mu : \mathcal{S}$ such that $\lambda \leq \mu$ and $\mu < \lambda$. Then by Lemma 5.27 we have $\mu < \mu$, which leads to $\mathbf{0}$ by Lemma 5.24. □

The next requirement for a full pseudo semi ring order we want to prove is that if we multiply two positive reals, the result is again a positive real. A more general statement than this is proven by Mizrahi [35, Pro. 4.63.], but we prove here a simpler version where all inequalities are with respect to 0. This is namely all that the HoTT library requires for a full pseudo semiring order. The statement of Mizrahi then follows from the general theory of a full pseudo semi ring order [2]

We first show that the multiplication of two non-negative Eudoxus reals is again non-negative before we show the result for positive reals.

Lemma 5.30 (✓). *For all $\lambda, \mu : \mathcal{S}$ such that $0 \leq \lambda$ and $0 \leq \mu$, we have $0 \leq \lambda \cdot \mu$.*

[2] The `Global Instance nonneg_mult_le_l` in `HoTT/Classes/orders/semirings`.

Proof. Let $\lambda, \mu : \mathcal{S}$ satisfy the hypotheses. Stripping the truncations, we know that

$$\begin{aligned} \sum_{S_\lambda:\mathbb{Z}} \prod_{n:\mathbb{N}} S_\lambda &\leq \lambda(n), \\ \sum_{S_\mu:\mathbb{Z}} \prod_{n:\mathbb{N}} S_\mu &\leq \mu(n). \end{aligned} \tag{4}$$

Hence we can prove by induction on S_μ that for all $n : \mathbb{N}$ it holds that $\lambda(\mu(n)) \geq \min\{S_\lambda, \lambda(S_\mu), \lambda(S_\mu + 1), \lambda(S_\mu + 2), \ldots, \lambda(-1)\}$. In the case that $0 \leq S_\mu$, this reduces to showing that $S_\lambda \leq \lambda(\mu(n))$ which immediately follows from (4). □

Lemma 5.31 (✓)**.** *For all $\lambda, \mu : \mathcal{S}$ such that $0 < \lambda$ and $0 < \mu$, we have $0 < \lambda \cdot \mu$.*

Proof. Let $\lambda, \mu : \mathcal{S}$ satisfy the hypotheses. By Lemma 5.30, we know that $\lambda \cdot \mu \geq 0$, so left is to prove the strict inequality. Stripping the truncations, we know in addition to (4) that

$$\begin{aligned} \prod_{T:\mathbb{Z}} \sum_{a:\mathbb{N}} T &\leq \lambda(a) \\ \prod_{T:\mathbb{Z}} \sum_{b:\mathbb{N}} T &\leq \mu(b). \end{aligned}$$

Let $T : \mathbb{Z}$ be given. Then there exists $a : \mathbb{N}$ such that $\lambda(a) \geq T - S_\lambda + d_\lambda$ and in turn there exists $b : \mathbb{N}$ such that $\mu(b) \geq a$. Define $d :\equiv \mu(b) - a$, then it holds that $0 \leq d$. Therefore we have

$$\lambda(\mu(b)) = \lambda(a + d) \geq \lambda(a) + \lambda(d) - d_\lambda \geq (T - S_\lambda + d_\lambda) + S_\lambda - d_\lambda = T,$$

which implies $\lambda \cdot \mu > 0$. □

For the ordering on the Eudoxus reals to be a full pseudo semiring order we also need to show requirement 7. This states for all $a, b : \mathcal{E}$ that if $\neg(b < a)$ there exists $c : \mathcal{E}$ such that $b = a + c$. We however do not even need the hypothesis to construct such c, since $b = a + (b - a)$.

These are all the requirements we can prove for a full pseudo semiring order so far. We will however end this section with the proof that the non-strict ordering is antisymmetric, since we will need it to define a lattice in Section 5.10.

Lemma 5.32 (✓)**.** *For all $\lambda, \mu : \mathcal{S}$ such that $\lambda \leq \mu$ and $\mu \leq \lambda$ we have $\mu \sim \lambda$.*

Proof. Let $\lambda, \mu : \mathcal{S}$ be such that the hypotheses hold. Then

$$\begin{aligned} \sum_{S_{\mu,\lambda}:\mathbb{Z}} \prod_{n:\mathbb{N}} S_{\mu,\lambda} &\leq \mu(n) - \lambda(n) \\ \sum_{S_{\lambda,\mu}:\mathbb{Z}} \prod_{n:\mathbb{N}} S_{\lambda,\mu} &\leq \lambda(n) - \mu(n) \end{aligned}$$

Hence we have $-(|S_{\mu,\lambda}| + |S_{\lambda,\mu}|) \leq \lambda(n) - \mu(n) \leq |S_{\mu,\lambda}| + |S_{\lambda,\mu}|$ and so $\lambda \sim \mu$. □

5.5 Absolute value

The absolute value of a slope or Eudoxus real was not yet defined by A'Campo and Mizrahi. We will however use it to define sequences and to define a closeness relation in Section 7.

Note that we cannot define the absolute value of a slope by taking the absolute value pointwise on the whole of $\mathbb{Z}$. If we would do so, the absolute value of the identity slope would not even be a slope. We can however take the absolute value pointwise on $\mathbb{N}_+$ and extend the result oddly.

Definition 5.33 (✓)**.** *Let $\lambda : \mathcal{S}$, then we define the absolute value of λ, denoted $|\lambda| : \mathcal{S}$, as the odd extension of $|\lambda|(n) = |\lambda(n)|$ for $n > 0$ with bound $d_{|\lambda|} :\equiv 7d_\lambda$.*

Proof. We only need to estimate $\big||\lambda|(m+n) - |\lambda|(m) - |\lambda|(n)\big|$ for $m, n > 0$ since $|\lambda|$ is defined as an odd extension [35, Pro. 4.26]. Hence take $m, n : \mathbb{N}_+$. We use the decidability of the ordering relation on $\mathbb{Z}$ to distinguish three cases.

The first case is when $\lambda(m+n) > d_\lambda$, $\lambda(m) > d_\lambda$ or $\lambda(n) > d_\lambda$. Then for all $a > 0$ we have $\lambda(a) \geq -d_\lambda$ [35, Pro. 4.50] and so for all $a > 0$, it holds that $\big||\lambda(a)| - \lambda(a)\big| \leq 2d_\lambda$. Then using the triangle inequality we see that

$$\begin{aligned}\big||\lambda|(m+n) - |\lambda|(m) - |\lambda|(n)\big| &= \big||\lambda(m+n)| - |\lambda(m)| - |\lambda(n)|\big| \\ &= \Big|\big(|\lambda(m+n)| - \lambda(m+n)\big) + \lambda(m+n) - \big(|\lambda(m)| - \lambda(m)\big) \\ &\quad - \lambda(m) - \big(|\lambda(n)| - \lambda(n)\big) - \lambda(n)\Big| \\ &\leq 2d_\lambda + 2d_\lambda + 2d_\lambda + d_\lambda \\ &= 7d_\lambda.\end{aligned}$$

In the second case, we have $\lambda(m+n) < -d_\lambda$, $\lambda(m) < -d_\lambda$ or $\lambda(n) < -d_\lambda$. We can reduce this case to the first one by taking $-\lambda$.

In the third case, it holds that $|\lambda(m+n)|, |\lambda(m)|, |\lambda(n)| \leq d_\lambda$. Then by the triangle inequality it holds that

$$\begin{aligned}\big||\lambda|(m+n) - |\lambda|(m) - |\lambda|(n)\big| &\leq |\lambda(m+n)| + |\lambda(m)| + |\lambda(n)| \\ &\leq 3d_\lambda.\end{aligned}$$

□

Definition 5.34 (✓). *By the recursion principle, the absolute value function on slopes induces an absolute value function* $|\cdot| : \mathcal{E} \to \mathcal{E}$.

Proof. By applying the recursion principle and using classeq, this reduces to proving for all $\lambda, \mu : \mathcal{S}$ that $\lambda \sim \mu$ implies $|\lambda| \sim |\mu|$. Hence take $\lambda, \mu : \mathcal{S}$ such that $\lambda \sim \mu$. Then the equivalence follows from the reverse triangle inequality on $\mathbb{Z}$, since for all $n : \mathbb{Z}$ we have

$$\big||\lambda|(n) - |\mu|(n)\big| = \big||\lambda(n)| - |\mu(n)|\big| \leq |\lambda(n) - \mu(n)|$$

which is bounded as $\lambda \sim \mu$. □

Note that it follows directly from the definition of the absolute value that for all $\lambda : \mathcal{S}$ it holds that $0 \leq |\lambda|$. Moreover, if the slope is already positive, the slope is equivalent to its absolute value.

Lemma 5.35 (✓). *Let* $\lambda : \mathcal{S}$ *such that* $0 \leq \lambda$, *then* $\lambda \sim |\lambda|$.

Proof. Take $\lambda : \mathcal{S}$ and let λ' be the odd slope equivalent to λ. Then it suffices to show that $\lambda' \sim |\lambda'|$. By Lemma 5.21 we have $0 \leq \lambda'$. We strip the truncation and find that there exists $S : \mathbb{Z}$ such that $S \leq \lambda'(n)$ for all $n : \mathbb{N}$. Hence for all $n : \mathbb{N}_+$ we have $-2|S| \leq |\lambda'|(n) - \lambda'(n) \leq 2|S|$. Since both $|\lambda'|$ and λ' are odd we conclude that they are equivalent. Therefore it follows that $\lambda \sim |\lambda|$ since the equivalence relation is transitive. □

Next we will prove two familiar statements involving the absolute value. The first states that the absolute value and multiplication commute, although for slopes only up to equivalence.

Lemma 5.36 (✓). *Let* $\lambda, \mu : \mathcal{S}$ *then* $|\lambda\mu| \sim |\lambda||\mu|$.

Proof. Take $\lambda, \mu : \mathcal{S}$ and without loss of generality assume that λ is odd. To show that the two slopes are equivalent it suffices to show that they are pointwise equal for positive integers by Lemma 5.8. Hence take $n : \mathbb{N}_+$, then we distinguish three cases.

In the first case we have $\mu(n) = 0$. Then both functions evaluate to zero since λ is odd. We see this by first unfolding the definitions in the slope of the left-hand side

$$(|\lambda\mu|)(n) = |(\lambda\mu)(n)|) = |\lambda(\mu(n))| = |\lambda(0)| = 0$$

and then in the slope in the right-hand side

$$(|\lambda||\mu|)(n) = |\lambda|(|\mu(n)|) = |\lambda(0)| = 0.$$

If $0 < \mu(n)$, then we can forget about the absolute value in $|\mu(n)|$ as follows

$$(|\lambda||\mu|)(n) = |\lambda|(|\mu(n)|) = |\lambda(\mu(n))| = |(\lambda\mu)(n)| = |\lambda\mu|(n).$$

In the third and last case we have $\mu(n) < 0$. Then we use that λ is odd to get rid of the negation in $-\mu(n)$ as follows

$$(|\lambda||\mu|)(n) = |\lambda|(|\mu(n)|) = |\lambda(-\mu(n))| = |-\lambda(\mu(n))| = |\lambda(\mu(n))| = |\lambda\mu|(n).$$

□

The next familiar statement we will prove is the triangle inequality for slopes.

Lemma 5.37 (✓). *For all $\lambda, \mu : \mathcal{S}$ we have $|\lambda + \mu| \leq |\lambda| + |\mu|$.*

Proof. Take $\lambda, \mu : \mathcal{S}$. We know that for all $n : \mathbb{Z}$ we have $|\lambda(n) + \mu(n)| \leq |\lambda(n)| + |\mu(n)|$ by applying the triangle inequality on $\mathbb{Z}$. Hence for all $n : \mathbb{N}$ we have $0 \leq |\lambda|(n) + |\mu|(n) - |\lambda + \mu|(n)$ and so the desired inequality follows. □

5.6 Sequences

In this section we will develop some basic theory around sequences of slopes. We first however use the absolute value to define when a sequence of slopes is Cauchy and when a slope is a limit of a sequence of slopes.

Definition 5.38 (✗). *Let $(\lambda_n)_{n:\mathbb{N}}$ be a sequence of slopes in $\mathcal{S}$. Then $(\lambda_n)_{n:\mathbb{N}}$ is Cauchy if for all $\varepsilon : \mathcal{Q}_+$ there exists $N : \mathbb{N}$ such that for all $n, m : \mathbb{N}_+$ we have $|\lambda_n - \lambda_m| < \varepsilon$.*

Definition 5.39 (✗). *Let $(\lambda_n)_{n:\mathbb{N}}$ be a sequence of slopes in $\mathcal{S}$. Then $\lambda : \mathcal{S}$ is a limit of $(\lambda_n)_{n:\mathbb{N}}$ if for all $\varepsilon : \mathcal{Q}_+$ there exists $N : \mathbb{N}$ such that for all $n > N$ we have $|\lambda - \lambda_n| < \varepsilon$. We denote this by $\lambda_n \to \lambda$.*

Note that in the case of slopes we cannot speak of *the* limit of a sequence. If we have found a limit, we can namely alter its function value in a finite amount of points and the resulting slope will still be a limit for the sequence.

We can work with sequences of slopes as we are accustomed to, at least in the cases of the following lemma. The list is not complete, but suffices for our goals. The lemma and its proof are mostly based on a similar proposition of Bishop [10, Pro. 2.3.4], but are adapted to deal with slopes.

Lemma 5.40 (✗). *Let $(\lambda_n)_{n:\mathbb{N}}, (\mu_n)_{n:\mathbb{N}}$ be two sequences of slopes such that $\lambda_n \to \lambda : \mathcal{S}$ and $\mu_n \to \mu : \mathcal{S}$. Let $\kappa : \mathcal{S}$. Then*

1. *$\lambda_n + \mu_n \to \lambda + \mu$,*
2. *$-\lambda_n \to -\lambda$,*
3. *if $\lambda_n = \kappa$ for all $n : \mathbb{N}$, then $\lambda_n \to \kappa$,*
4. *if for all $n : \mathbb{Z}$ we have $\lambda_n \sim \mu_n$, then $\lambda_n \to \mu$,*
5. *$|\lambda_n| \to |\lambda|$,*
6. *if additionally $\lambda_n \to \mu$ then $\lambda \sim \mu$,*
7. *if for all $n : \mathbb{N}$ we have $\lambda_n \leq \kappa$, then $\lambda \leq \kappa$. Similarly if for all $n : \mathbb{N}$ we have $\kappa \leq \lambda_n$, then $\kappa \leq \lambda$.*

Proof.

1. We refer to Bishop [10, Pro. 2.3.4a].
2. This follows directly from $|(-\lambda) - (-\lambda_n)| = |\lambda - \lambda_n|$.
3. We refer to Bishop [10, Pro. 2.3.4d].
4. By the proof of Definition 5.35 we have $|\mu - \lambda_n| \sim |\mu - \mu_n|$. Then the claim follows from Lemma 5.21.
5. The reverse triangle inequality states that $||\lambda| - |\lambda_n|| \leq |\lambda - \lambda_n|$, from which the claim follows immediately.
6. This follows from the triangle inequality in Lemma 5.37, since it implies that $|\lambda - \mu| \leq |\lambda - \lambda_n| + |\lambda_n - \mu|$ for all $n : \mathbb{N}$.

7. We prove only the first statement, since the second follows analogously.

   Define the constant sequence $\kappa_n :\equiv \kappa$ for all $n : \mathbb{N}$ and consider the sequence $(\kappa_n - \lambda_n)_{n:\mathbb{N}}$. On the one hand, we know by items 1, 2 and 3 that $\kappa_n - \lambda_n \to \kappa - \lambda$. On the other hand, we know for all $n : \mathbb{N}$ that $\kappa_n - \lambda_n \sim |\kappa_n - \lambda_n|$ by Lemma 5.35. The latter sequence converges to $|\kappa - \lambda|$ by item 5 and hence by item 4 it follows that $\kappa_n - \lambda_n \to |\kappa - \lambda|$. Therefore we have $\kappa - \lambda \sim |\kappa - \lambda|$ by item 6 and since $|\kappa - \lambda|$ is nonnegative, it follows by Lemma 5.21 that $0 \leq \kappa - \lambda$.

□

This enables us to prove a very familiar statement from our usual implementation of the rationals.

Lemma 5.41 (✗). *The sequence $(1/m)_{m:\mathbb{N}}$ converges in $\mathcal{S}$ to 0.*

Proof. Let $\varepsilon = (\varepsilon_1, \varepsilon_2) : \mathcal{Q}_+$ be given. Define $M :\equiv \frac{1}{\varepsilon}(1) \equiv \min\{l : \mathbb{N} | \varepsilon_1 l \geq \varepsilon_2\}$. Take $m : \mathbb{N}$ such that $m > M$, then $1/m < 1/M$ and so it remains to prove $1/M \leq \varepsilon$. We show the inequality pointwise for all $n : \mathbb{N}$ as follows

$$\begin{aligned}
\frac{1}{M}(n) &\equiv \min\left\{k : \mathbb{N} \middle| \frac{1}{\varepsilon}(1)k \geq n\right\} \\
&= \min\left\{k : \mathbb{N} \middle| \varepsilon_1 \frac{1}{\varepsilon}(1)k \geq \varepsilon_1 n\right\} \\
&\leq \min\{k : \mathbb{N} | \varepsilon_2 k \geq \varepsilon_1 n\} \\
&\equiv \varepsilon(n).
\end{aligned}$$

Hence $0 \leq \varepsilon(n) - \frac{1}{M}(n)$ for all $n : \mathbb{N}$ and so $1/M \leq \varepsilon$. □

Similarly as for slopes, we can also use the absolute value on the Eudoxus reals to work with sequences. However, in this case we can speak of *the* limit of a sequence, which will follow from Lemma 7.9.

Definition 5.42 (✗). *A sequence $(a_n)_{n:\mathbb{N}}$ of Eudoxus reals is Cauchy if for all $\varepsilon : \mathbb{Q}_+$ there exists $N : \mathbb{N}$ such that for all $m, n \geq N$ we have $|a_m - a_n| < \varepsilon$.*

Definition 5.43 (✗). *Let $(a_n)_{n:\mathbb{N}}$ be a sequence of Eudoxus reals. Then $a : \mathcal{E}$ is a limit of $(a_n)_{n:\mathbb{N}}$ if for all $\varepsilon : \mathbb{Q}_+$ there exists $N > 0$ such that for all $n \geq N$ it holds that $|a_n - a| < \varepsilon$.*

5.7 Archimedean property

The Archimedean property has multiple formulations of which Mizrahi shows one alternative to hold for slopes. She shows that if $\lambda, \mu : \mathcal{S}$ satisfy $\lambda < \mu$ that there merely exists $n : \mathbb{N}$ such that $n\lambda > \mu$ [35, Pro. 4.71], where we added the truncation to make the statement hold in our setting. We are however more interested in the formulation used by the HoTT library. This defines the Archimedean property as the mere existence of $q : \mathcal{Q}$ such that $\lambda < q < \mu$. Strictly speaking, the rationals numbers $\mathbb{Q}$ are used in the Archimedean property instead of $\mathcal{Q}$, but we have not defined $\mathbb{Q}$ as a subset of slopes but of the Eudoxus reals. Hence we will only be able to prove this formulation with $\mathcal{Q}$ taking on the role of $\mathbb{Q}$, but this will imply the Archimedean property with $\mathbb{Q}$ for the Eudoxus reals. This formulation can also be stated as that $\mathcal{Q}$ is dense in $\mathcal{S}$.

In the case of real numbers in classical mathematics, the formulation of Mizrahi implies the formulation of the HoTT library as shown for example by Rudin [37, Th. 1.20]. However, as far as we know, in such proofs the well-ordering of the natural numbers is used in determining the smallest element of $\{n : \mathbb{N} | x < n\}$ for some $x : \mathbb{R}$. When working with slopes, the condition $\lambda < n$ is not decidable and hence we cannot decide for all $n : \mathbb{N}$ whether they are an element of the set (or rather type). Therefore, we prove the density of $\mathcal{Q}$ in $\mathcal{S}$ more directly. First we however prove a different lemma which we will need for this.

Lemma 5.44 (✗). *Let $a, b : \mathbb{Z}$ be such that $b - a \geq 2l$ for some $l : \mathbb{N}$. Then $b - \frac{1}{2}(a + b) \geq l$ and $\frac{1}{2}(a + b) - a \geq l$.*

Proof. For the first statement we distinguish the cases where $b - a$ is even and where it is odd. In the first case, there exists $n : \mathbb{N}$ such that $b - a = 2n$ and $n \geq l$. Then a rewriting of definitions results in

$$\begin{aligned} b - \frac{1}{2}(a + b) &= b - \min\{k : \mathbb{N} | 2k \geq a + b\} \\ &= b - \min\{k : \mathbb{N} | 2k \geq 2(a + n)\} \\ &= b - a - n \\ &= n \\ &\geq l. \end{aligned}$$

In the second case $b - a$ is odd and hence there exists $n : \mathbb{N}$ such that $b - a = 2n - 1$ and $n > l$. Then compared to our reasoning in the previous case we need to make an extra estimate along the way, but we still end up with the same lower bound.

$$\begin{aligned} b - \frac{1}{2}(a + b) &= b - \min\{k : \mathbb{N} | 2k \geq a + b\} \\ &= b - \min\{k : \mathbb{N} | 2k \geq 2(a + n) - 1\} \\ &\geq b - \min\{k : \mathbb{N} | 2k \geq 2(a + n)\} \\ &= b - a - n \\ &= n - 1 \\ &\geq l. \end{aligned}$$

The second statement can be proven similarly. □

Lemma 5.45 (✗). *Let $\lambda, \mu : \mathcal{S}$ be two slopes such that $\lambda < \mu$. Then there merely exists $q : \mathcal{Q}$ such that $\lambda < q < \mu$.*

Proof. We can assume by Lemma 5.21 that the slopes $\lambda, \mu : \mathcal{S}$ are well-adjusted slopes. Since our goal is a mere proposition we can strip the truncation in $\lambda < \mu$, so that we know that there exists $a : \mathbb{N}$ such that $\mu(a) - \lambda(a) \geq 2$. Note that a is strictly positive because since λ, μ are odd slopes and hence $\lambda(0) = \mu(0) = 0$. Since we can also bound $|(\mu - \lambda)(m + n) - (\mu - \lambda)(m) - (\mu - \lambda)(n)|$ by 2, it follows [35, Pro. 4.50] that for all $n : \mathbb{N}_+$ we have $\mu(n) - \lambda(n) \geq -2 + n : a$.

In particular we have $\mu(8a) - \lambda(8a) \geq 6$. Now define $c :\equiv \frac{1}{2}(\mu(8a) + \lambda(8a))$ and $q :\equiv (c, 8a)$. We will assume $c > 0$ since the other case is analogous. Then $q(8a) \equiv \min\{k : \mathbb{N} | 8ak \geq 8ac\} = c$. To prove that $q < \mu$ we use Lemma 5.44, which implies

$$\mu(8a) - q(8a) = \mu(8a) - c \equiv \mu(8a) - \frac{1}{2}(\mu(8a) + \lambda(8a)) \geq 3 > 2 = d_\mu + d_q.$$

Hence by Lemma 5.20 we have $q < \mu$.

Similarly by applying Lemma 5.44 we observe the following

$$q(8a) - \lambda(8a) = c - \lambda(8a) = \frac{1}{2}(\mu(8a) + \lambda(8a)) - \lambda(8a) \geq 3 > 2 = d_q + d_\lambda.$$

Then Lemma 5.20 implies that $\lambda < q$. □

It immediately follows from this result that Eudoxus reals are Archimedean.

Corollary 5.46 (✗). *The Eudoxus reals are Archimedean: for all $a, b : \mathcal{E}$ such that $a < b$ there merely exists $q : \mathbb{Q}$ such that $a < q < b$.*

We can also use the Archimedean property for slopes to show that we can approximate every slope by a sequence of rational slopes.

Corollary 5.47 (✗). *Let $\lambda : \mathcal{S}$. Then there merely exists a sequence $(q_n)_{n:\mathbb{N}}$ such that $q_n \to \lambda$ and $q_n : \mathcal{Q}$ for all $n : \mathbb{N}$. Moreover, we can construct $(q_n)_{n:\mathbb{N}}$ such that either for all $n : \mathbb{N}$ we have $q_n < \lambda$ or for all $n : \mathbb{N}$ we have $q_n > \lambda$.*

Proof. Take $\lambda : \mathcal{S}$, then by Lemma 5.45 there merely exists $q_n : \mathcal{Q}$ such that $\lambda - 1/n < q_n < \lambda$ for all $n : \mathbb{N}$. To see that $(q_n)_{n:\mathbb{N}}$ converges to λ, take $\varepsilon : \mathcal{Q}_+$ and define $N :\equiv \frac{1}{\varepsilon}(1)$. Then by the proof of Lemma 5.41 we have $\frac{1}{N} \leq \varepsilon$ and so for all $n \geq N$ it follows that $|\lambda - q_n| \sim \lambda - q_n < \frac{1}{n} \leq \frac{1}{N} \leq \varepsilon$. Hence the sequence $(q_n)_{n:\mathbb{N}}$ converges to λ. □

5.8 Apartness

In constructive mathematics we need in general a stronger notion than the negation of equality to define division [41, p. 377]. A proof of $\lambda \neq 0$ does namely not imply a proof of $\lambda > 0$ or a similar (positive) statement which we often need to construct the multiplicative inverse. Therefore we introduce the notion of an apartness relation as required for a type to be an ordered field.

Definition 5.48. *Let $A : \mathcal{U}$ be a type. We say that $\# : A \to A \to \mathsf{Prop}$ is an apartness relation if for all $x, y, z : A$ we have*

1. *irreflexivity: $\neg(x\#x)$*
2. *symmetry: if $x\#y$ then $y\#x$*
3. *cotransitivity: if $x\#y$ then merely $x\#z$ or $z\#y$.*

If additionally $\neg(x\#y)$ implies $x = y$, then we call $\#$ tight.

The definition of an apartness relation we state here is slightly different from the one in the HoTT library since the HoTT library always requires an apartness relation to be tight. Hence in Section 4 we explicitly required a tight apartness relation.

To see that an apartness relation is indeed stronger than the negation of equality take $x, y : A$, where $A : \mathcal{U}$. Assume $x\#y$ and to derive $\neg(x = y)$ assume also $x = y$. Then it follows $x\#x$, which leads to a contradiction. The converse, that $\neg(x = y)$ implies $x\#y$, requires excluded middle [41, p. 377].

For the Eudoxus reals we use the strict ordering to define $a\#b = (a < b) + (b < a)$ for $a, b : \mathcal{E}$. To see that this is indeed an apartness relation we first need to prove the cotransitivity of $<$ for slopes. In order to do so, we will in turn first prove the cotransitivity in the special case where we are given an inequality of two rational slopes. Then to prove the general case we will reduce it to this special case. Note that the cotransitivity of the strict ordering is also a requirement for a full pseudo semiring order.

Lemma 5.49 (✗). *Let $q, r : \mathcal{Q}$ be such that $q < r$. Then for all $\kappa : \mathcal{S}$, we merely have $q < \kappa$ or $\kappa < r$.*

Proof. Let $q, r : \mathcal{Q}$ and $\kappa : \mathcal{S}$ be given such that $q < r$. By Lemma 5.45 there merely exist $s, t : \mathcal{Q}$ such that $\kappa - (r - q)/2 < s < \kappa$ and $\kappa < t < \kappa + (r - q)/2$. We can strip the truncations, since our goal is a mere proposition. Using the decidability of the ordering on $\mathcal{Q}$, we can now distinguish the cases $q < s$ and $s \leq q$. The first case implies by transitivity of $<$ that $q < \kappa$. In the second case assume $t > r$ for a contradiction. On the one hand we have $t - s > r - s \geq r - q$, whereas on the other hand we have $t - s = (t - \kappa) + (\kappa - s) < (r - q)/2 + (r - q)/2 \sim r - q$. This constitutes a contradiction by Lemma 5.21. Hence it must be the case that $t \leq r$ and so it follows that $\kappa < r$. □

Lemma 5.50 (✓). *Let $\lambda, \mu : \mathcal{S}$ be such that $\lambda < \mu$. Then for all $\kappa : \mathcal{S}$, we merely have $\lambda < \kappa$ or $\kappa < \mu$.*

Proof. Since our goal is a mere proposition, we can forget about the mere existence in the lemmas we will use in this proof. Let $\lambda, \mu : \mathcal{S}$ be such that $\lambda < \mu$ and take $\kappa : \mathcal{S}$. By applying the density of $\mathcal{Q}$ from Lemma 5.45 twice, we know there exists $q, r : \mathcal{Q}$ such that $\lambda < q < r < \mu$. By applying Lemma 5.49 it holds that $q < \kappa$ or $\kappa < r$. The former implies $\lambda < \kappa$ and the latter $\kappa < \mu$. □

Using the cotransitivity of $<$ we are also finally able to prove the reverse direction of Lemma 5.29, which was in turn also a requirement for a full pseudo semi ring order. The proof is similar to a proof of Gilbert [25, Lemma 4.8].

Lemma 5.51 (✗). *For all $\lambda, \mu : \mathcal{S}$ such that $\neg(\mu < \lambda)$, we have $\lambda \leq \mu$.*

Proof. Take $\lambda, \mu : \mathcal{S}$ such that $\neg(\mu < \lambda)$ and take $q : \mathcal{Q}_+$. Then $\lambda - q < \lambda$ by Lemma 5.26. By the cotransitivity of $<$ in Lemma 5.50 we merely have $\mu < \lambda$ or $\lambda - q < \mu$. Since our goal $\lambda \leq \mu$ is a mere proposition, we can forget about the merely and distinguish two cases.

In the first case we have $\mu < \lambda$ which leads to a contradiction with the assumption $\neg(\mu < \lambda)$. In the second case we have $\lambda - q < \mu$. Since q was arbitrary we have $\lambda - 1/m < \mu$ for all $m : \mathbb{N}$. By Lemma 5.41 and Lemma 5.40.7 it then follows that $\lambda \leq \mu$. □

Now we are able to prove that the relation we had in mind is indeed an apartness relation for the Eudoxus reals.

Lemma 5.52 (✓). *Define $\# : \mathcal{E} \to \mathcal{E} \to \mathsf{Prop}$ as $a\#b :\equiv (a < b) + (b < a)$. Then $\#$ is a tight apartness relation.*

Proof. First note that we do not need to truncate the type $(a < b) + (b < a)$ in the definition of $a\#b$ for it to be a mere proposition, since it is a disjoint sum [3]. That is, if both $a < b$ and $b < a$ hold we have a contradiction by Lemma 5.24.

Then we check the conditions from Definition 5.48.

1. This follows from the irreflexivity of $<$ for slopes in Lemma 5.24.
2. This follows from the symmetry of $+$.
3. Let $a, b, c : \mathcal{E}$ be such that $a\#b$, then $a < b$ or $b < a$. We only prove this lemma in the case $a < b$, since the second case is similar. By Lemma 5.50 we know that then either $a < c$ or $c < b$. In the former case we have $a\#c$ and in the latter $c\#b$. Therefore it holds that $a\#c$ or $c\#b$.
4. Let $a, b : \mathcal{E}$ be such that $\neg(a\#b)$. Then by definition we have $\neg((a < b) + (b < a))$, which implies $\neg(a < b)$ and $\neg(b < a)$. By Lemma 5.51 it follows that $b \leq a$ and $a \leq b$. Using Lemma 5.32 we then know $a = b$.

□

After defining this apartness relation we can immediately conclude that the type of Eudoxus reals satisfies a few more requirements of an ordered field.

Concerning the requirements for a field, we can conclude that $0\#1$, since $0 < 1$ in the type of Eudoxus reals. Furthermore, we can also conclude the strong binary extensionality of addition since by Lemma 5.26 addition preserves the strict ordering.

5.9 Division

Since we now have defined an apartness relation, we can construct the multiplicative inverse of an Eudoxus real apart from 0. However, we have to be careful working with the truncations in this instance, because for slopes we only know the mere existence of an inverse and for the Eudoxus reals we want an untruncated existence. We will use a workaround discussed in the HoTT book [41, just after Cor. 3.9.2].

Lemma 5.53 (✓)**.** *Let $\lambda : \mathcal{S}$ be such that $\lambda < 0$ or $0 < \lambda$. Then there merely exists $\mu : \mathcal{S}$ such that $\lambda \cdot \mu \sim 1$. Hence, if $a : \mathcal{E}$ such that $a\#0$ then there exists $b : \mathcal{E}$ such that $a \cdot b = 1$.*

Proof. Let $\lambda : \mathcal{S}$ be such that $0 < \lambda$ or $\lambda < 0$. Without loss of generality we can assume λ to be well-adjusted. In both cases we can strip the truncation, so that we know there exists $n : \mathbb{N}$ such that $|\lambda(n)| > 1$. From this point onward, we can follow the proof by Mizrahi [35, Pro. 4.52] from which the mere existence of the multiplicative inverse of λ follows.

For the Eudoxus reals, take $a : \mathcal{E}$ such that $a\#0$. We first note that the existence of a multiplicative inverse for an Eudoxus real is a mere proposition. To see this take $(b_1, p_1), (b_2, p_2) : \sum_{b:\mathcal{E}} ab = 1$. Then it follows that $b_1 = b_1 a b_2 = b_2$. If we denote this witness of $b_1 = b_2$ by q, then it follows that $q_* p_1 = p_2$ since the Eudoxus reals are a set. Hence $(b_1, p_1) = (b_2, p_2)$ and $\sum_{b:\mathcal{E}} ab = 1$ is indeed a mere proposition.

To define a function $\prod_{a:\mathcal{E}} a\#0 \to \sum_{b:\mathcal{E}} a \cdot b = 1$, it suffices by induction to construct a function

$$d : \prod_{\lambda:\mathcal{S}} ((\lambda < 0) + (0 < \lambda)) \to \sum_{b:\mathcal{E}} \mathsf{classof}(\lambda) \cdot b = 1.$$

The path constructors are taken into account since we have just shown that our goal is a mere proposition. Now given a slope $\lambda : \mathcal{S}$ and a proof $p : (\lambda < 0) + (0 < \lambda)$, we know by the first part of this lemma that the multiplicative inverse μ of λ exists up to equivalence. We are able to strip the truncation here, because our goal is a mere proposition. Then we can define $\mathsf{pr}_1\, d(\lambda, p) :\equiv \mathsf{classof}\, \mu$ and it follows that $\mathsf{classof}(\lambda) \cdot \mathsf{classof}(\mu) = 1$. □

5.10 Lattice

To finish our proof that the Eudoxus reals are an ordered field, it only remains to show that it is a lattice and to show the strong binary extensionality of the multiplication. We will show in this section that the Eudoxus reals are a lattice, and will show in the next section, Section 5.11, the strong binary extensionality.

To define the Eudoxus reals as a lattice, we need to define both a join and meet, which are similar to a supremum and infimum respectively. Their definitions are as follows.

[3] For this we refer to `ishprop_sum` in Types/Sum.v in the HoTT library.

Definition 5.54. *Let $A : \mathsf{Set}$ be a type and let $\leq\colon A \to A \to A$ be a transitive, antisymmetric, reflexive and mere relation* [4]*. Then a relation $\vee : A \to A \to A$ is a join if for all $a, b, c : A$ we have*

- $a \leq (a \vee b)$
- $b \leq (a \vee b)$
- *if $a \leq c$ and $b \leq c$ then $(a \vee b) \leq c$.*

A relation $\wedge : A \to A \to A$ is a meet if for all $a, b, c : A$ we have

- $(a \wedge b) \leq a$
- $(a \wedge b) \leq b$
- *if $c \leq a$ and $c \leq b$ then $c \leq (a \wedge b)$.*

We have already shown that the non-strict ordering $\leq$ for slopes and the Eudoxus reals is transitive in Lemma 5.27 and reflexive in Lemma 5.24. However, only for the Eudoxus reals the ordering is antisymmetric, which we see in Lemma 5.32. Hence we will not be able to show that the type of slopes is a lattice. However we will define functions on slopes which will induce a join and meet on the Eudoxus reals.

Definition 5.55 (✓). *Let $\lambda, \mu : \mathcal{S}$ be slopes. Then we define their maximum $\max(\lambda, \mu) : \mathcal{S}$ as the odd extension of $\max(\lambda, \mu)(n) :\equiv \max(\lambda(n), \mu(n))$ for $n > 0$ with bound $2d_\lambda + 2d_\mu$. We define their minimum $\min(\lambda, \mu) : \mathcal{S}$ as the odd extension of $\min(\lambda, \mu)(n) :\equiv \min(\lambda(n), \mu(n))$ for $n > 0$ with bound $2d_\lambda + 2d_\mu$.*

Proof. We will prove only for the maximum that the stated bound is indeed a bound to the function, since the proof for the minimum is similar.

Let $\lambda, \mu : \mathcal{S}$. Since max is defined as an odd extension, we only need to show that $2d_\lambda + 2d_\mu$ bounds $|\max(\lambda, \mu)(m + n) - \max(\lambda, \mu)(m) - \max(\lambda, \mu)(n)|$ for $m, n > 0$ [35, Pro. 4.26]. We prove three cases from which the remaining cases follow analogously.

In the first case we have $\max(\lambda, \mu)(m+n) = \lambda(m+n)$, $\max(\lambda, \mu)(m) = \lambda(m)$ and $\max(\lambda, \mu)(n) = \lambda(n)$. Then we can just apply the proof that d_λ is a bound of λ.

In the second case we have $\max(\lambda, \mu)(m + n) = \lambda(m + n)$, $\max(\lambda, \mu)(m) = \mu(m)$ and $\max(\lambda, \mu)(n) = \mu(n)$. Then using the properties of the maximum on $\mathbb{Z}$, we see that

$$\max(\lambda, \mu)(m + n) - \max(\lambda, \mu)(m) - \max(\lambda, \mu)(n) \leq \lambda(m + n) - \lambda(m) - \lambda(n) \leq d_\lambda \tag{5}$$

and

$$\max(\lambda, \mu)(m + n) - \max(\lambda, \mu)(m) - \max(\lambda, \mu)(n) \geq \mu(m + n) - \mu(m) - \mu(n) \geq -d_\mu.$$

In the third and last case we have $\max(\lambda, \mu)(m + n) = \lambda(m + n)$, $\max(\lambda, \mu)(m) = \lambda(m)$ and $\max(\lambda, \mu)(n) = \mu(n)$. Then the first inequality follows as in (5). For the other inequality, we now distinguish two subcases, since it holds that either $\lambda(n) - \mu(n) \geq -d_\lambda - d_\mu$ or $\lambda(n) - \mu(n) < -d_\lambda - d_\mu$. In the first subcase we have

$$\begin{aligned}\max(\lambda, \mu)(m + n) - \max(\lambda, \mu)(m) - \max(\lambda, \mu)(n) &= \lambda(m + n) - \lambda(m) - \mu(n)\\ &\geq \lambda(n) - d_\lambda - \mu(n)\\ &\geq -2d_\lambda - d_\mu.\end{aligned}$$

In the second subcase it follows that $\mu(m) - \lambda(m) \geq -d_\lambda - d_\mu + m : n \geq -d_\lambda - d_\mu$ [35, Pro. 4.50]. Hence we have

$$\begin{aligned}\max(\lambda, \mu)(m + n) - \max(\lambda, \mu)(m) - \max(\lambda, \mu)(n) &= \lambda(m + n) - \lambda(m) - \mu(n)\\ &\geq \lambda(m + n) - \lambda(m) - \mu(m + n) + \mu(m) - d_\mu\\ &\geq -d_\lambda - 2d_\mu.\end{aligned}$$

□

This new function max can be related to the absolute value function we defined earlier in Section 5.5.

Lemma 5.56 (✓). *Let $\lambda : \mathcal{S}$. Then $|\lambda| \sim \max(\lambda, -\lambda)$.*

[4] That is, $\leq$ is a partial order.

Proof. Let $\lambda : \mathcal{S}$. Since both the absolute value and the join are defined as an odd extension, it suffices to show for $n : \mathbb{N}_+$ that there exists $S : \mathbb{Z}$ such that

$$-S \leq |\lambda(n)| - \max(\lambda(n), -\lambda(n)) \leq S.$$

We see that we can take as bound 0, since for all $a : \mathbb{Z}$ we have $|a| = \max(a, -a)$. □

As stated earlier, the functions max and min on slopes are not a join or meet, since the order $\leq$ is not antisymmetric. They do however satisfies all other properties as the following lemma shows.

Lemma 5.57 (✓). *The function* $\max, \min : \mathcal{S} \to \mathcal{S} \to \mathcal{S}$ *satisfy for all* $\lambda, \mu, \kappa : \mathcal{S}$ *the following*

- $\min(\lambda, \mu) \leq \lambda \leq \max(\lambda, \mu)$
- $\min(\lambda, \mu) \leq \mu \leq \max(\lambda, \mu)$
- *if* $\lambda \leq \kappa$ *and* $\mu \leq \kappa$, *then* $\max(\lambda, \mu) \leq \kappa$
- *if* $\kappa \leq \lambda$ *and* $\kappa \leq \mu$, *then* $\kappa \leq \min(\lambda, \mu)$.

Proof. We again show this only for the maximum, as the proof for the minimum is similar.

Let $\lambda, \mu : \mathcal{S}$. To see that $\lambda \leq \max(\lambda, \mu)$ note that for all $n : \mathbb{N}_+$ we have $0 \leq \max(\lambda(n), \mu(n)) - \lambda(n)$. Analogously we see that $\mu \leq \max(\lambda, \mu)$.

Let $\kappa : \mathcal{S}$ be such that $\lambda \leq \kappa$ and $\mu \leq \kappa$. Then for all $n : \mathbb{N}_+$ we have

$$\kappa(n) - \max(\lambda(n), \mu(n)) = \kappa(n) + \min(-\lambda(n), -\mu(n)) = \min(\kappa(n) - \lambda(n), \kappa(n) - \mu(n)).$$

Hence $\kappa(n) - \max(\lambda(n), \mu(n))$ is bounded from below for $n : \mathbb{N}_+$, as both $\kappa(n) - \lambda(n)$ and $\kappa(n) - \mu(n)$ are bounded from below. Therefore we have $\max(\lambda(n), \mu(n)) \leq \kappa(n)$. □

The minimum and maximum on slopes induce functions on the Eudoxus reals, which are respectively a meet and a join.

Definition 5.58 (✓). *By the recursion principle, the function* $\max : \mathcal{S} \to \mathcal{S} \to \mathcal{S}$ *induces a function* $\max : \mathcal{E} \to \mathcal{E} \to \mathcal{E}$. *This function is a join.*

By the recursion principle, the $\min : \mathcal{S} \to \mathcal{S} \to \mathcal{S}$ *induces a function* $\min : \mathcal{E} \to \mathcal{E} \to \mathcal{E}$. *This function is a meet.*

Proof. We again show this only for the maximum, as the proof for the minimum is similar.

To apply the recursion principle it suffices by classeq to prove that for all $\lambda_1, \lambda_2, \mu_1, \mu_2 : \mathcal{S}$ if $\lambda_1 \sim \lambda_2$ and $\mu_1 \sim \mu_2$ that $\max(\lambda_1, \mu_1) \sim \max(\lambda_2, \mu_2)$. Hence let $\lambda_1, \lambda_2, \mu_1, \mu_2 : \mathcal{S}$ be such that $\lambda_1 \sim \lambda_2$ and $\mu_1 \sim \mu_2$. Since max is defined as an odd extension, it suffices to bound the difference between $\max(\lambda_1, \mu_1)(n)$ and $\max(\lambda_2, \mu_2)(n)$ for positive n. Since we are then just working with integers, we know that

$$|\max(\lambda_1(n), \mu_1(n)) - \max(\lambda_2(n), \mu_2(n))| \leq \max(|\lambda_2(n) - \lambda_1(n)|, |\mu_2(n) - \mu_1(n)|).$$

The right-hand side is bounded since $\lambda_1 \sim \lambda_2$ and $\mu_1 \sim \mu_2$, and hence it follows that $\max(\lambda_1, \mu_1) \sim \max(\lambda_2, \mu_2)$.

That this function is indeed a join follows from Lemma 5.57 and Lemma 5.23. □

Since we have constructed a join and meet on the Eudoxus reals, we have proven that the type is a lattice.

Corollary 5.59 (✓). *The Eudoxus reals together with the operations* min, max *and mere relation* $\leq$ *is a lattice.*

5.11 Strong binary extensionality of the multiplication

The last requirement remaining for the Eudoxus reals to be an ordered field is the strong binary extensionality of the multiplication. Since we have developed quite some theory by now for the Eudoxus reals, we will prove this result without using induction to reduce it to a statement for slopes.

We will follow here the proof(s) of Gilbert [25], since he proves this result in a similar setting although for a different type. Since he only wrote down these proofs in the Coq files accompanying his paper and not in the actual paper itself, we chose to state the proofs here. For each lemma we will refer to the name of the lemma in Gilbert's Coq file `full_ring`[5].

The first lemma is Gilbert's `Rmult_pos_decompose_nonneg`.

[5]The file can be found at `https://github.com/SkySkimmer/HoTTClasses/tree/master/theories/cauchy_reals`.

Lemma 5.60 (✓). *Let $a, b : \mathcal{E}$. Then if $0 \leq a$ and $0 < ab$ then $0 < b$.*

The proof of the lemma might take more work then one might expect at first glance. One might namely think that if $0 \leq a$ and $b \leq 0$ that then $ab \leq 0$ by Lemma 5.31, which is in contradiction with the hypothesis that $0 < ab$. However, this reasoning uses that if $0 < b$ then $\neg(b \leq 0)$ for $b : \mathcal{E}$, which we have not been able to show. Instead, we only have been able to show that $0 \leq b$ if and only if $\neg(b < 0)$ in respectively Lemma 5.51 and Lemma 5.29. We can still prove the lemma, but it will require some more work.

Proof of Lemma 5.60. Let $a, b : \mathcal{E}$ be such that $0 \leq a$ and $0 < ab$. Then first of all, it holds that $\neg(b < 0)$, because if we assume that $b < 0$, then by Lemma 5.30 we have $0 < ab \leq 0$ which is a contradiction. Hence we know that $0 \leq b$ using Lemma 5.51.

By the Archimedean property, there merely exists $q : \mathbb{Q}$ such that $0 < q < ab$ and $r : \mathbb{Q}$ such that $a < r < a + 1$. We strip the truncations, since our goal is a mere proposition. Then $0 < q/r$ and we claim $q/r \leq b$, implying that $0 < b$. We prove $q/r \leq b$ using Lemma 5.51 by showing that $\neg(b < q/r)$. Hence assume for a contradiction that $b < q/r$. Then we have $q < ab < r \cdot (q/r) = q$ which constitutes a contradiction by Lemma 5.24. □

The next lemma is similar to the requirement that max is the least upper bound, but now we have strict inequalities instead of non-strict inequalities. Gilbert proves this result in `Rlt_join`.

Lemma 5.61 (✓). *Let $a, b, c : \mathcal{E}$. Then if $a < c$ and $b < c$ then $\max(a, b) < c$.*

Proof. Take $a, b, c : \mathcal{E}$ such that $a < c$ and $b < c$. Then by the Archimedean property of Lemma 5.45, there exists $e_1, e_2 : \mathbb{Q}_+$ such that $a + e_1 \leq c$ and $b + e_2 \leq c$. Now define $e :\equiv \min(e_1, e_2)$, then we can conclude that $0 < e$ since we are working with $\mathbb{Q}$. We also know that $\max(a + e, b + e) \leq c$, since both $a + e \leq c$ and $b + e \leq c$. Hence $\max(a, b) + e \leq c$, implying that $\max(a, b) < c$. □

The next lemma can be seen as the reverse of the previous lemma, although we need the truncate part of the statement since we use the density of $\mathbb{Q}$. Gilbert shows this in `Rlt_join_either`.

Lemma 5.62 (✓). *Let $a, b, c : \mathcal{E}$. If $a < \max(b, c)$ then merely $a < b$ or $a < c$.*

Proof. Let $a, b, c : \mathcal{E}$ be such that $a < \max(b, c)$. By Lemma 5.50 applied to the Eudoxus reals and after stripping the truncation, we know that $a < b$ or $b < \max(b, c)$. In the first case we are done and in the second case we apply cotransitivity once again to $a < \max(b, c)$. Then we know that $a < c$ or $c < \max(b, c)$. Again, we are done in the first case. In the second case it follows by Lemma 5.61 that $\max(b, c) < \max(b, c)$, which is a contradiction. □

The next lemma intuitively states that if the distance between two points is strictly larger than zero that the two points are apart. Gilbert's fitting name for this lemma is `metric_to_apart`.

Lemma 5.63 (✓). *Let $x, y : \mathcal{E}$. If $0 < |x - y|$, then $x \# y$.*

Proof. Let $x, y : \mathcal{E}$ be such that $0 < |x - y|$. Then by Lemma 5.56 it holds that $0 < \max(x - y, y - x)$ and so by Lemma 5.62 we have $0 < x - y$ or $0 < y - x$. Here we stripped the truncations since the goal is a mere proposition. Then it follows that either $y < x$ or $x < y$, which both imply $x \# y$. □

Then using all the previous lemmas we can first prove the strong binary extensionality in the case where two Eudoxus reals are equal. Then general form of the strong binary extensionality follows as a corollary. Gilbert proves this in respectively `Rmult_lt_apart` and `real_full_pseudo_srorder`.

Lemma 5.64 (✓). *Let $a, b, c : \mathcal{E}$. If $ca \# cb$ then $a \# b$.*

Proof. Let $a, b, c : \mathcal{E}$ be such that $ca \# cb$. Then without loss of generality we will assume that $ca < cb$. This implies that $0 < c(b - a)$ and hence $0 < |c||b - a|$ by Lemma 5.36. Then by Lemma 5.60 we have $0 < |b - a|$ and hence by Lemma 5.63, we have $a \# b$. □

Corollary 5.65 (✓). *Let $a, b, c, d : \mathcal{E}$. If $ab \# cd$ then merely $a \# c$ or $b \# d$.*

Proof. Since multiplication is commutative on the type of Eudoxus reals, it suffices to show this in the case that $a = c$[6]. This is exactly Lemma 5.64. □

[6]Shown in the HoTT library in `Classes/theory/apartness` in lemma `strong_binary_setoid_morphism_commutative`.

The strong binary extensionality of the multiplication was the last requirement for an ordered field we had to show. Hence we can conclude that the Eudoxus reals form an ordered field.

Theorem 5.66 (✓). *The Eudoxus reals together with constants* 0, 1, *operations* $+$, $-$, $\cdot$, $/$, min, max *and mere relations* $\leq$, $<$, $\#$ *is an ordered field.*

5.12 Rationals

We end this section on the Eudoxus reals with a discussion on our definition of the rationals. We constructed the rational numbers as part of the Eudoxus reals with the following definition.

Definition 5.67 (✗). *Define the subset* $\mathbb{Q}$ *of the Eudoxus reals as*

$$\mathbb{Q} :\equiv \sum_{a:\mathcal{E}} \left\| \sum_{\lambda:\mathcal{Q}} \mathsf{classof}\ \lambda = a \right\|.$$

To rightfully denote this type by $\mathbb{Q}$ and call it the type of rational numbers we should check whether it is indeed a construction of the rational numbers. We will not prove this here, but instead we will explain why we believe this result to be true.

Operations as addition and multiplication are inherited from the Eudoxus reals, just as the proofs that these operations for instance commute. However, we also need to establish that the type of rational numbers is closed under these operations. To do so, we can define variants of these operations for rational slopes under which the type of rational slopes is closed. Therefore we identify a rational slope with an element of $\mathbb{Z} \times \mathbb{N}_+$ and then use the 'standard' computation rules for rational numbers. We then end up with the following operations for rational slopes. Note that these orderings for rational slopes are decidable in contrast to the general case of slopes, since the orderings on $\mathbb{Z}$ are decidable.

Definition 5.68 (✗). *Identify* $p, q : \mathcal{Q}$ *with respectively* $(p_1, p_2) : \mathbb{Z} \times \mathbb{N}_+$ *and* $q = (q_1, q_2) : \mathbb{Z} \times \mathbb{N}_+$. *Then we define*

- $p +_{\mathcal{Q}} q :\equiv (p_1 q_2 + q_1 p_2, p_2 q_2)$,
- $p \cdot_{\mathcal{Q}} q :\equiv (p_1 q_1, p_2 q_2)$,
- $(p <_{\mathcal{Q}} q) :\equiv (p_1 q_2 < q_1 p_2)$,
- $(p \leq_{\mathcal{Q}} q) :\equiv (p_1 q_2 \leq q_1 p_2)$,
- $-_{\mathcal{Q}} p :\equiv (-p_1, p_2)$,
- *and if* $p \neq 0$ *then we define* $1/_{\mathcal{Q}} p :\equiv \begin{cases} (p_2, p_1), & \text{if } 0 < p_1 \\ (-p_2, -p_1), & \text{if } p_1 < 0. \end{cases}$

Then we can show that these operations agree, up to equivalence, with the operations we defined in earlier sections for general slopes. The proofs for all these operations are very similar and hence we show just two examples, of which one is a special case of multiplication with two rational slopes.

Lemma 5.69 (✗). *Let* $p, q : \mathcal{Q}$, *then* $(p +_{\mathcal{S}} q) \sim (p +_{\mathcal{Q}} q)$, *where on the left-hand side we have addition in* $\mathcal{S}$ *and on the right-hand side in* $\mathcal{Q}$.

Proof. We can assume $p, q > 0$ and write $p = (p_1, p_2)$ and $q = (q_1, q_2)$ with $p_1, p_2, q_1, q_2 > 0$, since the non-positive cases can be proven similarly. Then for all $n : \mathbb{N}_+$ we can bound $(p +_{\mathcal{S}} q)(n)$ from below by $(p +_{\mathcal{Q}} q)(n)$ since

$$\begin{aligned}
(p +_{\mathcal{S}} q)(n) &\equiv p(n) + q(n) \\
&\equiv \min\{k : \mathbb{N} | p_2 k \geq p_1 n\} + \min\{k : \mathbb{N} | q_2 k \geq q_1 n\} \\
&= \min\{k : \mathbb{N} | p_2 q_2 k \geq p_1 q_2 n\} + \min\{k : \mathbb{N} | p_2 q_2 k \geq q_1 p_2 n\} \\
&\geq \min\{k : \mathbb{N} | p_2 q_2 k \geq (p_1 q_2 + q_1 p_2) n\} \\
&\equiv (p +_{\mathcal{Q}} q)(n).
\end{aligned}$$

Similarly $(p +_{\mathcal{S}} q)(n) - 1$ is a lower bound of $(p +_{\mathcal{Q}} q)(n)$ since

$$\begin{aligned}(p +_{\mathcal{S}} q)(n) - 1 &\equiv p(n) + q(n) - 1\\ &\equiv \min\{k : \mathbb{N} | p_2 k \geq p_1 n\} + \min\{k : \mathbb{N} | q_2 k \geq q_1 n\} - 1\\ &= \min\{k : \mathbb{N} | p_2 q_2 k \geq p_1 q_2 n\} + \min\{k : \mathbb{N} | p_2 q_2 k \geq q_1 p_2 n\} - 1\\ &\leq \min\{k : \mathbb{N} | p_2 q_2 k \geq (p_1 q_2 + q_1 p_2) n\}\\ &\equiv (p +_{\mathcal{Q}} q)(n).\end{aligned}$$

Hence for all $n : \mathbb{N}_+$ we have $(p +_{\mathcal{S}} q)(n) - 1 \leq (p +_{\mathcal{Q}} q)(n) \leq (p +_{\mathcal{S}} q)(n)$, implying $(p +_{\mathcal{S}} q) \sim (p +_{\mathcal{Q}} q)$ since the slopes are odd. □

Lemma 5.70 (✗)**.** *Let $\varepsilon : \mathbb{Q}$ and $c : \mathbb{Z}$, then $(c \cdot_{\mathcal{S}} \varepsilon) \sim (c \cdot_{\mathcal{Q}} \varepsilon)$, where on the left-hand side we have multiplication in $\mathcal{S}$ and on the right-hand side in $\mathcal{Q}$.*

Proof. We will assume $c > 0$, since the case $c < 0$ is similar and the case $c = 0$ is trivial. For similar reasons we will assume $\varepsilon > 0$ and write $\varepsilon = (\varepsilon_1, \varepsilon_2)$ for $\varepsilon_1, \varepsilon_2 : \mathbb{N}_+$. Take $n : \mathbb{N}_+$, then we can bound $(c \cdot_{\mathbb{Q}} \varepsilon)(n)$ from above by $(c \cdot_{\mathcal{S}} \varepsilon)(n)$ since

$$\begin{aligned}(c \cdot_{\mathbb{Q}} \varepsilon)(n) &= \min\{k : \mathbb{N} | \varepsilon_2 k \geq c\varepsilon_1 n\}\\ &\leq c \min\{k : \mathbb{N} | \varepsilon_2 k \geq \varepsilon_1 n\}\\ &= c(\varepsilon(n))\\ &= (c \cdot_{\mathcal{S}} \varepsilon)(n).\end{aligned}$$

Next we prove $(c \cdot_{\mathcal{S}} \varepsilon)(n) - c \leq (c \cdot_{\mathcal{Q}} \varepsilon)(n)$ for all $n : \mathbb{N}$ and $c > 0$ by induction on c. If $c = 1$, we have an equality and we are done. Now assume as induction hypothesis (IH) that there exists $c : \mathbb{N}$ such that $(c \cdot_{\mathcal{S}} \varepsilon)(n) - c \leq (c \cdot_{\mathbb{Q}} \varepsilon)(n)$ for all $n : \mathbb{N}$. Then for $n : \mathbb{N}_+$ we have

$$\begin{aligned}((c+1) \cdot_{\mathcal{S}} \varepsilon)(n) - (c+1) &= (c+1) \min\{k : \mathbb{N} | \varepsilon_2 k \geq \varepsilon_1 n\} - (c+1)\\ &= (c \min\{k : \mathbb{N} | \varepsilon_2 k \geq \varepsilon_1 n\} - c) + \min\{k : \mathbb{N} | \varepsilon_2 k \geq \varepsilon_1 n\} - 1\\ &\overset{\text{IH}}{\leq} \min\{k : \mathbb{N} | \varepsilon_2 k \geq c\varepsilon_1 n\} + \min\{k : \mathbb{N} | \varepsilon_2 k \geq \varepsilon_1 n\} - 1\\ &\leq \min\{k : \mathbb{N} | \varepsilon_2 k \geq (c+1)\varepsilon_1 n\}\\ &= ((c+1) \cdot_{\mathcal{Q}} \varepsilon)(n).\end{aligned}$$

Hence for all $c, n : \mathbb{N}_+$ we have $(c \cdot_{\mathcal{S}} \varepsilon)(n) - c \leq (c \cdot_{\mathbb{Q}} \varepsilon)(n) \leq (c \cdot_{\mathcal{S}} \varepsilon)(n)$. Therefore we conclude that $c \cdot_{\mathcal{S}} \varepsilon \sim c \cdot_{\mathcal{Q}} \varepsilon$ since the functions are odd. □

After showing that all the operations from Definition 5.68 agree up to equivalence with the usual operations on slopes, we can conclude that the type of rational numbers $\mathbb{Q}$ is closed under these operations.

It then remains to show that it is the initial field with characteristic 0. We could try to show this directly, but easier is to conclude that our rationals are equivalent to the rationals as defined in the HoTT book. The HoTT book namely defines the rationals as $\mathbb{Z} \times \mathbb{N}/\approx$, where $(u, a) \approx (v, b) :\equiv (u(b+1) = v(a+1))$, which coincides with our identification of rational slopes with $\mathbb{Z} \times \mathbb{N}_+$ if we shift by 1.

We believe that it is possible to work out the details here and thereby prove that the type we denote by $\mathbb{Q}$ is indeed a construction of the rationals. However, due to time constraints we leave it at this explanation.

6 Completeness

For the Eudoxus reals to be a construction of the real numbers, it remains to show that the type is either Dedekind or Cauchy complete. A'Campo and Mizrahi show in the classical setting that the Eudoxus reals are Dedekind complete, but we are not able to replicate the proof in our constructive setting. Hence we show that the type of slopes is Cauchy complete, since in our constructive setting Dedekind and Cauchy completeness are not equivalent [41, p. 419]. We will discuss however that we need the axiom of choice to transfer this result to the Eudoxus reals.

6.1 Dedekind completeness

A'Campo shows in his paper that the Eudoxus reals are Dedekind complete using the following definition, which is also known as the least-upper-bound-property.

Definition 6.1 (Dedekind completeness)**.** *A partially ordered set S is Dedekind complete if every non-empty subset $A \subset S$ which has an upper bound, also has a least upper bound.*

In his proof A'Campo defines the least upper bound of a bounded set of reals, by taking the pointwise maximum of the representative slopes of the set. We will try to follow this idea in our setting of homotopy type theory.

Therefore, let $P : \mathcal{E} \to \mathsf{Prop}$ be a type family and define $D :\equiv \sum_{a:\mathcal{E}} P(a)$. Let m be an upper bound of D, so that for all $(a, p) : D$ we have $a \leq m$. Then we want to construct the least upper bound of this set D. At this point, A'Campo defines Δ to be a set of well-adjusted slopes representing the Eudoxus reals in D. However, this is a definition of Δ which requires the axiom of choice. Mizrahi avoids this axiom by defining Δ as the set of *all* well-adjusted slopes representing the Eudoxus reals in D. This means we can define Δ as follows, where we truncate the existence of $d : D$ to make sure that a slope $\delta : \mathcal{S}$ is associated with at most one element of Δ.

$$\Delta :\equiv \sum_{\delta:\mathcal{S}} \left(\delta \text{ is well-adjusted } \times \left\| \sum_{d:D} \mathsf{classof}\, \delta = d \right\| \right).$$

Since the least upper bound is unique if it exists, we can strip the truncation to conclude that there exists a slope $\mu : \mathcal{S}$ such that $\mathsf{classof}\, \mu = m$. Then by definition of the ordering, we know that for all $\delta : \Delta$ we have $\delta \leq \mu$. Hence by Lemma 5.20 and Lemma 5.29 we see that for all $\delta : \Delta$ and $n : \mathbb{N}_+$ we have $\delta(n) \leq \mu(n) + 2$. This implies that for all $n : \mathbb{N}_+$ the integer $\mu(n) + 2$ is an upper bound of the subset $S(n) :\equiv \sum_{z:\mathbb{Z}} \| \sum_{\delta:\Delta} (\delta(n) = z) \|$. Once again we truncate the existence of $\delta : \Delta$ to define $S(n)$ as a subset instead of just a subtype.

Both A'Campo and Mizrahi now take the maximum of the set $S(n)$. They prove that the function σ defined as the odd extension of $n \mapsto \max(S(n))$ is a slope and that its class is the least upper bound of the set D. They can do so because in classical logic, every bounded subset of the integers has a least upper bound. However, in our constructive setting this result does not hold.

The problem is that constructing the maximum of $S(n)$ requires the law of excluded middle. If we namely want to construct the maximum we need to be able to check whether an integer is an element of $S(n)$. The maximum in particular namely needs to be part of $S(n)$, otherwise we can find a smaller lower bound by subtracting 1 which constitutes a contradiction. Hence given $z : \mathbb{Z}$ we need to be able to decide whether there merely exists a slope $\delta : \Delta$ such that $\delta(n) = z$. This in turn entails that we should be able to show for this δ that it is well-adjusted and that there merely exists $d : D$ such that $\mathsf{classof}\, \delta = d$. The type family P which defines D is however in general not decidable, and therefore the type family $z \mapsto \sum_{\delta:\Delta} (\delta(n) = z)$ is not decidable. This means that in general we cannot show that z is indeed an element of $S(n)$ and hence we cannot construct the maximum.

If we would have assumed the law of excluded middle, we would have been able to conclude for any $z : \mathbb{Z}$ whether it was part of the set $S(n)$. Then we could have constructed the maximum of $S(n)$ by subtracting 1 from the upper bound $\mu(n) + 2$ until we would encounter an element in $S(n)$, which would then be the maximum of $S(n)$. After this step, we could then continue in the same line as Mizrahi and A'Campo.

We do not expect that we are able to avoid having to construct a maximum for a subset of the integers in order to prove that the Eudoxus reals are Dedekind complete. Even when we are working with a more constructive formulation of the least-upper-bound-property such as for instance defined by Bridges [12], we expect this to remain problematic.

Hence in next sections we will discuss whether slopes and the Eudoxus reals are Cauchy complete.

6.2 Slopes are Cauchy complete

In this section we will first show that the type of slopes is Cauchy complete, after which we will discuss in Section 6.3 why this result does not transfer to the Eudoxus reals.

We will assume in this section, without loss of generality, that all well-adjusted slopes, and hence in particular rational slopes, are slopes with bound 1. We can do so because slopes based on the same function $\mathbb{Z} \to \mathbb{Z}$ but with a different bound are equivalent and the orderings respect equivalences. This will simplify parts of the proofs.

The first step in showing that slopes are Cauchy complete is to show that if we can bound the difference between two slopes by a rational slope, that this implies a pointwise inequality.

Lemma 6.2. *Let $\lambda, \mu : \mathcal{S}$ be slopes and let $\varepsilon : \mathcal{Q}_+$. Define $N :\equiv \frac{1}{\varepsilon}(d_\lambda + d_\mu + 1)$. If $\mu < \lambda + \varepsilon$, then for all $n : \mathbb{N}$ such that $n \geq N$ we have $\mu(n) \leq \lambda(n) + 2 \cdot \varepsilon(n)$.*

Proof. Since $0 < \lambda - \mu + \varepsilon$ there merely exists $a : \mathbb{N}_+$ such that $\lambda(a) - \mu(a) + \varepsilon(a) > d_{\lambda-\mu+\varepsilon}$. We will use this to prove that $\lambda - \mu + \varepsilon$ is pointwise bounded from below, which is a mere proposition. Hence we can apply the induction principle for truncations and forget about the merely. Then by [35, Pro. 4.50] we have for all $n : \mathbb{N}$ that

$$-d_{\lambda-\mu+\varepsilon} + n : a \leq \lambda(n) - \mu(n) + \varepsilon(n).$$

Here we can bound the left-hand side by the bounds of the separate slopes, since $d_{\lambda-\mu+\varepsilon} \leq d_\lambda + d_\mu + d_\varepsilon$ [35, Pro. 4.48]. Hence we have for all $n : \mathbb{N}_+$ that

$$-d_\lambda - d_\mu - 1 \leq \lambda(n) - \mu(n) + \varepsilon(n).$$

Now that we have bounded these terms from below, we will check that for all $n : \mathbb{N}$ such that $n \geq N$ we have $\varepsilon(n) \geq d_\lambda + d_\mu + 1$. Recall that $\frac{1}{\varepsilon}$ for $n : \mathbb{N}_+$ is defined as

$$\frac{1}{\varepsilon}(n) = \min\{k : \mathbb{N} | \varepsilon_1 k \geq \varepsilon_2 n\},$$

where $\varepsilon = (\varepsilon_1, \varepsilon_2)$. It follows that

$$\varepsilon_1 \cdot \left(\frac{1}{\varepsilon}(d_\lambda + d_\mu + 1)\right) \geq (d_\lambda + d_\mu + 1)\varepsilon_2$$

and so for all $n : \mathbb{N}$ such that $n \geq N$ we have

$$\begin{aligned}
\varepsilon(n) &\geq \varepsilon(N) \\
&= \min\left\{k : \mathbb{N} | \varepsilon_2 k \geq \varepsilon_1 \left(\frac{1}{\varepsilon}(d_\lambda + d_\mu + 1)\right)\right\} \\
&\geq \min\{k : \mathbb{N} | \varepsilon_2 k \geq (d_\lambda + d_\mu + 1)\varepsilon_2\} \\
&= d_\lambda + d_\mu + 1.
\end{aligned}$$

Hence we are finished since for all $n : \mathbb{N}$ such that $n \geq N$ we see that

$$\lambda(n) - \mu(n) + 2 \cdot \varepsilon(n) = (\lambda(n) - \mu(n) + \varepsilon(n)) + \varepsilon(n) \geq (-d_\lambda - d_\mu - 1) + (d_\lambda + d_\mu + 1) = 0.$$

□

To prove the completeness we will use the notion of a rapid Cauchy sequence.

Definition 6.3. *Let $(\lambda_n)_{n:\mathbb{N}}$ be a sequence of slopes in $\mathcal{S}$. Then the sequence $(\lambda_n)_{n:\mathbb{N}}$ is rapid Cauchy if for all $k : \mathbb{N}$ it holds that $|\lambda_k - \lambda_{k+1}| < 2^{-k}$.*

These rapid Cauchy sequences are easier to work with since the difference between two subsequent elements of the sequence is uniformly bounded. Given a Cauchy sequence we can always find a rapid Cauchy subsequence to work with.

Lemma 6.4. *Let $(\lambda_n)_{n:\mathbb{N}}$ be a Cauchy sequence of slopes. Then there exists a subsequence $(\lambda_{nk})_{k:\mathbb{N}}$ such that $(\lambda_{nk})_{k:\mathbb{N}}$ is rapid Cauchy.*

Proof. Let $(\lambda_n)_{n:\mathbb{N}}$ be a Cauchy sequence. Then there exists a function $N : \mathcal{Q}_+ \to \mathbb{N}$ such that for all $\varepsilon : \mathcal{Q}_+$ and $m, n : \mathbb{N}$ satisfying $m, n \geq N(\varepsilon)$ we have $|\lambda_m - \lambda_n| < \varepsilon$.

We construct the subsequence by recursion on $\mathbb{N}$. Hence define $n_0 :\equiv N(1)$ and, assuming that we have defined n_k, define $n_{k+1} :\equiv \max\{N(2^{-(k+1)}), n_k\}$. To see that this subsequence is rapid Cauchy, take $k : \mathbb{N}$. Then we have $n_{k+1} \geq n_k \geq N(2^{-k})$, which implies that $|\lambda_{n_k} - \lambda_{n_{k+1}}| < 2^{-k}$. □

For these rapid Cauchy sequences we can construct the limit as the diagonal sequence. However, before showing that this is indeed the limit, we first show it is a slope.

Lemma 6.5. *Let $(\lambda_n)_{n:\mathbb{N}}$ be a rapid Cauchy sequence of well-adjusted slopes. Define the function $\lambda : \mathbb{Z} \to \mathbb{Z}$ as the odd extension of $\lambda(n) :\equiv \lambda_n(n)$ for $n > 0$. Then λ is a slope with bound $d_\lambda :\equiv 45$.*

Proof. Since we define λ as an odd extension we only have to bound $\lambda(m+n)-\lambda(m)-\lambda(n)$ for $n, m > 0$ [35, Prop. 4.26]. Hence take $m, n : \mathbb{N}$ such that $n, m > 0$. Then a first bound on this difference is

$$\begin{aligned}|\lambda(m+n) - \lambda(m) - \lambda(n)| &= |\lambda_{m+n}(m+n) - \lambda_m(m) - \lambda_n(n)| \\ &\leq d_{\lambda_{m+n}} + |\lambda_{m+n}(m) + \lambda_{m+n}(n) - \lambda_m(m) - \lambda_n(n)| \\ &\leq 1 + |\lambda_{m+n}(m) - \lambda_m(m)| + |\lambda_{m+n}(n) - \lambda_n(n)|.\end{aligned}$$

Now we will bound the term $|\lambda_{m+n}(m)-\lambda_m(m)|$ and note the last term can be bound analogously. Since $(\lambda_n)_{n:\mathbb{N}}$ is a rapid Cauchy sequence we have a bound for the difference of subsequent elements, but we can also bound the difference between slopes further apart:

$$\begin{aligned}|\lambda_{m+n} - \lambda_m| &\leq |\lambda_{m+n} - \lambda_{m+n-1}| + \cdots + |\lambda_{m+1} - \lambda_m| \\ &< \sum_{k=m}^{m+n-1} \frac{1}{2^k} \\ &\sim \frac{2^n - 1}{2^{m+n-1}}.\end{aligned} \tag{6}$$

Then by Lemma 6.2 we know that for $A :\equiv \frac{2^{m+n-1}}{2^n-1}(3)$ and all $a : \mathbb{N}$ such that $a \geq A$ we have

$$|\lambda_{m+n}(a) - \lambda_m(a)| \leq 2 \cdot \frac{2^n - 1}{2^{m+n-1}}(a). \tag{7}$$

We can use this to bound $|\lambda_{m+n}(m)-\lambda_m(m)|$. We first bound the function evaluations of λ_{m+n} and λ_m in m to their respective evaluations in A and $m + A$. Then we are able to apply the inequality from (7), after which we bound the function evaluation in $m + A$ by evaluations in A and m.

$$\begin{aligned}|\lambda_{m+n}(m) - \lambda_m(m)| &\leq d_{\lambda_{m+n}} + d_{\lambda_m} + |\lambda_{m+n}(m+A) - \lambda_{m+n}(A) - (\lambda_m(m+A) - \lambda_m(A))| \\ &\leq 2 + |\lambda_{m+n}(m+A) - \lambda_m(m+A)| + |\lambda_{m+n}(A) - \lambda_m(A))| \\ &\leq 2 + 2 \cdot \frac{2^n - 1}{2^{m+n-1}}(m+A) + 2 \cdot \frac{2^n - 1}{2^{m+n-1}}(A) \\ &\leq 2 + 2 \cdot d_{\frac{2^n-1}{2^{m+n-1}}} + 2 \cdot \frac{2^n - 1}{2^{m+n-1}}(m) + 4 \cdot \frac{2^n - 1}{2^{m+n-1}}(A) \\ &\leq 4 + 2 \cdot \frac{2^n - 1}{2^{m+n-1}}(m) + 4 \cdot \frac{2^n - 1}{2^{m+n-1}}(A).\end{aligned}$$

Hence we have two terms left to bound. The first we can bound as

$$\begin{aligned}\frac{2^n - 1}{2^{m+n-1}}(m) &= \min\{k : \mathbb{N} | 2^{m+n-1}k \geq (2^n - 1)m\} \\ &\leq \min\{k : \mathbb{N} | 2^{m+n-1}k \geq 2^n m\} \\ &= \min\{k : \mathbb{N} | 2^{m-1}k \geq m\} \\ &\leq 1.\end{aligned}$$

For the second term note that $(2^n-1)(A-1) < 3 \cdot 2^{m+n-1}$. Hence we can bound this term as

$$\begin{aligned}\frac{2^n-1}{2^{m+n-1}}(A) &\equiv \min\{k : \mathbb{N} | 2^{m+n-1}k \geq (2^n-1)A\} \\ &= \min\{k : \mathbb{N} | 2^{m+n-1}k - (2^n-1) \geq (2^n-1)(A-1)\} \\ &\leq \min\{k : \mathbb{N} | 2^{m+n-1}k - (2^n-1) \geq 3 \cdot 2^{m+n-1}\} \\ &\leq \min\{k : \mathbb{N} | 2^{m+n-1}k - 2^n \geq 3 \cdot 2^{m+n-1}\} \\ &= \min\{k : \mathbb{N} | 2^{m-1}k - 1 \geq 3 \cdot 2^{m-1}\} \\ &\leq 4.\end{aligned}$$

Therefore we can bound $|\lambda_{m+n}(m) - \lambda_m(m)|$ by $4 + 2 \cdot 1 + 4 \cdot 4 = 22$, and so λ is a slope with bound

$$|\lambda(m+n) - \lambda(m) - \lambda(n)| \leq 1 + 22 + 22 = 45.$$

□

Now we prove that the slope we constructed in the previous lemma is actually a limit of the given sequence.

Lemma 6.6. *Let $(\lambda_n)_{n:\mathbb{N}}$ be a rapid Cauchy sequence of well-adjusted slopes and define $\lambda : \mathcal{S}$ as in Lemma 6.5. Then λ is a limit of $(\lambda_n)_{n:\mathbb{N}}$.*

Proof. We have to prove that for all $\varepsilon : \mathcal{Q}_+$ there exists $N > 0$ such that for all $n \geq N$ we have $|\lambda - \lambda_n| < \varepsilon$. Take $\varepsilon : \mathcal{Q}_+$ and take N such that $\frac{1}{2^{N-1}} < \frac{\varepsilon}{4}$. Then we claim that for all $n \geq N$ we have $|\lambda - \lambda_n| < \varepsilon$. By Lemma 5.20, it suffices to prove that for all $n \geq N$ there merely exists $k : \mathbb{N}_+$ such that

$$\varepsilon(k) - |\lambda(k) - \lambda_n(k)| > d_\varepsilon + d_{|\lambda - \lambda_n|} = 1 + d_{|\lambda - \lambda_n|}.$$

By a similar reasoning as (6) in the proof of Lemma 6.5, we have for all $m \geq l$ that $|\lambda_m - \lambda_l| < \frac{2^m - 2^l}{2^{m+l-1}}$. Hence if $m \geq l \geq N$ it holds that $|\lambda_m - \lambda_l| < \frac{\varepsilon}{4}$, to which we can apply Lemma 6.2. This tells us that for all $a \geq A :\equiv \frac{4}{\varepsilon}(3)$, we have $|\lambda_m(a) - \lambda_l(a)| < 2 \cdot \frac{\varepsilon}{4}(a)$.

In order to construct the k we need, we first use this bound to estimate $\varepsilon(k) - |\lambda(k) - \lambda_n(k)|$ for all $n \geq N$ and $k > 2A$. Hence we first bound $\lambda(k)$ and $\lambda_n(k)$ by their function evaluations in $k + A$ and A

$$\begin{aligned}\varepsilon(k) - |\lambda(k) - \lambda_n(k)| &= \varepsilon(k) - |\lambda_k(k) - \lambda_n(k)| \\ &\geq \varepsilon(k) - d_{\lambda_k} - d_{\lambda_n} - |\lambda_k(k+A) - \lambda_n(k+A)| - |\lambda_k(A) - \lambda_n(A)| \\ &> \varepsilon(k) - 2 - 2 \cdot \frac{\varepsilon}{4}(k+A) - 2 \cdot \frac{\varepsilon}{4}(A).\end{aligned}$$

Then to relate all the function evaluations of ε, we bound $\varepsilon(k)$ by $\varepsilon(k+A)$ and $\varepsilon(A)$, and use that $2 \cdot (\varepsilon/4) \sim \varepsilon/2$ with a pointwise difference of 1

$$... \geq (\varepsilon(k+A) - \varepsilon(A) - d_\varepsilon) - 2 - \left(\frac{\varepsilon}{2}(k+A) + 1\right) - \left(\frac{\varepsilon}{2}(A) + 1\right).$$

Next we use that $\varepsilon - \varepsilon/2 \sim \varepsilon/2$ and $\varepsilon + \varepsilon/2 \sim 3\varepsilon/2$ with a pointwise difference of 1, to gather the function evaluations in $k + A$ and A

$$... \geq \left(\frac{\varepsilon}{2}(k+A) - 1\right) - \left(\frac{3\varepsilon}{2}(A) + 1\right) - 5.$$

Since $k > 2A$ there exists $l > 0$ such that $k = 2A + l$, so we can bound the evaluation of $\varepsilon/2$ in $k + A$ by the evaluation in l and $3A$

$$... \geq \left(\frac{\varepsilon}{2}(l) + \frac{\varepsilon}{2}(3A) - d_{\frac{\varepsilon}{2}}\right) - \frac{3\varepsilon}{2}(A) - 7.$$

When we write out the definitions we see that $\frac{\varepsilon}{2}(3A) = \frac{3\varepsilon}{2}(A)$. Hence we end up with the following bound for our original left-hand side

$$\varepsilon(k) - |\lambda(k) - \lambda_n(k)| \geq \frac{\varepsilon}{2}(l) - 8.$$

Since $0 < \frac{\varepsilon}{2}$ there exists $B : \mathbb{N}_+$ such that $\frac{\varepsilon}{2}(B) > d_{\frac{\varepsilon}{2}} = 1$. We stripped the truncation here since we only need to prove the mere existence of k. Hence by [35, Pro. 4.50] we have for all $b : \mathbb{N}_+$ that

$$\frac{\varepsilon}{2}(b) \geq b : B - 1.$$

Therefore, if we choose $l :\equiv B(8+1+d_{|\lambda-\lambda_n|}+1)$ and $k :\equiv 2A+l$ we have

$$\varepsilon(k) - |\lambda(k) - \lambda_n(k)| > 1 + d_{|\lambda-\lambda_n|}.$$

□

Then by combining the previous lemmas a completeness result for well-adjusted slopes follows.

Corollary 6.7. *Let $(\lambda_n)_{n:\mathbb{N}}$ be a Cauchy sequence of well-adjusted slopes and define $\lambda : \mathcal{S}$ as in Lemma 6.5. Then λ is a limit of $(\lambda_n)_{n:\mathbb{N}}$.*

Proof. Let $(\lambda_n)_{n:\mathbb{N}}$ be a Cauchy sequence of well-adjusted slopes. Then by Lemma 6.4 there exists a rapid Cauchy subsequence $(\lambda_{n_k})_{n:\mathbb{N}}$. This subsequence converges by Lemma 6.6 to $\lambda : \mathcal{S}$ from Lemma 6.5. Then it only remains to show that the original sequence $(\lambda_n)_{n:\mathbb{N}}$ converges to λ. Let $\varepsilon : \mathcal{Q}_+$ and take $N : \mathbb{N}$ such that for all $n_k \geq N$ we have

$$|\lambda_{n_k} - \lambda| < \frac{\varepsilon}{2}$$

and such that for all $n, m \geq N$ we have

$$|\lambda_n - \lambda_m| < \frac{\varepsilon}{2}.$$

Take $n_k \geq N$, then by the triangle inequality for slopes from Lemma 5.37 we know for all $n \geq N$ that

$$|\lambda_n - \lambda| \leq |\lambda_n - \lambda_{n_k}| + |\lambda_{n_k} - \lambda| < \varepsilon.$$

Hence the sequence $(\lambda_n)_{n:\mathbb{N}}$ converges to λ. □

Using the Concentration Lemma, we can generalize the result of Corollary 6.7 to hold for Cauchy sequences of arbitrary slopes.

Theorem 6.8. *Let $(\lambda_n)_{n:\mathbb{N}}$ be a Cauchy sequence of slopes, then there exists $\lambda : \mathcal{S}$ such that λ is a limit of $(\lambda_n)_{n:\mathbb{N}}$.*

Proof. Using the concentration lemma (Lemma 5.10) and induction on $\mathbb{N}$, we first define a sequence $(\lambda'_n)_{n:\mathbb{N}}$ such that for all $n : \mathbb{N}$ the slope λ'_n is well-adjusted and equivalent to λ_n. This sequence $(\lambda'_n)_{n:\mathbb{N}}$ is also Cauchy since $|\lambda'_n - \lambda'_m| \sim |\lambda_n - \lambda_m|$ for all $n, m : \mathbb{N}$ by the proof of Definition 5.34. As a consequence of Corollary 6.7 there exists $\lambda : \mathcal{S}$ such that λ is a limit of $(\lambda'_n)_{n:\mathbb{N}}$. By Lemma 4 it follows that also $(\lambda_n)_{n:\mathbb{N}}$ converges to λ. □

6.2.1 Cauchy completeness with respect to Cauchy approximations

We have now proven that the type of slopes is Cauchy complete with respect to Cauchy sequences. However, both the HoTT library [7] and the HoTT book [41, Sec. 11.2.2] define Cauchy complete using Cauchy approximations of which the definition is as follows.

Definition 6.9. *A function $x : Q_+ \to \mathcal{S}$ is a Cauchy approximation if for all $\varepsilon, \delta : \mathcal{Q}_+$ we have*

$$|x_\varepsilon - x_\delta| < \varepsilon + \delta.$$

An element $z : \mathcal{S}$ is the limit of a Cauchy approximation x if for all $\varepsilon, \delta : \mathcal{Q}_+$ we have

$$|x_\varepsilon - z| < \varepsilon + \delta.$$

In such cases we will say that x is a Cauchy approximation of z.

Therefore we also want to conclude that the type of slopes is Cauchy complete with respect to Cauchy approximations. The HoTT book [41, p. 379] and Booij [11, p. 83] state that Cauchy sequences and approximations contain the same information, but do not give a full proof of this. Since we are also working with rational slopes instead of rational numbers, we prove here explicitly that these two notions of completeness are equivalent for slopes.

Lemma 6.10. *The type of slopes is Cauchy complete with respect to Cauchy sequences if and only if it is Cauchy complete with respect to Cauchy approximations.*

[7] In the file `Classes/theory/premetric.v`

Proof. This proof consists of three steps: firstly we show that we can construct a Cauchy sequence from a Cauchy approximation and vice versa, secondly we show that these constructions preserves limits and thirdly we show that the two versions of Cauchy completeness are equivalent.

Let $x : \mathcal{Q}_+ \to \mathcal{S}$ be a Cauchy approximation. Define the sequence $(y_n)_{n:\mathbb{N}}$ by $y_n :\equiv x_{1/(2n)}$ for all $n : \mathbb{N}$. To see that $(y_n)_{n:\mathbb{N}}$ is a Cauchy sequence, take $\varepsilon : \mathcal{Q}_+$ and take $N : \mathbb{N}$ such that $1/N \leq \varepsilon$. Then for all $n, m \geq N$ we have

$$|y_n - y_m| \equiv |x_{\frac{1}{2n}} - x_{\frac{1}{2m}}| < \frac{1}{2n} + \frac{1}{2m} \leq \frac{1}{2N} + \frac{1}{2N} \sim \frac{1}{N} \leq \varepsilon.$$

Similarly, given a sequence $(y_n)_{n:\mathbb{N}}$ we can also construct a Cauchy approximation $x : \mathcal{Q}_+ \to \mathcal{S}$. Let $N : \mathcal{Q} \to \mathbb{N}$ be the modulus of convergence, so that for all $\varepsilon : \mathcal{Q}_+$ and $n, m \geq N(\varepsilon)$ we have $|y_n - y_m| < \varepsilon$. Define $x : \mathcal{Q}_+ \to \mathcal{S}$ as $x_\varepsilon :\equiv y_{N(\varepsilon)}$ for all $\varepsilon : \mathcal{Q}_+$. Then x is a Cauchy approximation, since for all $\varepsilon, \delta : \mathcal{Q}_+$ we have

$$|x_\delta - x_\varepsilon| \equiv |y_{N(\delta)} - y_{N(\varepsilon)}| \leq |y_{N(\delta)} - y_{N(\delta)+N(\varepsilon)}| + |y_{N(\delta)+N(\varepsilon)} - y_{N(\varepsilon)}| < \delta + \varepsilon.$$

Then we claim that both of these constructions preserve limits. Hence we first prove that if $x : \mathcal{Q}_+ \to \mathcal{S}$ is a Cauchy approximation of $z : \mathcal{S}$ and $(y_n)_{n:\mathbb{N}}$ is the induced Cauchy sequence, then y converges to z. Take $\varepsilon : \mathcal{Q}_+$ and $N : \mathbb{N}$ such that $1/N < \varepsilon$, then for all $n \geq N$ we have

$$|y_n - z| \equiv |x_{\frac{1}{2n}} - z| < \frac{1}{2n} + \frac{1}{2n} \sim \frac{1}{n} \leq \frac{1}{N} < \varepsilon.$$

Similarly, if $(y_n)_{n:\mathbb{N}}$ is a Cauchy sequence converging to $z : \mathcal{S}$ and $x : \mathcal{Q}_+ \to \mathcal{S}$ the induced Cauchy approximation then we claim that also x converges to z. For this, let $M : \mathcal{Q}_+ \to \mathbb{N}$ be such that if $n \geq M(\varepsilon)$ then $|y_n - z| < \varepsilon$. Take $\varepsilon, \delta : \mathcal{Q}_+$, then

$$|x_\varepsilon - z| \equiv |y_{N(\varepsilon)} - z| \leq |y_{N(\varepsilon)} - y_{N(\varepsilon)+M(\delta)}| + |y_{N(\varepsilon)+M(\delta)} - z| < \varepsilon + \delta.$$

Now assume that the type is Cauchy complete with respect to sequences and let $x : \mathcal{Q}_+ \to \mathcal{S}$ be a Cauchy approximation. Let $(y_n)_{n:\mathbb{N}}$ be the induced Cauchy sequence. Then there exists a limit $z : \mathcal{S}$ of $(y_n)_{n:\mathbb{N}}$ and $(y_n)_{n:\mathbb{N}}$ induces a Cauchy approximation $x' : \mathcal{Q}_+ \to \mathcal{S}$ which also converges to z. It holds that $x'_\varepsilon \equiv x_{1/(2N(\varepsilon))}$, where $1/N(\varepsilon) \leq \varepsilon$ for all $\varepsilon : \mathcal{Q}_+$. Then x converges to z since for all $\varepsilon, \delta : \mathcal{Q}_+$ we have

$$|x_\varepsilon - z| \leq |x_\varepsilon - x'_{\delta/2}| + |x'_{\delta/2} - z| < |x_\varepsilon - x_{\frac{1}{2N(\delta/2)}}| + \frac{\delta}{2} + \frac{\delta}{4} < \varepsilon + \frac{1}{2N(\delta/2)} + \frac{3\delta}{4} \leq \varepsilon + \frac{\delta}{4} + \frac{3\delta}{4} \sim \varepsilon + \delta.$$

Therefore the type of slopes is Cauchy complete with respect to approximations.

Then assume that the type is Cauchy complete with respect to approximations. Let $(y_n)_{n:\mathbb{N}}$ be Cauchy sequence of slopes. Let $x : \mathcal{Q}_+ \to \mathcal{S}$ be the induced Cauchy approximation. Then there exists a limit $z : \mathcal{S}$ of x, and x induces a Cauchy sequence $(y'_n)_{n:\mathbb{N}}$ which also converges to z. Then it holds that $y'_n \equiv x_{1/(2n)} \equiv y_{N(1/(2n))}$, where N is the modulus of convergence of $(y_n)_{n:\mathbb{N}}$. Therefore the sequence $(y'_n)_{n:\mathbb{N}}$ is a subsequence of $(y_n)_{n:\mathbb{N}}$ and hence also $(y_n)_{n:\mathbb{N}}$ converges to z. □

Corollary 6.11. *Let $\mu : \mathcal{Q}_+ \to \mathcal{S}$ be a Cauchy approximation, then there exists $\lambda : \mathcal{S}$ such that λ is a limit of μ.*

6.3 Are the Eudoxus reals Cauchy complete?

Now that we have shown that the type of slopes is Cauchy complete, we would like to show that the type of Eudoxus reals is complete. Therefore we would have to show that every Cauchy sequence $(a_n)_{n:\mathbb{N}}$ of Eudoxus reals has a limit $a : \mathcal{E}$.

However, here we also run into some problems. Given a Cauchy sequence of real numbers $(a_n)_{n:\mathbb{N}}$, we would like to work with its representative Cauchy sequence of slopes $(\lambda_n)_{n:\mathbb{N}}$. From Section 6.2, we namely know that we can construct for such a Cauchy sequence a limit $\lambda : \mathcal{S}$. Then the limit of the sequence $(a_n)_{n:\mathbb{N}}$ would be $\mathsf{classof}\,\lambda$.

The induction principle would normally allow us to reduce a statement for $a : \mathcal{E}$ to a statement for $\lambda : \mathcal{S}$. Hence, at first glance, we could lift each Cauchy sequence in $\mathcal{E}$ to a Cauchy sequence in $\mathcal{S}$. However, we would have to apply the induction principle for each element in the Cauchy sequence individually, since the induction principle cannot deal with sequences as a whole. This would result in an infinite amount of applications of the induction principle and so this procedure is not constructive.

Another idea might be to use that the existence of a limit of a sequence is unique, which means that being Cauchy complete is a mere proposition. Hence we could strip the truncation in the surjectivity of $\mathsf{classof}$ and from this retrieve a representative sequence of slopes. The surjectivity states after all that for all $a : \mathcal{E}$ there merely exists $\lambda : \mathcal{S}$ such that $\mathsf{classof}\,\lambda = a$. However, we can only strip the truncation for each λ_n individually, which hence again results in infinitely many steps in our reasoning and a non-constructive proof.

To avoid these infinite procedures we need the axiom of choice from Definition 2.19. This allows us to move the truncation to the front of the statement, after which we could get rid of this truncation in just one step. Then this would give us a sequence $(\lambda_n)_{n:\mathbb{N}}$ such that for all $n : \mathbb{N}$ we have $\mathsf{classof}\,\lambda_n = a_n$. More precisely, we need the axiom of choice to conclude that

$$\left(\prod_{n:\mathbb{N}} \left\| \sum_{\lambda:\mathcal{S}} \mathsf{classof}\,\lambda = a_n \right\|\right) \to \left\| \sum_{\lambda:\mathbb{N}\to\mathcal{S}} \prod_{n:\mathbb{N}} \mathsf{classof}\,\lambda_n = a_n \right\|.$$

Since we are working with sequences we see that we only need the axiom of *countable* choice, which is the case where $X = \mathbb{N}$ in Definition 2.19.

Hence to conclude that the Eudoxus reals are Cauchy complete we need the axiom of choice. This agrees with the statement in the HoTT book [41, p. 381], which states that this is required to prove Cauchy completeness for every construction of which the last step is a quotient.

7 The completion of the Eudoxus reals

In Section 6 we discussed that we cannot show that the Eudoxus reals are Cauchy complete without assuming the axiom of choice. Hence in this section we will formulate a different type based on the Eudoxus reals, of which we can prove that it is a construction of the reals.

For this we follow the the construction of the Cauchy reals in the HoTT book [41, Ch. 11.3]. Here the completion of the type $\mathbb{Q}$ is defined using a higher inductive-inductive type, after which it is shown that the result is a construction of the real numbers and hence in particular Cauchy complete.

This completion is only defined for the rationals, but Gilbert has later generalized this construction to so-called premetric spaces [25]. He shows that the result is indeed Cauchy complete for premetric spaces and continues to show that the completion of the rationals is an Archimedean ordered field, similar to the HoTT book. Hence once we have shown that the Eudoxus reals form a premetric space, we can define the completion of the Eudoxus reals and conclude that this completion is Cauchy complete. In Section 7.1 we will state the definition of a premetric space, its completion and some other relevant concepts. In Section 7.2 we will then show that the Eudoxus reals are a premetric space.

We then have to show that the completion is an Archimedean ordered field. We cannot directly apply the work of Gilbert and the HoTT book to conclude this, since they only show this for the completion of the rational numbers. Once we have shown this, we want to compare the Eudoxus reals to the Cauchy reals. From the HoTT book we namely know that the Cauchy reals embed into every Cauchy complete Archimedean ordered field while preserving the rationals [41, Th. 11.3.50].

However, due to time constraints, we are not able to prove that the completion of the Eudoxus reals is an Archimedean ordered field *and* compare it to the Cauchy reals. Since we have shown that the type of Eudoxus reals is an Archimedean ordered field, we expect to also be able to show that its completion is an Archimedean ordered field. We expect that adapting the embedding of the Cauchy reals into any Cauchy complete Archimedean ordered field is harder. The construction in the HoTT book is namely based on extending the canonical embedding of the rationals into any ordered field, to the completion of the rationals. We do not have such a canonical embedding for the Eudoxus reals and hence we cannot easily adapt the proof to our case.

Hence in Section 7.3 we will explain, but not prove, why we think that the completion of the Eudoxus reals is indeed an Archimedean ordered field. Then in Section 7.4 we will construct an embedding from the completion to any Cauchy complete Archimedean ordered field which preserves the rationals. Using this result, we show that the completion of the Eudoxus reals is equivalent to, and by univalence equal to, the Cauchy reals. In a later project, it would be interesting to prove that the completion of the Eudoxus reals is indeed an Archimedean ordered field.

7.1 The completion of a premetric space

We start with the definition of a premetric space. The $\approx$-relation from the following definition is called a closeness relation, since $x \approx_\varepsilon y$ will mean that x and y are ε close. In other words, it will mean that the distance between x and y is strictly less than ε.

Definition 7.1. *A premetric space is a type A together with a mere relation $\_ \approx_\_ \_ : \mathbb{Q}_+ \to A \to A \to \mathsf{Prop}$ satisfying the following:*

1. *reflexivity: for all $\varepsilon : \mathbb{Q}_+$ and $x : A$ we have $x \approx_\varepsilon x$,*
2. *symmetry: for all $\varepsilon : \mathbb{Q}_+$ and $x, y : A$ such that $x \approx_\varepsilon y$ we have $y \approx_\varepsilon x$,*
3. *separatedness: for all $x, y : A$ such that for all $\varepsilon : \mathbb{Q}_+$ we have $x \approx_\varepsilon y$, it holds that $x = y$,*
4. *triangularity: for all $\varepsilon, \delta : \mathbb{Q}_+$ and $x, y, z : A$ such that $x \approx_\varepsilon y$ and $y \approx_\delta z$, we have $x \approx_{\varepsilon+\delta} z$,*
5. *roundedness: for all $\varepsilon : \mathbb{Q}_+$ and $x, y : A$ we have $x \approx_\varepsilon y$ if and only if there merely exists $\delta : \mathbb{Q}_+$ such that $\delta < \varepsilon$ and $x \approx_\delta y$.*

The product of two premetric spaces is again a premetric space, which will be useful to define functions with more than one argument[8].

Lemma 7.2. *Let $A, B : \mathcal{U}$ be premetric spaces. Then $A \times B$ is a premetric space if we endow it with the following closeness relation: for all $(a_1, b_1), (a_2, b_2) : A \times B$ and $\varepsilon : \mathbb{Q}_+$ define $(a_1, b_1) \approx_\varepsilon (a_2, b_2)$ if and only if $a_1 \approx_\varepsilon a_2$ and $b_1 \approx_\varepsilon b_2$.*

[8] This is proven in the HoTT library in `Classes/theory/premetric` as `prod_premetric`.

The completion of a premetric space, will not be the completion with respect to Cauchy sequences but with respect to Cauchy approximations. We already encountered both in Section 6.2.1 in the case of slopes, but we state here the more general definition of a Cauchy approximation and its limit for premetric spaces. In the following, being Cauchy complete will mean with respect to Cauchy approximations.

Definition 7.3. *Let $A : \mathcal{U}$ be a premetric space with closeness relation $\approx\colon \mathbb{Q}_+ \to A \to A \to \mathsf{Prop}$. A function $x : \mathbb{Q}_+ \to A$ is a Cauchy approximation, denoted $x : \mathsf{Approximation}\, A$, if for all $\varepsilon, \delta : \mathbb{Q}_+$ we have*

$$x_\varepsilon \approx_{\varepsilon+\delta} x_\delta.$$

An element $y : A$ is the limit of a Cauchy approximation x if for all $\varepsilon, \delta : \mathbb{Q}_+$ we have

$$x_\varepsilon \approx_{\varepsilon+\delta} y.$$

Now we are able to define the completion of a premetric space. It is defined as a higher inductive-inductive type, which means that it constructs a type and a type family simultaneously. This type family will turn out to be a closeness relation [25, Th. 3.16], so we will denote it by $\approx$.

The type itself contains two sorts of elements. Firstly, it contains an element for each point in the premetric space. Secondly, it also contains the limit of each Cauchy approximation. However, note that we do not compute the limit of a Cauchy approximations nor assume we know its value. We just state that the completion contains for all Cauchy approximations a point which we call the limit of the approximation. That this limit element is indeed a limit is encoded in the definition of the closeness relation. This relation is defined inductively in the second part of the definition. The closeness relation is defined to be a mere relation by the last path generator and is separated by the last path generator of the type $\mathcal{C}A$. Together these path generators ensure that the completion of a premetric space is a set [25, Lem. 3.6].

Definition 7.4. *Let $A : \mathcal{U}$ be a premetric space. Then we define the type $\mathcal{C}A : \mathcal{U}$ and the mere relation $\approx\colon \mathbb{Q}_+ \to \mathcal{C}A \to \mathcal{C}A \to \mathsf{Prop}$ as a higher inductive-inductive type. The type $\mathcal{C}A$, which we call the completion of A, is generated by*

- *a function $\eta : A \to \mathcal{C}A$,*
- *an element $\lim x : \mathcal{C}A$ for every Cauchy approximation $x : \mathbb{Q}_+ \to \mathcal{C}A$*
- *a path $x = y$ for all $x, y : \mathcal{C}A$ satisfying for all $\varepsilon : \mathbb{Q}_+$ that $x \approx_\varepsilon y$.*

The type family $\approx\colon \mathbb{Q}_+ \to \mathcal{C}A \to \mathcal{C}A \to \mathsf{Prop}$ is generated by the following constructors, where $a, b : A$, $u, v : \mathcal{C}A$, $\varepsilon, \delta, \gamma : \mathbb{Q}_+$ and $x, y : \mathsf{Approximation}(\mathcal{C}A)$,

- *if $a \approx_\varepsilon b$, then $\eta(a) \approx_\varepsilon \eta(b)$*
- *if $0 < \varepsilon - \delta$ and $\eta(a) \approx_{\varepsilon-\delta} y_\delta$, then $\eta(a) \approx_\varepsilon \lim y$*
- *if $0 < \varepsilon - \delta$ and $x_\delta \approx_{\varepsilon-\delta} \eta(b)$, then $\lim x \approx_\varepsilon \eta(b)$*
- *if $0 < \varepsilon - \delta - \gamma$ and $x_\delta \approx_{\varepsilon-\delta-\gamma} y_\gamma$, then $\lim x \approx_\varepsilon \lim y$*
- *if $p, q : u \approx_\varepsilon v$, then $p = q$.*

We can now perform induction on just $\mathcal{C}A$, just $\approx$ or on both simultaneously. We however refer to Gilbert [25, Def. 3.2-4] and the HoTT book [41, Sec. 11.3.2] for this. Especially the full induction principle is quite long, so if possible we try to avoid this.

Instead we make use of Lipschitz extensions, which are often easier to work with. We can define these, just as a notion of continuity, because the closeness relation of a premetric space captures the distance between terms.

Definition 7.5. *Let $A, B : \mathcal{U}$ be premetric spaces and $f : A \to B$ a function. Then f is Lipschitz if there exists $L : \mathbb{Q}_+$ such that for all $a_1, a_2 : A$ satisfying $a_1 \approx_\varepsilon a_2$, it holds that $f(a_1) \approx_{L\varepsilon} f(a_2)$.*

Definition 7.6. *Let $A, B : \mathcal{U}$ be two premetric spaces and $f : A \to B$ a function. Then f is continuous if for all $\varepsilon : \mathbb{Q}_+$ and $a_1 : A$, there merely exists $\delta : \mathbb{Q}_+$ such that for all $a_2 : A$ satisfying $a_1 \approx_\delta a_2$, it holds that $f(a_1) \approx_\varepsilon f(a_2)$.*

Just as in classical mathematics, a Lipschitz function is continuous. It also follows that if $A, B, C : \mathcal{U}$ are premetric spaces and $f : A \times B \to C$ is a function such that $a \mapsto f(a, b)$ is L-Lipschitz for all $b : B$, and $b \mapsto f(a, b)$ is K-Lipschitz for all $a : A$ that f itself is $(K + L)$-Lipschitz [25].

Lipschitz functions and especially the ability to extend them from a premetric space to its completion, let us easily define functions on the completion of a premetric space without having to explicitly use the induction principle. Gilbert proves this result for premetric spaces [25, Th. 3.20], but a proof for the rationals can also be found in the HoTT book [41, Lemma 11.3.15].

Lemma 7.7. *Let $A, B : \mathcal{U}$ be premetric spaces and let B be Cauchy complete. Let $f : A \to B$ be L-Lipschitz. Then there exists a L-Lipschitz function $\bar{f} : \mathcal{C}A \to B$ such that for all $x : A$ we have $\bar{f}(\eta x) = f(x)$. We refer to $\bar{f}$ as the Lipschitz extension of f.*

A continuous function on the completion of a premetric space is completely characterized by its value on non-limit elements. Gilbert proves this result for the unary case as stated below in his article [25, Th. 3.19]. He extends the lemma to functions of higher arity in `unique_continuous_binary_extension` and `unique_continuous_ternary_extension` in the file `base` of his formalization[9].

Lemma 7.8. *Let $A, B : \mathcal{U}$ be premetric spaces and let $f, g : \mathcal{C}A \to B$ be continous functions. If for all $a : A$ we have $f(\eta a) = g(\eta a)$, then for all $x : \mathcal{C}A$ we have $f(x) = g(x)$.*

7.2 The Eudoxus reals are a premetric space

Now we show that the Eudoxus reals are a premetric space.

An initial thought may however be that the type of slopes together with $\lambda \approx_\varepsilon \mu :\equiv |\lambda - \mu| < \varepsilon$ is already a premetric space. The first problem here is however that we have not defined $\mathbb{Q}$ as part of the slopes, but only the related $\mathcal{Q}$. If we ignore this for a moment and work with $\mathcal{Q}$ instead of $\mathbb{Q}$, the separatedness condition is still not satisfied. Define for instance $\lambda : \mathcal{S}$ by $\lambda(n) = 0$ for all $n : \mathbb{Z}$, and $\mu : \mathcal{S}$ by $\mu(0) = 1$ and $\mu(n) = 0$ for all $n : \mathbb{Z}$ such that $n \neq 0$. Then we have $|\lambda - \mu| = \mu \sim 0$. Since $<$ respects equivalences on slopes, this implies that $|\lambda - \mu| < \varepsilon$ for all $\varepsilon : \mathcal{Q}_+$. Hence, we see this type is not separated as λ and μ are not equal.

We can solve this problem by working with the Eudoxus reals and the relation $a \approx_\varepsilon b :\equiv |a - b| < \varepsilon$. Then in the above example we have $\mathsf{classof}\, \lambda \approx_\varepsilon \mathsf{classof}\, \mu$ for all $\varepsilon : \mathbb{Q}_+$ and indeed $\mathsf{classof}\, \lambda = \mathsf{classof}\, \mu$.

Lemma 7.9. *The type of the Eudoxus reals, $\mathcal{E}$, together with the mere relation $\approx : \mathbb{Q}_+ \to \mathcal{E} \to \mathcal{E} \to \mathsf{Prop}$ defined by $a \approx_\varepsilon b :\equiv (|a - b| < \varepsilon)$ for all $a, b : \mathcal{E}$ is a premetric space.*

Proof. We check al the requirements for a premetric space:

1. Let $a : \mathcal{E}$. By definition $a \approx_\varepsilon a = (|a - a| < \varepsilon)$. The claim then follows since $0 < \varepsilon$ for all $\varepsilon : \mathbb{Q}_+$.
2. This follows directly from $|a - b| = |b - a|$ for all $a, b : \mathcal{E}$.
3. Let $a, b : \mathcal{E}$ such that for all $\varepsilon : \mathbb{Q}_+$ we have $a \approx_\varepsilon b$, that is $|a - b| < \varepsilon$. Stripping the truncation in the surjectivity of $\mathsf{classof}$, we know that there exist slopes $\lambda, \mu : \mathcal{S}$ such that $\mathsf{classof}\, \lambda = a$ and $\mathsf{classof}\, \mu = b$. Hence for all $\varepsilon' : \mathcal{Q}_+$ we have $|\lambda - \mu| < \varepsilon'$. In particular it holds for all $m : \mathbb{N}$ that $|\lambda - \mu| < 1/m$. Since the sequence $(1/m)_{m:\mathbb{N}}$ is Cauchy and converges to 0 by Lemma 5.41, it follows by Lemma 5.40 that $|\lambda - \mu| \leq 0$. By definition we have $|\lambda - \mu| \geq 0$ and so by Lemma 5.32, we know $|\lambda - \mu| \sim 0$ implying that $\lambda \sim \mu$. Hence we have $a = b$.
4. Using the surjectivity of $\mathsf{classof}$, it suffices to prove for all $\varepsilon, \delta : \mathcal{Q}_+$ and $\lambda, \mu, \kappa : \mathcal{S}$ such that $|\kappa - \mu| < \varepsilon$ and $|\mu - \lambda| < \delta$ that $|\kappa - \lambda| < \varepsilon + \delta$. This follows directly from Lemma 5.26 and the triangle inequality in Lemma 5.37.
5. The left-to-right implication follows from the density of $\mathbb{Q}$ as proven in Lemma 5.46. For the right-to-left implication we use Lemma 5.23 to reduce the statement to slopes. Then the result follows from the transitivity of $<$ from Lemma 5.27.

□

[9] The formalization can be found at `https://github.com/SkySkimmer/HoTTClasses/tree/master/theories/cauchy_reals`.

7.3 Cauchy complete Archimedean ordered field

At this moment we can conclude that the completion of the Eudoxus reals is a Cauchy complete set and premetric space [25]. However, as stated at the beginning of this section, we do not have the time to fully prove that the completion is also an Archimedean ordered field, if we also want to compare it to the Cauchy reals from the HoTT book. Hence in this section, we will explain why we think that the completion of the Eudoxus reals is an Archimedean ordered field by explaining how we can define the relevant operations and prove their required properties. We will however not prove any of this.

Even though we do not take the completion of the rationals but of the Eudoxus reals, we can still follow the reasoning from the HoTT book [41, Sec. 11.3] and Gilbert [25]. Just as they do, we can define the addition, negation, join and meet using Lemma 7.7 as the Lipschitz extension of the respective functions on the Eudoxus reals. Since multiplication is not Lipschitz, we cannot use Lemma 7.7 to define it on the completion. We can however, follow either the reasoning of the HoTT book, which defines multiplication indirectly by extending squaring to the completion, or follow Gilbert by extending multiplication on bounded intervals. After both methods, we can then define the multiplicative inverse again as a Lipschitz extension.

It then remains to define the orderings on the completion, but we can again follow the HoTT book and Gilbert. The non-strict ordering can directly be defined using the join on the completion by stating $(x \leq y) :\equiv (\max(x, y) = y)$ for $x, y : \mathcal{CE}$. This extends the definition of the non-strict ordering for the Eudoxus reals, since this equality holds for all ordered fields propositionally as we will show in Lemma 7.16. The strict ordering can be defined as $(x < y) :\equiv \sum_{a,b:\mathcal{E}}(x \leq \eta(a)) \times (a < b) \times (\eta(b) \leq y)$ for $x, y : \mathcal{CE}$, implying immediately that the completion is Archimedean.

To show that these definitions now make the completion into an ordered field, we can use Lemma 7.8 to extend all identities which hold for the Eudoxus reals to its completion. This includes in particular the non-strict ordering, since its reflexivity can for instance be captured as $\max(x, x) = x$ for all $x : \mathcal{CE}$. Statements including the strict ordering can also be reduced to the Eudoxus reals itself, since a strict inequality on the completion requires a strict inequality on the Eudoxus reals.

We believe that the approach we just outlined, will prove that the completion of the Eudoxus reals is an Archimedean ordered field. Nevertheless we also believe that working out the details here and adapting the proofs of the HoTT book and Gilbert, would make an interesting follow-up project. For now, we will however assume in the next section that it indeed holds that the completion of the Eudoxus reals is an Archimedean ordered field.

7.4 Embedding

Then we arrive at our final goal, which is to prove that the completion of the Eudoxus reals embeds into any Cauchy complete Archimedean ordered field. This will then show that our construction is equivalent, and hence equal, to the Cauchy reals from the HoTT book. In this section we will first define this embedding, and then prove that the completion of the Eudoxus reals equals the Cauchy reals.

Before starting on the embedding we first want to make a remark on the type of rational numbers. We defined the type of rational numbers, $\mathbb{Q}$, as a subtype of the Eudoxus reals. Then the subset of $\mathcal{CE}$ consisting of every term $p : \mathcal{CE}$ such that there merely exists $q : \mathcal{E}$ such that $p = \eta(q)$ is again a construction of the rational numbers. During this section we will refer to this subtype of $\mathcal{CE}$ when referring to the type of rationals numbers $\mathbb{Q}$.

The HoTT book [41, Th. 11.3.50] shows that the Cauchy reals embed into any Cauchy complete Archimedean ordered field F by extending the canonical embedding of the rational numbers into F. We would like to construct a similar embedding, but the Eudoxus reals do not only consist of the rationals, but also of other elements. Hence we would need to determine to where we have to map these other elements. However, we cannot even decide given an arbitrary slope (or Eudoxus real) whether it is in $\mathcal{Q}$ (or $\mathbb{Q}$). Therefore, we cannot map the subtype of rational numbers to another field using the canonical embedding. Our solution is to construct for every slope, whether it is rational or not, a rational Cauchy approximation, apply to this the canonical embedding and then take the limit in F. We can only merely construct such an approximation in order to construct a function with codomain F, but we can strip the truncation if we define an auxiliary type as in Lemma 5.53.

To define this embedding we will need that every Cauchy complete Archimedean ordered field F is a premetric space, which enables us to work with Lipschitz and continuous functions. For this we will need to define additional functions on F. Firstly, since the rationals embed into every ordered field, there exists for any ordered field F an embedding $i : \mathbb{Q} \to F$ [41, p. 374]. Hence when we need a construction of the rationals in F, we will work the terms $p : F$ such that there merely exist $q : \mathbb{Q} \subset \mathcal{CE}$ such that

$p = i(q)$. Secondly, we define for all $x : F$ the absolute value $|x| : F$ as $|x| :\equiv \max(x, -x)$. Then the relation $a, b, \varepsilon \mapsto (|a - b| < i\varepsilon)$ for $a, b : F$ and $\varepsilon : \mathbb{Q}_+$ is a closeness relation [10].

Lemma 7.10. *Let F be a Cauchy complete Archimedean ordered field. Then there exists a function $e : \mathcal{CE} \to F$ which preserves $\mathbb{Q}$. That is, if $i : \mathbb{Q} \to F$ is the canonical embedding of the rationals into F, then for all $q : \mathbb{Q}$ it holds that $e(q) = i(q)$.*

Proof. Let F be a Cauchy complete Archimedean ordered field. To shorten our notation, we define $j :\equiv i \circ \eta \circ \mathsf{classof} : \mathcal{Q} \to F$.

We will define a function $e : \mathcal{CE} \to F$ as the extension of a Lipschitz map $f : \mathcal{E} \to F$. To define f we will use in turn the induction principle for $\mathcal{E}$. Hence we first define a map $g : \mathcal{S} \to F$ such that for all slopes $\lambda, \mu : \mathcal{S}$ if $\lambda \sim \mu$ then $g(\lambda) = g(\mu)$.

Define the auxiliary type family $P : \mathcal{S} \to \mathcal{U}$ by

$$P(\lambda) :\equiv \sum_{a:F} \prod_{\mu:\mathsf{Approximation}\ \mathcal{Q}} (\mu \to \lambda) \to (\lim(j\mu) = a),$$

where we denote by $\mu \to \lambda$ that μ converges to λ. Then $P(\lambda)$ is a mere proposition for all $\lambda : \mathcal{S}$ since this element $a : F$ is unique and $\lim(j\mu) = a$ is a mere proposition. Hence to define a function $h : \prod_{\lambda:\mathcal{S}} P(\lambda)$ we can strip the truncation in Lemma 5.45 to end up for all $\lambda : \mathcal{S}$ with a Cauchy approximation $\mu : \mathcal{Q}_+ \to \mathcal{Q}$ that converges to λ. Then we define $\mathsf{pr}_1\, h(\lambda) :\equiv \lim(j\mu)$. This limit exists since F is Cauchy complete and $j\mu$ is a Cauchy approximation in F as both i and $\mathsf{classof}$ are order preserving. To define $\mathsf{pr}_2\, h$, we have to check whether $\lim(j\mu)$ depends on the approximation μ.

Hence take $\lambda : \mathcal{S}$ and let $\mu, \mu' : \mathcal{Q}_+ \to \mathcal{Q}$ be two Cauchy approximations of λ. Since the relation $a, b, \varepsilon \mapsto (|a - b| < j\varepsilon)$ for $a, b : F$ and $\varepsilon : Q_+$ is a closeness relation, it is in particular separated. Hence it suffices to prove that for all $\varepsilon : \mathcal{Q}_+$ we have

$$|\lim(j\lambda) - \lim(j\lambda')| < j\varepsilon.$$

Take $\varepsilon : \mathcal{Q}_+$. Since j preserves orderings, we can bound the distance between the two limits using the triangle inequality

$$\begin{aligned} |\lim(j\mu) - \lim(j\mu')| &\leq |\lim(j\mu) - j\mu_{\varepsilon/8}| + |j\mu_{\varepsilon/8} - j\mu'_{\varepsilon/8}| + |j\mu'_{\varepsilon/8} - \lim(j\mu'_\varepsilon)| \\ &< \left(j\frac{\varepsilon}{8} + j\frac{\varepsilon}{8}\right) + j|\mu_{\varepsilon/8} - \mu'_{\varepsilon/8}| + \left(j\frac{\varepsilon}{8} + j\frac{\varepsilon}{8}\right) \\ &< j\frac{4\varepsilon}{8} + j\frac{4\varepsilon}{8} \\ &= j\varepsilon, \end{aligned} \tag{8}$$

where we used that $j\mu$ and $j\mu'$ are Cauchy approximations and that

$$|\mu_{\varepsilon/8} - \mu'_{\varepsilon/8}| \leq |\mu_{\varepsilon/8} - \lambda| + |\lambda - \mu'_{\varepsilon/8}| < \left(\frac{\varepsilon}{8} + \frac{\varepsilon}{8}\right) + \left(\frac{\varepsilon}{8} + \frac{\varepsilon}{8}\right) = \frac{4\varepsilon}{8}.$$

Hence $\lim(j\mu) = \lim(j\mu')$ and h is well-defined.

Now we can define $g : \mathcal{S} \to F$ as $g :\equiv \mathsf{pr}_1\, h$. To check that g induces a function $f : \mathcal{E} \to F$, let $\lambda, \lambda' : \mathcal{S}$ be equivalent. If $\mu : \mathcal{Q}_+ \to \mathcal{S}$ is a Cauchy approximation of λ, then it also holds that $\lim \mu = \lambda'$. Hence $g(\lambda) = g(\lambda')$ and so we have a function $f : \mathcal{E} \to F$.

Now we show that f is 2-Lipschitz to be able to extend it to a function $e : \mathcal{CE} \to F$. We prove this by induction on $\mathcal{E}$, where we do not have to take the path constructors into account since the goal is a mere proposition. Take $\lambda, \lambda' : \mathcal{S}$ and $\varepsilon : \mathcal{Q}_+$ such that $|\lambda - \lambda'| < \varepsilon$, then we have to show $|g(\lambda) - g(\lambda')| < 2\varepsilon$. We show this in a similar way as in (8), but we have to apply the triangle inequality twice more since in this case the limits of μ and μ' are not the same. Hence we first bound the difference between elements of the Cauchy approximations as

$$|\mu_{\varepsilon/8} - \mu'_{\varepsilon/8}| \leq |\mu_{\varepsilon/8} - \lambda| + |\lambda - \lambda'| + |\lambda' - \mu'_{\varepsilon/8}| < (\frac{\varepsilon}{8} + \frac{\varepsilon}{8}) + \varepsilon + (\frac{\varepsilon}{8} + \frac{\varepsilon}{8}) \sim \frac{12\varepsilon}{8},$$

[10] This is proven in Gilbert's formalization in the file `initial` as `F_premetric`.

after which we apply this to

$$\begin{aligned}|g(\lambda) - g(\lambda')| &= |\lim(j\mu) - \lim(j\mu')| \\ &\leq |\lim(j\mu) - j\mu_{\varepsilon/8}| + |j\mu_{\varepsilon/8} - j\mu'_{\varepsilon/8}| + |j\mu'_{\varepsilon/8} - \lim(j\mu'_\varepsilon)| \\ &< \left(j\frac{\varepsilon}{8} + j\frac{\varepsilon}{8}\right) + j|\mu_{\varepsilon/8} - \mu'_{\varepsilon/8}| + \left(j\frac{\varepsilon}{8} + j\frac{\varepsilon}{8}\right) \\ &\leq j\frac{4\varepsilon}{8} + j\frac{12\varepsilon}{8} \\ &= j(2\varepsilon).\end{aligned}$$

Hence f is 2-Lipschitz and we have finally defined a function $e : \mathcal{CE} \to F$.

Note that if $q : \mathcal{Q}$ that $g(q) = jq$ by approximating q with the constant approximation. Hence e preserves rationals. □

To check that e is an embedding of ordered fields we will need to check whether it preserves all operations on $\mathcal{CE}$ which make it an ordered field. This means that e has to satisfy the following:

- $e(0_{\mathcal{CE}}) = 0_F$,
- $e(1_{\mathcal{CE}}) = 1_F$,
- $e(x + y) = e(x) + e(y)$,
- $e(-x) = -e(x)$,
- $e(xy) = e(x)e(y)$,
- $e(\min(x, y)) = \min(e(x), e(y))$,
- $e(\max(x, y)) = \max(e(x), e(y))$,
- $x < y$ if and only if $e(x) < e(y)$,
- $x \leq y$ if and only if $e(x) \leq e(y)$,
- $x\#y$ if and only if $e(x)\#e(y)$,
- $e(1/x) = 1/e(x)$ if $0\#x$.

The requirement that e preserves the additive and multiplicative identity follows immediately, since e preserves the rationals. In the following we will check all other requirements in the order listed above, starting with addition. The only exception is that we will first prove the left-to-right direction of the two requirements related to the orderings before proving their right-to-left direction.

During the remainder of this section, F will be a Cauchy complete Archimedean ordered field and we will keep referring to e, f, g, h, i and j as defined in Lemma 7.10.

Lemma 7.11. *The embedding $e : \mathcal{CE} \to F$ preserves addition, that is $e(x + y) = e(x) + e(y)$ for all $x, y : \mathcal{CE}$.*

Proof. By applying Lemma 7.8 twice, we know that it is enough to show that for all $a, b : \mathcal{E}$ it holds that $e(\eta(a) + \eta(b)) = e(\eta(a)) + e(\eta(b))$ and that the functions $e \circ +_{\mathcal{CE}}$ and $+_F \circ (e, e)$ are continuous.

By definition $e \circ \eta = f$ and so we have to show $f(a + b) = f(a) + f(b)$ for $a, b : \mathcal{E}$, but this follows directly from the definition of h after applying the induction principle for $\mathcal{E}$.

To show that $e \circ +_{\mathcal{CE}}$ and $+_F \circ (e, e)$ are continuous, we can show the continuity of the separate functions before taking compositions. To see that addition on $\mathcal{CE}$ is continuous, note that the function is 1-Lipschitz in one variable since $|(a_1 + b) - (a_2 + b)| = |a_1 - a_2|$ for $a_1, a_2, b : \mathcal{CE}$. Hence the addition itself is (1+1)-Lipschitz. Similarly, we show that addition on F is 2-Lipschitz.

The function e is 2-Lipschitz by Lemma 7.7 since it is the Lipschitz extension of the 2-Lipschitz function f. Hence e is also continuous. From Lemma 7.2, it follows that the function $(e, e) : \mathcal{CE} \times \mathcal{CE} \to F \times F$ is 2-Lipschitz and therefore also continuous.

Since we have now established that the compositions $e \circ +_{\mathcal{CE}}$ and $+_F \circ (e, e)$ are continuous, we know that e preserves addition. □

Lemma 7.12. *The embedding $e : \mathcal{CE} \to F$ preserves negation, that is $e(-x) = -e(x)$ for all $x : \mathcal{CE}$.*

Proof. We could show this by using Lemma 7.8 again, but easier is to use the previous lemma. Take therefore $x : \mathcal{CE}$. Then using that e preserves addition we see that

$$e(-x) = e(-x) + 0 = e(-x) + e(x) - e(x) = e(-x + x) - e(x) = e(0) - e(x) = 0 - e(x) = -e(x).$$

□

Lemma 7.13. *The embedding $e : \mathcal{CE} \to F$ preserves multiplication, that is $e(x \cdot y) = e(x) \cdot e(y)$ for all $x, y : \mathcal{CE}$.*

Proof. By Lemma 7.8, it suffices to show that multiplication is continuous and that f preserves multiplication.

We start by showing that multiplication on F is continuous. Take $(x_1, y_1) : F \times F$ and $\varepsilon : i(\mathbb{Q}_+)$. We can merely bound $|x_1| + |y_1|$ by q for some $q : i(\mathbb{Q}_+)$ since F is Archimedean. Moreover, we can strip the truncation here because being continuous is a mere proposition. Then define $\delta :\equiv \min(1, \varepsilon, \varepsilon/q)/2$. Take $(x_2, y_2) : F$ such that (x_1, y_1) and (x_2, y_2) are δ close. Then by definition we have $|x_1 - x_2| < \delta$ and $|y_1 - y_2| < \delta$. From the triangle inequality it follows that

$$\begin{aligned}
|x_1y_1 - x_2y_2| &\leq |x_1y_1 - x_1y_2| + |x_1y_2 - x_2y_2| \\
&= |x_1||y_1 - y_2| + |x_1 - x_2||y_2| \\
&< (|x_1| + |y_2|)\delta \\
&\leq (|x_1| + |y_1| + |y_2 - y_1|)\delta \\
&< (q + \delta)\delta \\
&< \varepsilon.
\end{aligned}$$

Hence multiplication on F is continuous and similarly we can show that multiplication on $\mathcal{CE}$ is continuous.

Next we prove that f preserves multiplication. We apply induction on $\mathcal{E}$ and so we have to show for all $\lambda, \lambda' : \mathcal{S}$ that there exists Cauchy approximations $\mu, \mu', \kappa : \mathcal{Q}_+ \to \mathcal{Q}$ of respectively $\lambda, \lambda', \lambda \cdot \lambda'$ such that $\lim(j \circ \mu) \lim(j \circ \mu') = \lim(j \circ \kappa)$. By separatedness of F it suffices to show for all $\varepsilon : \mathcal{Q}_+$ that

$$|\lim(j \circ \mu) \lim(j \circ \mu') - \lim(j \circ \kappa)| < j(\varepsilon).$$

Hence take $\lambda, \lambda' : \mathcal{S}$ and let $\mu, \mu', \kappa : \mathcal{Q}_+ \to \mathcal{Q}$ be Cauchy approximations of respectively λ, λ' and $\lambda\lambda'$. Let $\varepsilon : \mathcal{Q}_+$ be given. Using the density of $\mathcal{Q}$ and since we are trying to prove a mere proposition, we can bound $|\lambda'|$ and $|\lambda|$ from above by respectively $a, b : \mathcal{Q}_+$. Take $\gamma = \min(\varepsilon/(8(a + b + 1)), c/6)$, where $c : \mathcal{Q}_+$ such that $c^2 \leq \varepsilon$. Then by repeatedly applying the triangle inequality and the definition of the limit of a Cauchy approximation, we get

$$\begin{aligned}
|\lim(j\mu) \lim(j\mu') - \lim(j\kappa)| &= |(j\mu_\gamma + (\lim(j\mu) - j\mu_\gamma))(j\mu'_\gamma + (\lim(j\mu') - j\mu'_\gamma)) \\
&\quad - (j\kappa_\gamma + (\lim(j\kappa) - j\kappa_\gamma))| \\
&= |j\mu_\gamma \cdot j\mu'_\gamma + (\lim(j\mu) - j\mu_\gamma)j\mu'_\gamma + (\lim(j\mu') - j\mu'_\gamma)j\mu_\gamma \\
&\quad + (\lim(j\mu) - j\mu_\gamma)(\lim(j\mu') - j\mu'_\gamma) - j\kappa_\gamma - (\lim(j\kappa) - j\kappa_\gamma)| \\
&< j|\mu_\gamma\mu'_\gamma - \kappa_\gamma| + |j\mu'_\gamma|j(2\gamma) + |j\mu_\gamma|j(2\gamma) \\
&\quad + j(2\gamma)j(2\gamma) + j(2\gamma).
\end{aligned} \tag{9}$$

We will bound the first three terms separately, starting with the first. To bound this term, we proceed in a similar way as in (9) to apply the triangle inequality

$$\begin{aligned}
|\mu_\gamma\mu'_\gamma - \kappa_\gamma| &\leq |\mu_\gamma\mu'_\gamma - \lambda\lambda'| + |\lambda\lambda' - \kappa_\gamma| \\
&< |(\lambda + (\mu_\gamma - \lambda))(\lambda' + (\mu'_\gamma - \lambda')) - \lambda\lambda'| + 2\gamma \\
&\leq |\mu_\gamma - \lambda||\lambda'| + |\mu'_\gamma - \lambda'||\lambda| + |\mu_\gamma - \lambda||\mu'_\gamma - \lambda'| + 2\gamma \\
&< 2\gamma(|\lambda'| + |\lambda| + 2\gamma + 1) \\
&< 2\gamma(a + b + 2\gamma + 1).
\end{aligned}$$

We can also use a, b to bound $|\mu_\gamma|$ using again the triangle inequality

$$|\mu_\gamma| \sim |\lambda| + |\mu_\gamma - \lambda| < a + 2\gamma,$$

and similarly we see $|\mu'_\gamma| < b + 2\gamma$. All together we now have

$$\begin{aligned}
|\lim(j\mu) \lim(j\mu') - \lim(j\kappa)| &< j(2\gamma(a + b + 2\gamma + 1 + a + 2\gamma + b + 2\gamma + 2\gamma + 1)) \\
&= j(4\gamma(a + b + 4\gamma + 1)).
\end{aligned}$$

By our choice of γ we can estimate

$$4\gamma(a + b + 4\gamma + 1) \sim 16\gamma^2 + 4\gamma(a + b + 1) < \frac{\varepsilon}{2} + \frac{\varepsilon}{2} = \varepsilon.$$

Hence $\lim(j \circ \mu) \lim(j \circ \mu') = \lim(i \circ \kappa)$ and so $g(\lambda)g(\lambda) = g(\lambda\lambda')$, showing that e respects multiplication. □

To show that the embedding also preserves the join and meet, we will first prove the following lemma which gives an upper bound to the difference between two joins.

Lemma 7.14. *Let A be an ordered field and let $a, b, c, d : A$. Then*

$$|\max(a, b) - \max(c, d)| \leq \max(|c - a|, |d - b|).$$

Proof. Let A be an ordered field and let $a, b, c, d : A$. Then

$$\max(a, b) = \max(a - c + c, b - d + d) \leq \max(a - c, b - d) + \max(c, d),$$

where the inequality holds since both $a - c + c$ and $b - d + d$ are less or equal to the right-hand side. Hence by moving $\max(c, d)$ to the left-hand side of the equation we have

$$\max(a, b) - \max(c, d) \leq \max(a - c, b - d) \leq \max(|a - c|, |b - d|),$$

where the second inequality follows since $a - c \leq |a - c|$ and $b - d \leq |b - d|$ which are both in turn smaller or equal to $\max(|a - c|, |b - d|)$.

By switching the roles of a by c, and b by d, we show that

$$\max(c, d) - \max(a, b) \leq \max(c - a, d - b) \leq \max(|a - c|, |b - d|),$$

which proves the statement. □

Lemma 7.15. *The embedding $e : \mathcal{CE} \to F$ preserves the join and meet, that is $\max(e(x), e(y)) = e(\max(x, y))$ and $\min(e(x), e(y)) = e(\min(x, y))$ for all $x, y : \mathcal{CE}$.*

Proof. We only prove the first statement, since the statement for min can be proven similarly. It suffices by Lemma 7.8 to show that $\max \circ (e, e), e \circ \max : \mathcal{CE} \to \mathcal{CE} \to F$ are continuous, and that the equality holds on $\mathcal{E}$.

For the continuity we only have to show that max is a continuous function on $\mathcal{CE}$ and F, since we have already shown that e and (e, e) are continuous. We only show this for F, since the proof is analogous for $\mathcal{CE}$. Take $\varepsilon : i\mathbb{Q}$ and let $(a, b) : F$ be given. Take $(c, d) : F$ such that $(a, b) \approx_\varepsilon (c, d)$, which means that $|c - a| < \varepsilon$ and $|d - b| < \varepsilon$. Then we see that max is continuous since

$$|\max(a, b) - \max(c, d)| \leq \max(|c - a|, |d - b|) < \varepsilon,$$

where the first inequality follows from Lemma 7.14 and the second follows from a similar reasoning as the proof of Lemma 5.61.

Then we can show that the equality holds on $\mathcal{E}$. We will prove this by induction on $\mathcal{E}$, where the path constructor is respected since the goal is a mere proposition. Hence, we have to show for all $\lambda, \lambda' : \mathcal{S}$ that

$$\max(g(\lambda), g(\lambda')) = g(\max(\lambda, \lambda')).$$

Take $\lambda, \lambda' : \mathcal{S}$ and let $\mu, \mu' : \mathsf{Approximation}\,\mathcal{S}$ be rational Cauchy approximations of respectively λ and λ'. Then we will show that

$$\max(\lim(j\mu), \lim(j\mu')) = \lim(j \max(\mu, \mu')). \tag{10}$$

We first show that the left-hand side of (10) equals $\lim(\max(j\mu, j\mu'))$, by showing that $\max(j\mu, j\mu')$ converges to this left-hand side. Take therefore $\varepsilon, \delta : \mathbb{Q}_+$, then by using Lemma 7.14 and Lemma 5.61 again we see that

$$|\max(\lim(j\mu), \lim(j\mu')) - \max(j\mu_\varepsilon, j\mu'_\varepsilon)| \leq \max(|\lim(j\mu) - (j\mu_\varepsilon)|, |\lim(j\mu') - (j\mu'_\varepsilon)|) < j(\varepsilon + \delta).$$

This proves that $\max(j\mu, j\mu')$ converges to the left-hand side of (10).

Next we note that $\lim(\max(j\mu, j\mu'))$ equals the right-hand side of (10). This follows from the observation that $\max(j\mu_\varepsilon, j\mu'_\varepsilon) = j \max(\mu_\varepsilon, \mu'_\varepsilon)$ for all $\varepsilon : \mathbb{Q}_+$, since j preserves orderings.

Hence (10) holds and therefore e preserves the join. □

To show that e respects both orderings, we will need the following lemma which relates the non-strict ordering to the maximum. The lemma holds on $\mathcal{CE}$ per definition, but we can prove it holds propositionally for all ordered fields.

Lemma 7.16. *Let A be an ordered field and let $a, b : A$. Then $(a \leq b) = (\max(a, b) = b)$.*

Proof. Using univalence and by noting that both sides are a mere proposition it suffices to show that $a \leq b$ if and only if $\max(a, b) = b$ for all $a, b : A$.

Let $a, b : A$ and assume $a \leq b$. Then we know by definition that $b \leq \max(a, b)$. On the other hand we also know that $\max(a, b) \leq b$, since it holds that $a \leq b$ and $b \leq b$. Hence by antisymmetry we have $\max(a, b) = b$.

For the second direction assume that $a, b : A$ are such that $\max(a, b) = b$. Then by definition $a \leq \max(a, b)$ and so $a \leq b$. □

Now we will first prove that e preserves the orderings, so the directions where the hypothesis is an inequality on $\mathcal{CE}$. We will use Lemma 7.16, to reduce the following corollary to Lemma 7.15.

Corollary 7.17. *The embedding $e : \mathcal{CE} \to F$ preserves the non-strict ordering, that is if $x \leq y$ then $e(x) \leq e(y)$ for all $x, y : \mathcal{CE}$.*

Proof. By Lemma 7.16 it suffices to show that for all $x, y : \mathcal{CE}$ if $\max(x, y) = y$ that then $\max(e(x), e(y)) = e(y)$. By replacing the y on the right-hand side of the goal with $\max(x, y)$ from the hypothesis, we see that we have to show

$$\max(e(x), e(y)) = e(\max(x, y)).$$

This is something we already showed in Lemma 7.15. □

Using the Archimedean property and Corollary 7.17, it follows that e also preserves the strict-ordering.

Corollary 7.18. *The embedding $e : \mathcal{CE} \to F$ preserves the strict ordering, that is if $x < y$ then $e(x) < e(y)$ for all $x, y : \mathcal{CE}$.*

Proof. Let $x, y : \mathcal{CE}$ be such that $x < y$. Since $\mathcal{CE}$ is Archimedean we know there exists $q, r : \mathbb{Q}$ such that $x < q < r < y$. We stripped the truncation here, because the non-strict ordering on F is also a mere relation. By Corollary 7.17 it follows that $e(x) \leq e(q)$ and $e(r) \leq e(y)$. Since e coincides for rationals with the canonical embedding of $\mathbb{Q}$ in F, we know that $e(q) < e(r)$. Hence by transitivity on F, we conclude that $e(x) < e(y)$. □

Then we can use that the embedding e preserves the orderings, to show that it also reflects the orderings. This is the other direction, where the hypothesis is an inequality on F.

Corollary 7.19. *The embedding $e : \mathcal{CE} \to F$ reflects the non-strict ordering, that is if $e(x) \leq e(y)$ then $x \leq y$ for all $x, y : \mathcal{CE}$.*

Proof. Take $x, y : \mathcal{CE}$ such that $e(x) \leq e(y)$. Since $\mathcal{CE}$ is an ordered field, it suffices to show that $\neg(y < x)$. Hence assume $y < x$ for a contradiction. Then by Corollary 7.18 it follows that $e(y) < e(x)$. But since also F is a ordered field, the hypothesis $e(x) \leq e(y)$ implies that $\neg(e(y) < e(x))$. Hence we have a contradiction and the statement is proven. □

Corollary 7.20. *The embedding $e : \mathcal{CE} \to F$ reflects the strict ordering, that is if $e(x) < e(y)$ then $x < y$ for all $x, y : \mathcal{CE}$.*

Proof. We can repeat the proof of Corollary 7.18 by interchanging the roles of F and $\mathcal{CE}$, and by using Corollary 7.19 instead of Corollary 7.17. □

Since the apartness relation of an ordered field is closely related to the strict-ordering, we reduce the statement that e preserves the apartness relation to the statements that e both preserves and reflects the strict ordering.

Corollary 7.21. *The embedding $e : \mathcal{CE} \to F$ preserves the apartness relation, that is $x \# y$ if and only if $e(x) \# e(y)$ for all $x, y : \mathcal{CE}$.*

Proof. Since F is an ordered field, it holds that $x \# y$ if and only if $(x < y) + (y < x)$ for all $x, y : F$, and similarly for $\mathcal{CE}$. Hence the statement follows immediately from Corollary 7.18 and Corollary 7.20. □

Since the apartness relation is preserved, we can now also show that the embedding preserves division.

Lemma 7.22. *The embedding $e : \mathcal{CE} \to F$ preserves the division, that is $e(1/x) = 1/e(x)$ for all $x : \mathcal{CE}$ such that $0 \# x$.*

Proof. We can prove this by showing that division is continuous and that the equality holds on $\mathcal{E}$. Easier is however to show that the multiplicative inverse is unique. Take therefore $x : \mathcal{CE}$. Then using Lemma 7.13, which states that the embedding preserves multiplication, we have

$$e\left(\frac{1}{x}\right) = 1 \cdot e\left(\frac{1}{x}\right) = \frac{1}{e(x)} \cdot e(x) \cdot e\left(\frac{1}{x}\right) = \frac{1}{e(x)} \cdot e\left(x \cdot \frac{1}{x}\right) = \frac{1}{e(x)} \cdot e(1) = \frac{1}{e(x)} \cdot 1 = \frac{1}{e(x)}.$$

□

Since we now have checked all requirements for e to be an embedding of ordered fields, and since it fixes the rationals, we have now shown that the completion of the Eudoxus reals embeds into every Cauchy complete Archimedean ordered field.

Theorem 7.23. *The completion of the Eudoxus reals embeds into every Cauchy complete Archimedean ordered field.*

To now prove that the completion of the Eudoxus reals is equal to the Cauchy reals, we shortly recall the definition of the embedding of the Cauchy reals into a Cauchy complete Archimedean ordered field from the HoTT book [41, Th. 11.3.50]. Let F be such a field and let $\mathbb{R}_{\mathsf{C}} :\equiv \mathcal{CQ}$ denote the Cauchy reals. Let $i' : \mathbb{Q}_{\mathbb{R}_{\mathsf{C}}} \to F$ be the canonical embedding of the rationals to F, where we see the rationals as subtype of $\mathbb{R}_{\mathsf{C}}$. Then using recursion, we define $e' : \mathbb{R}_{\mathsf{C}} \to F$ for $q : \mathbb{Q}_{\mathbb{R}_{\mathsf{C}}}$ as

$$e'(\eta(q)) :\equiv i'(q)$$

and for Cauchy approximations $x : \mathsf{Approximation}\,\mathbb{R}_{\mathsf{C}}$ as

$$e'(\lim(x)) :\equiv \lim(e' \circ x)$$

Corollary 7.24. *The completion of the Eudoxus reals is equal to the Cauchy reals.*

Proof. Let $e : \mathcal{CE} \to \mathbb{R}_{\mathsf{C}}$ be the embedding of Lemma 7.10 and let $e' : \mathbb{R}_{\mathsf{C}} \to \mathcal{CE}$ be the embedding from the HoTT book, whose definition we recalled above. By univalence it suffices to show that the completion of the Eudoxus reals and the Cauchy reals are equivalent. Hence, we will show that $e \circ e' = \mathrm{id}_{\mathbb{R}_{\mathsf{C}}}$ and $e' \circ e = \mathrm{id}_{\mathcal{CE}}$.

By Lemma 7.8 it suffices to show these equalities on respectively $\mathbb{Q}$ and $\mathcal{E}$, because both e and e' are continuous. Since e and e' also both preserve the rationals it follows immediately that for all $q : \mathbb{Q}$ we have $(e \circ e')(q) = q$ and hence $e \circ e' = \mathrm{id}_{\mathbb{R}_{\mathsf{C}}}$.

For the other equality, we use induction on the Eudoxus reals to reduce the statement to slopes. Then we have to prove for all $\lambda : \mathcal{S}$ that

$$e'(e(\eta(\mathsf{classof}\ \lambda))) = \eta(\mathsf{classof}\ \lambda).$$

Let $\mu : \mathcal{Q}_+ \to \mathcal{Q}$ be a Cauchy approximation of λ. Then by definition of e and subsequently f and g, this reduces to showing that

$$e'(\lim(j\mu)) = \eta(\mathsf{classof}\ \lambda). \tag{11}$$

By definition of e' the left-hand side equals $\lim(e'(j\mu))$. Recall that we defined j as the composition $i \circ \eta \circ \mathsf{classof}$. Here i is the canonical embedding of the rationals in $\mathcal{CE}$ to $\mathbb{R}_{\mathsf{C}}$, whereas $e'\big|_{\mathbb{Q}}$ is the canonical embedding of the rationals in $\mathbb{R}_{\mathsf{C}}$ to $\mathcal{CE}$. Hence $e' \circ i$ is just the identity on the rationals in $\mathcal{CE}$, and the left-hand side of (11) equals $\lim(\eta(\mathsf{classof}\ \mu))$. This equals the right-hand side of (11), since for all $\varepsilon : \mathbb{Q}_+$ we have

$$|\eta(\mathsf{classof}\ \lambda) - \eta(\mathsf{classof}\ \mu_\varepsilon)| = |\eta(\mathsf{classof}\ \lambda - \mathsf{classof}\ \mu_\varepsilon)| = |\eta(\mathsf{classof}(\lambda - \mu_\varepsilon))|$$

and both η and $\mathsf{classof}$ preserve the strict ordering. □

In particular this implies that the completion of the Eudoxus reals is an *initial* Cauchy complete Archimedean ordered field since the Cauchy reals are.

We also could have shown this result directly without Corollary 7.24 by using the density of the rationals in the completion of the Eudoxus reals. We prove this in the lemma below. This would then have implied in turn the equivalence of the Cauchy reals and the completion of the Eudoxus reals as initial objects are unique. This latter result is for instance shown in the HoTT book [41, Th. 5.4.4], although in a different setting.

Lemma 7.25. *The completion of the Eudoxus reals is an* initial *Cauchy complete Archimedean ordered field. That is, if F is a Cauchy complete Archimedean ordered field and $e_1, e_2 : \mathcal{CE} \to F$ are embeddings of ordered fields preserving the rationals, then $e_1 = e_2$.*

Proof. Let F and $e_1, e_2 : \mathcal{CE} \to F$ satisfy the hypotheses and take $x : \mathcal{CE}$.

Since the completion of the Eudoxus reals is Archimedean, we can approximate x by a Cauchy approximation $q : \mathsf{Approximation}\,\mathcal{CE}$ such that $\lim q = x$ and such that $q_\varepsilon : \mathbb{Q} \subset \mathcal{CE}$ for all $\varepsilon : \mathbb{Q}_+$. It holds that $\lim(e_1 q) = e_1(\lim q)$ and similarly $\lim(e_2 q) = e_2(\lim q)$, since e_1 and e_2 preserve the strict ordering. Since e_1 and e_2 also both preserve the rationals, it follows that $e_1(q_\varepsilon) = e_2(q_\varepsilon)$ for all $\varepsilon : \mathbb{Q}_+$ and hence $\lim(e_1 q) = \lim(e_2 q)$. Therefore we see that

$$e_1(x) = e_1(\lim q) = \lim(e_1 q) = \lim(e_2 q) = e_2(\lim q) = e_2(x)$$

and by functional extensionality we then have $e_1 = e_2$.

□

8 Higher inductive types in Coq

Now that we have concluded that the completion of the Eudoxus reals is equivalent to the Cauchy reals, we will in this and the next section look how we can formalize in Coq that the Eudoxus reals are an Archimedean ordered field. In this section we will first explain our choice to use the HoTT library to implement our results in Coq. Then we will discuss how we can define a higher inductive type in Coq, because Coq itself does not support the definition of such types. In Section 9 we will discuss our actual formalization.

In Appendix A we included a list of tactics we used in the formalization in Coq of which we already use a few in this section. A reader unfamiliar with Coq may find this appendix useful.

8.1 The UniMath and HoTT libraries

There are two main libraries for Coq which add homotopy type theory to Coq. The first is the UniMath library[11], which is based on Voevodsky's Foundations library and two later repositories extending this Foundations library [43]. The second is the HoTT library[12], whose initial goal was to gain an understanding of this same Foundations library of Voevodsky [7]. We will now shortly compare the two libraries.

The first difference between the two libraries are their goals. The HoTT library namely focuses on the HoTT book and continues in this direction to develop homotopy type theory in Coq [7]. The UniMath library on the contrary focuses on the formalization of mathematics and so contains for instance more results on category theory.

A practical difference for readers of the HoTT book is that the HoTT library follows the notation in the book. It even has a file stating under what name the formalizations of the results in the HoTT book can be found[13]. This is not the case for the UniMath library. An example of this is `apd` from Section 2.4, which is called `apD` in the HoTT library and `transport_section` in the UniMath library.

A more technical difference is that the HoTT library deals with universe polymorphism, which makes it possible to declare definitions to be used in different universes. The UniMath library on the other hand assumes the inconsistent assumption that `Type:Type`. This can be seen as an example of the more conservative approach of UniMath to new Coq features, whereas the HoTT library states to serve as a testbed for these new features [7].

For this project we are interested in formalizing higher inductive types in Coq. This has not yet been implemented in the UniMath library, but has been in the HoTT library. A large part of the chapter in the HoTT book on higher inductive types, Chapter 6, has been formalized in the HoTT library including the quotient type. Hence we chose to use the HoTT library for this project.

The HoTT library does however not yet contain a formalization of the Cauchy reals from Chapter 11 of the HoTT book, since this is a higher inductive-inductive type. They do plan to include Gilbert's formalization [25] of the Cauchy reals, which uses an experimental branch of Coq, when Coq has sufficiently developed [7].

8.2 Axioms

In the next sections we will see how we can define higher inductive types in Coq, since this it not yet supported by Coq itself. Possibly the easiest option to define higher inductive types in Coq is to use axioms. Then we declare the point constructors as usual in an inductive type using `Inductive`, and we assume the path constructor as axioms. However, this needs to be done with caution as adding paths as axioms can easily cause inconsistencies. Hence in this section we will give a short example as to how axioms can cause inconsistencies, to show the need for another method to implement higher inductive types in Coq.

As an example we start with the booleans which are defined in HoTT library as follows. We first state that we define an inductive type of the name `Bool` by `Inductive Bool`. Then we declare it as a set by stating that it is an element of `Type0`, which is an alias for `Set`. Then the two point constructors are stated which ends our definition of the booleans.

```
Inductive Bool : Type0 :=
  | true : Bool
```

[11] The UniMath library is available at https://github.com/UniMath/UniMath
[12] The HoTT library is available at https://github.com/HoTT/HoTT
[13] Available in the Github repository of the HoTT library as `contrib/HoTTBook`

```
  | false : Bool.
```

The booleans are not a higher inductive type, but the interval type I is. This type is defined as the type generated by the point constructors $\mathsf{zero} : I$ and $\mathsf{one} : I$, and a path constructor $\mathsf{seg} : \mathsf{zero} = \mathsf{one}$. Hence to define the interval in Coq, we can just take the definition of `Bool` and add a path `false = true` as an axiom.

```
Axiom path : false = true.
```

However, now we have a contradiction with `true_ne_false` from the HoTT library in `Types/Bool`, which states that `false` is not equal to `true`. We leave some code out which we indicate by the use of `[...]`.

```
Definition false_ne_true : ~ (false = true)
  := fun H => match H [...]
  end.
```

To see that we indeed have created an inconsistency, note that if we apply `false_ne_true` to our axiom `path`, we have constructed an element of the the empty type `empty`. Using the reduction principle of the empty type, we can now prove anything.

```
Lemma contradiction : empty.
Proof.
  exact (false_ne_true path).
Defined.
```

Hence we see that it is not a suitable method to define a higher inductive type in Coq by just stating the path constructors as axioms. The solution in the next section will still use axioms to define the path constructors, but will also disable the use of `match`. Hence in that case we cannot prove `false_ne_true` by the current proof. The solution in the next section instead forces us to use an induction principle which takes paths into account if we want to prove a statement for a higher inductive type. Hence when we want to prove a result for the interval, we cannot just use the the induction principle of the booleans.

To see how taking paths into account prevents inconsistencies when using axioms, we will try to prove `false_ne_true` using the induction principle for the interval type. We could try to mimic the approach in the UniMath library. There `false_ne_true` is also proven but under the name `nopathsfalsetotrue`. Their proof uses of course the induction principle without taking paths into account, but this enables us to see the importance of paths.

The library first defines a function `bool_to_type : Bool->Type` using the induction principle, where `true` is sent to `empty`, and `false` to `unit`, which are the empty and unit type respectively. Let `tt` be the constructor of `unit`. Then they use the function `bool_to_type` to transport `tt : bool_to_type false` via the path `path : false = true` to an element of `bool_to_type true`. This is however the empty type and therefore causes an inconsistency.

If we now also take the path constructor `false = true` into account, we would have to state a path `bool_to_type false = bool_to_type true`. This means that we have to prove that `empty = unit` in order to define the function `bool_to_type`. This is something we cannot do, which means that we cannot prove `false_ne_true`. Hence taking path constructors into account when using an induction principle indeed prevents inconsistencies.

8.3 Private inductive types

To prevent the inconsistencies from Section 8.2 when defining higher inductive types in Coq, Bertot [8] adapted a method to Coq which was initially proposed by Licata for the proof assistant Agda [32]. In this section we will discuss the method in general and then in the next section we will show as an example how this method is used to define the circle in Coq. The idea of the method is that we can prevent inconsistencies if we disable the use of matching and we replace the induction principle which Coq by default defines for an inductive type by the correct induction principle which also takes path constructors into account. The method is also summarized in the style document of the HoTT library [31].

First however, we take a look at what `match` does, to see why we need to disable it. The tactic `match` refers to the more general practice of pattern matching in type theory, which can be seen as a form of case-analysis. It is the convention that to define for instance a function on a coproduct $f : A + B \to C$ we state $f(\mathsf{inl}(a)) :\equiv g_A(a)$ for $a : A$ and $f(\mathsf{inr}(b) :\equiv g_B(b)$ for $b : B$. Implicitly we then define f using the recursion principle and equivalently we could also state that $f :\equiv \mathsf{rec}_{A+B}(C, g_A, g_B)$. However, using the recursion (or induction) principle explicitly in the latter definition of f makes the definition of f often less readable. Hence we often use pattern matching as a convenient way to denote the recursion principle.

In Coq we use pattern matching similarly when working with inductive types. An example is the definition of `not` from the Coq standard library which maps `true` to `false`, and `false` to `true`. It uses pattern matching to decide whether the input matches `true` or `false` and maps the input to the corresponding term.

```
Definition not (b:bool) :=
  match b with
  | true => false
  | false => true
  end.
```

This all means however that `match` only takes the point constructors into account, which is the reason why we have to disable it.

The idea of the method is therefore to define the point constructors of a higher inductive type in a private inductive type instead of an ordinary inductive type. This prevents Coq from automatically defining an induction principle for the inductive type. When used within a module, the `Private` prefix also disables the use of `match` outside of the module. However, when we are working in the same module as the definition of the higher inductive type, we can still use `match` to define the correct induction principle. We will not discuss here what a module in Coq actually entails or how it can be used more generally. For this we refer an interested reader to the documentation [17] or Coq's tutorial on modules[14].

Hence when we want to define a higher inductive type, we first open a module with for example the name `myHIT` by `Module myHIT`. Then we define a `Private Inductive` type with as constructors the point constructors of the higher inductive type we want to define, just as for an ordinary inductive type. Next, we assume the path constructors as axioms.

Then, when still working in the module, we define the induction principle using pattern matching on the point constructors. We want the induction principle to take the path constructors into account by requiring the relevant path hypotheses. However, we also have to make sure to actually use these path hypotheses in the definition of the induction principle, because otherwise we introduce other inconsistencies which we discuss in Section 8.5. This is often done by defining the induction principle as a function applied to all the path hypotheses. An example of this can be found in Section 8.4. Since we cannot define how the induction principle acts on paths, we also assume as axioms the computation rules for the path constructors.

After defining the induction principle we close the module by `End myHIT`. This makes sure that in our further dealings with the higher inductive type we only use the correct induction principle which takes path constructors into account. Often the module is followed by defining the recursion principle and its computation rules, using the induction principle we defined inside the module.

This method does require from the programmer that he defines a correct induction principle which does not cause any inconsistencies. However the method at least prevents the use of methods surely leading to inconsistencies outside of the module, such as pattern matching and induction without considering the path constructors. The developers of the HoTT library state on this topic that "the standard of rigor in formalized proofs is at least a great deal higher than the generally accepted level of rigor in traditional written mathematics" [7].

8.4 The circle in Coq

To illustrate how higher inductive types are defined in Coq using the method of the previous section, we will discuss here the example of the circle. The circle S^1 is defined as generated by the point $\mathsf{base} : S^1$ and the path $\mathsf{loop} : \mathsf{base} = \mathsf{base}$. Hence, following the method described in the previous section, our definition of circle should start as follows. We added `Export` to the opening of the module to be able to refer outside of the module to the types and terms defined in module directly. Hence using `Export`, we can refer to `base` outside of the module instead of having to write `Circle.base`.

```
Module Export Circle.
Private Inductive S1 : Type :=
  | base : S1.
Axiom loop : base = base.
```

Now we have to define the induction principle of the circle, since Coq does not do this automatically because we defined the circle as a private type. Hence we want to state that given $P : S^1 \to U$,

[14]The tutorial is part of Coq's Wiki on Github and is available at `https://github.com/coq/coq/wiki/ModuleSystemTutorial`.

$b : P(\mathsf{base})$ and $l : \mathsf{transport}^P(\mathsf{loop}, b) = b$ we can define a function $f : \prod_{x \in S^1} P(x)$ such that $f(\mathsf{base}) :\equiv b$ and $\mathsf{apd}_f(\mathsf{loop}) = l$. A first attempt to define the induction principle in Coq might look like the following, where we define `S1_rect` by sending `base` to `b`. An axiom ensures that `loop` is indeed mapped to `l`.

```
Definition S1_ind (P:S1->Type) (b : P base) (l : loop # b = b)
   : forall (x:S1), P x
   := fun x => match x with base => b end.

Axiom S1_ind_beta_loop : forall (P : S1 -> Type)
   (b : P base) (l : loop # b = b),
   apD (S1_ind P b l) loop = l.
```

However, this definition could cause an inconsistency, because Coq will notice that the path `l` is not used anywhere in the definition of `S1_ind`. From this Coq will conclude that `S1_ind P b l1` is judgmentally equal to `S_ind P b l2` even though `l1` and `l2` are not [7]. This leads to inconsistencies, as we will give an example of in Section 8.5.

The solution is to change `S_ind` to the following. The only difference here is in the match construction of `S_ind`. In the first attempt we defined `S_ind` to be the function which sent `base` to `b`. Here we map `base` to the evaluation at `l` of a function which sends everything to `b`. In this way the definition of the induction principle does depend on `l` and so Coq will not conclude that `S_ind P b l1` and `S_ind P b l2` are judgmentally equal. We tried to make the definition clearer by adding some extra brackets compared to the formulation in the library.

```
Definition S1_ind (P : S1 -> Type) (b : P base) (l : loop # b = b)
  : forall (x:S1), P x
  := fun x => ((match x with base => (fun _ => b) end) l).
```

In conclusion, the full definition of the circle in Coq is now as in the HoTT library [15].

```
Module Export Circle.

Private Inductive S1 : Type1 :=
  | base : S1.

Axiom loop : base = base.

Definition S1_ind (P : S1 -> Type) (b : P base) (l : loop # b = b)
  : forall (x:S1), P x
  := fun x => match x with base => fun _ => b end l.

Axiom S1_ind_beta_loop
  : forall (P : S1 -> Type) (b : P base) (l : loop # b = b),
  apD (S1_ind P b l) loop = l.

End Circle.
```

with recursion principle

```
Definition S1_rec (P : Type) (b : P) (l : b = b)
  : S1 -> P
  := S1_ind (fun _ => P) b (transport_const _ _ @ l).

Definition S1_rec_beta_loop (P : Type) (b : P) (l : b = b)
  : ap (S1_rec P b l) loop = l.
Proof.
  [...]
Defined.
```

[15] This is actually the definition of the Circle in the HoTT library (in `HIT/Circle`) as in commit a80b13c in the master branch. Since March 28th 2015 the circle is implemented in the library (in `Spaces/Circle`) as a coequalizer. Since then Coq marked `Private Inductive` as legacy and has replaced this with `#[ private(matching) ] Inductive` [18]. The HoTT library however still uses the former in current implementations of higher inductive types.

8.5 An inconsistent definition of the induction principle

In Section 8.4 we discussed how to define the circle in Coq as a higher inductive type. We stated that defining the induction principle of the circle as follows would cause inconsistencies. In this section we give an example of such an inconsistency.

```
Definition S1_ind (P:S1->Type)
  (b : P base) (l : loop # b = b)
  : forall (x:S1), P x
  := fun x => match x with base => b end.
```

First we show that Coq indeed ignores the path argument of the induction principle. We will do this by showing that if $P : S^1 \to \mathcal{U}$, $b : P(\mathsf{base})$ and $l_1, l_2 : \mathsf{transport}^P(\mathsf{loop}, b) = b$ that $\mathsf{ind}_{S^1}(P, b, l_1) \equiv \mathsf{ind}_{S^1}(P, b, l_2)$. This looks in Coq as follows. Note that if we would have used the correct definition of the induction principle, we could not have applied reflexivity here and therefore we would not end up with a contradiction.

```
Lemma S1_ind_indendent_path
   : forall (P : S1 -> Type)(b : P base)
      (l1 l2: loop # b = b),
      (S1_ind P b l1 = S1_ind P b l2).
Proof.
   reflexivity.
Defined.
```

Similarly we show that also the recursion principle does not care for the path argument, which means that if $A : \mathcal{U}$, $a : A$ and $l_1, l_2 : a = a$ that $\mathsf{rec}_{S^1}(A, a, l_1) \equiv \mathsf{rec}_{S^1}(A, a, l_2)$.

```
Lemma S1_rec_independent_path
   : forall {A: Type} {a : A} (l1 l2: a = a),
      S1_rec _ _ l1 = S1_rec _ _ l2.
Proof.
   reflexivity.
Defined.
```

Now we are able to show that $\mathsf{base} = \mathsf{base}$ is a mere proposition. Hence we show that for all l_1 and l_2 of type $\mathsf{base} = \mathsf{base}$, we have $l_1 = l_2$. We do this by noting that if the recursion principle ignores the path argument, we can interchange $\mathsf{rec}_{S^1}(S^1, \mathsf{base}, l_1)$ with $\mathsf{rec}_{S^1}(S^1, \mathsf{base}, l_2)$ to end up with

$$l_1 = \mathsf{ap}_{\mathsf{rec}_{S^1}(S^1,\mathsf{base},l_1)}(\mathsf{loop}) = \mathsf{ap}_{\mathsf{rec}_{S^1}(S^1,\mathsf{base},l_2)}(\mathsf{loop}) = l_2.$$

In Coq we prove this as follows, where we use `S1_rec_beta_loop` from the previous section.

```
Lemma l1isl2 : forall (l1 l2: base=base), l1=l2.
Proof.
   intros l1 l2.
   rewrite <- (S1_rec_beta_loop _ _ l1).
   rewrite <- (S1_rec_beta_loop _ _ l2).
   auto.
Defined.
```

In particular, we can apply this result to $\mathsf{refl}_{\mathsf{base}}$ and loop to conclude that $\mathsf{refl}_{\mathsf{base}} = \mathsf{loop}$, which is in contradiction with $\neg(\mathsf{refl}_{\mathsf{base}} = \mathsf{loop})$ [41, Lem. 6.4.1].

```
Lemma inconsistency : idpath base = loop.
Proof.
   apply l1isl2.
Defined.
```

8.6 Quotient type

As stated earlier, an important reason for us to use the HoTT library in Coq is that it contains higher inductive types. In particular the quotient type is defined in the HoTT library[16] which we use to define

[16] The definition of the quotient type in the HoTT library can be found in `HIT/quotient`.

the Eudoxus reals. Hence in this section we will take a short look at this definition and see that it is indeed defined using a `Module` and `Private Inductive` as we just discussed.

Its definition is very similar to the one in the HoTT book. The module starts with defining a context which we will discuss in Section 9.9, but for now we shortly explain as assumptions added to the context. Hence we assume that `A` is a type with a mere relation `R`. Then we see that the `Private Inductive` called `quotient` is defined with point constructor `class_of`.

```
Context {A : Type} (R : Relation A) {sR: is_mere_relation _ R}.

Private Inductive quotient {sR: is_mere_relation _ R} : Type :=
  | class_of : A -> quotient.
```

Then the path constructors `related_classes_eq` and `quotient_set` are defined as `Axiom`s. This is a good example how closely the HoTT library follows the HoTT book, since these path constructors are even given in the same order as in the definition of the HoTT book.

```
Axiom related_classes_eq
  : forall {x y : A}, R x y ->
  class_of x = class_of y.

Axiom quotient_set : IsHSet (@quotient sR).
```

Then the induction principle is defined as follows. We see that the induction principle is defined as a function evaluated in the path constructors `sP` and `dequiv`.

```
Definition quotient_ind (P : (@quotient sR) -> Type)
  {sP : forall x, IsHSet (P x)} (dclass : forall x, P (class_of x))
  (dequiv : (forall x y (H : R x y),
  (related_classes_eq H) # (dclass x) = dclass y))
  : forall q, P q
  := fun q
    => match q with class_of a => fun _ _ => dclass a end sP dequiv.
```

To simplify the use of the induction principle, it is proven separately that the induction principle behaves as expected on points. That is, if `x:A` then the induction principle sends `class_of x` to `dclass x`.

```
Definition quotient_ind_compute {P sP} dclass dequiv x
  : @quotient_ind P sP dclass dequiv (class_of x) = dclass x.
Proof.
  reflexivity.
Defined.
```

As in the example of the circle from Section 8, it is not possible to prove that the induction principle also behaves on paths as expected. Hence we assume it as an axiom.

```
Axiom quotient_ind_compute_path
  : forall P sP dclass dequiv,
  forall x y (H : R x y),
  apD (@quotient_ind P sP dclass dequiv) (related_classes_eq H)
  = dequiv x y H.
```

This ends the definition of the quotient type inside the module. Next the recursion principle is defined, but for this we refer to the file in HoTT library.

9 Formalization of the Eudoxus reals in Coq

In this section we will show how we formalize in Coq that the Eudoxus reals are an Archimedean ordered field, now that we know how higher inductive types can be implemented in Coq.

We will first shortly state our definition of slopes and the Eudoxus reals in Coq in Section 9.1. However, since we follow the definitions and lemmas in Section 5 closely, we will not discuss all of their implementations in Coq. We will however discuss when and how the implementation in Coq differs significantly from Section 5. This will in general not include small changes to for instance account for the 'without loss of generality' in proofs. We will also discuss some of the workings of Coq such as the use of `Context` we saw used in the definition of the quotient type in 8.6.

We were however not able to prove everything in Coq due to time constraints. Hence we will discuss in Section 9.11 what is and what is not implemented. Before this, we will however explain the general structure of the Coq files in Section 9.10.

In Appendix A we have included a list of tactics we used in the formalization in Coq.

9.1 Slopes and Eudoxus reals

We define the type `slopes` using the following three definitions. In these definitions Coq assumes that `Z` is a construction of the integers. We explain in Section 9.9 how Coq knows this and why `` `{Integers Z} `` appears in the definition of `slopes`.

```
Definition IsBound (l:Z->Z) (S:Z)
  := forall (m n : Z), (-S ≤ l(m+n)-l(m)-l(n) ≤ S).

Definition IsSlope (l: Z-> Z) := exists (S : Z), IsBound l S.

Definition slopes `{Integers Z} := exists (l:Z -> Z), IsSlope l.
```

Hence a term of `slopes` consists of a function `l:Z -> Z`, a bound `S:Z` and a proof that indeed `-S ≤ l(m+n)-l(m)-l(n) ≤ S` for all `m n : Z`. In statements and proofs we will often use not the whole slope, but only the function, its bound or the inequality they satisfy. Therefore, we define the projection functions `pr_slope`, `pr_bound` and `pr_ineq` which map a slope to respectively its function, bound or the associated inequality.

Having defined slopes, we can define the equivalence relation. We denote it by `equiv_slopes` and define it exactly as in Section 5. The `merely` indicates the truncation to a mere proposition.

```
Definition equiv_slopes (k l : slopes)
  := merely (exists (S:Z), forall (n:Z),
  -S ≤ pr_slope k n - pr_slope l n ≤ S).
```

Then we can define the type of the Eudoxus reals, which we denote by `e_reals`, as the quotient type with respect to `equiv_slopes`. The type `slopes` and the proof that `equiv_slopes` is a mere relation are inferred from the context, but more on this in Section 9.9.

```
  Definition e_reals := quotient equiv_slopes.
```

The function `class_of` from the definition of the quotient type can now be applied to a slope `l:slopes` to get the Eudoxus real `class_of l : e_reals`.

9.2 Concentration lemma

Often we can just follows the proofs of A'Campo and Mizrahi, but sometimes we have to make some (minor) adjustments. To give the reader an example, we will here discuss the concentration lemma and some adjustments we had to make in Coq.

A'Campo proves that given a slope λ with bound $d_\lambda : \mathbb{N}_+$, the slope λ' given by $\lambda'(n) = \lambda(3d_\lambda n) : 3d_\lambda n$ for $n : \mathbb{Z}$ is a well-adjusted slope equivalent to λ. However, the optimal euclidean divisor is not an odd function [35, Cor. 4.40]. Therefore, the thesis of Mizrahi already takes the odd extension of this slope. We define this function for non-zero bounds in `waslope_close_helpfunction` and prove it is a slope in `waslope_close_helpslope`. The `help` in these definitions indicates that this is not the final definition of the well-adjusted slope, since we also need to account for slopes with bound zero. In such cases `waslope_close_helpslope` is namely not defined. Hence we take an extra step to include this case in `waslope_close`, which construct for all slopes the well-adjusted slope equivalent to the given slope.

We could have chosen to implement 'without loss of generality the bound is strictly positive'. Then in the case of a slope `l` with bound 0, we could prove that a slope `k` consisting of the same function but with bound 1 is equivalent to `l` and proceed the proof using `k`. We chose however to define `waslope_close` for slopes with bound 0 as the identity and use `waslope_close_helpslope` for slopes with nonzero bound, since we believed this to be easier. Then when proving that `waslope_close` outputs indeed a well-adjusted slope, this reduces to the proof of A'Campo if this bound is non-zero, and to $0 \leq 1$ if the bound is zero. To prove that the well-adjusted slope is equivalent to the given slope, we can use the reflexivity of the equivalence relation in the case that the bound is zero.

A second adjustment we had to make, was for the step in the proof where A'Campo states that $-d_\lambda(t-1) \leq \lambda(tn) - t\lambda(n) \leq d_\lambda(t-1)$ for $\lambda : \mathcal{S}$ can be proven by induction on $t \in \mathbb{N}_+$. We proved this statement in Coq in `concentration_lemma_induction`. However, we had to make some slight adjustments for workability. First of all Coq cannot perform operations as addition or multiplication if one input is in `N` and the other is in `Z`. We could have chosen to prove the statement by induction on `t:N` and apply `N_to_Z` to every occurrence of `t` in the statement to map `t` from `N` to `Z`, possibly in the form of a coercion as in Section 9.7. However, after proving this statement we want to apply it to `t = 3 * pr_bound l` which is a term in `Z`. This is not possible (at least not directly) if we prove this statement by induction on `t:N`. We could convert `3 * pr_bound l` to `N` and then apply the statement, but we went for another option. We chose to take `t:Z`, add the condition `0 < t` and use induction on `t:Z`. This does also require a small bit of extra work, since in the case that `t = 0` or `t ≤ 0`, we have to prove a contradiction with the assumption that `0 < t`.

9.3 Transparent and opaque proofs

We use two ways to end a proof or definition started by `Proof`, which are `Defined` and `Qed`. The first option makes the proof transparent, which enables us to unfold the proof or definition later on. The second makes the proof opaque, so that we cannot unfold it later. Hence, `Qed` is used when it is enough to know that a term of a certain type exists without knowing what this term actually is. An example would be the type which states that addition on the Eudoxus reals is commutative. The style guide of the HoTT library advises for performance reasons to make declarations opaque if possible [31].

If a proof is transparent, certain tactics are to be avoided since they take away too much control over the proof and the terms they are applied to [31]. The style guide contains a list of tactics which are allowed in transparent proofs.

A noteworthy absence on this list is the tactic `rewrite`. We often use this tactic to manipulate terms, since it enables us to replace `a` by `b` if `a = b`. Even though applying `rewrite` gives therefore a predictable result, the library prefers to see this result achieved by smaller tactics or lemmas [31]. There exist two main solutions to the appearance of `rewrite` in transparent proofs. Firstly, we can avoid `rewrite` by using smaller steps as `ap` and `concat`. Secondly, we can make the proof, or part of the proof opaque. Small parts of the proof can be made opaque by the use of `abstract`, but this has as disadvantage that this part of the proof should then be executed at once and we cannot see intermediate results anymore.

We can also avoid the use of `rewrite` in transparent definitions by splitting the definition in multiple proofs, where we use `rewrite` only in the proofs we can make opaque. An example of this is `plus_slopes`, where we define addition for slopes. We cannot make this definition opaque as later on we will need to know how addition is defined to be able to work with it. However, as the output of `plus_slopes` is in `slopes`, we will not only need to output a function `Z -> Z` but also prove it is a slope. We may want to know the resulting bound of the function, but we do not necessarily need to know the exact formulation of the proof why this is indeed a bound. This part of the definition is exactly the part where we would want to use `rewrite`. Hence to define addition on slopes, we first define the function and its bound by

```
Definition plus_slopes_function (k l : slopes) : Z->Z
  := fun n  => (pr_slope k n + pr_slope l n).

Definition plus_slopes_bound ( k l : slopes) : Z
  := pr_bound k + pr_bound l.
```

and then we prove that `plus_slopes_bound` is indeed a bound to `plus_slopes_function` in

```
Lemma isbound_plus_slopes ( k l : slopes)
  : IsBound (plus_slopes_function k l) (plus_slopes_bound k l).
Proof.
  [...]
```

```
Qed.
```

Splitting up the definition of addition like this enables us to make the definitions `plus_slopes_function` and `plus_slopes_bound` transparent and the lemma `IsBound_plus_slopes` opaque. Therefore we can use `rewrite` in `IsBound_plus_slopes`, the place where we actually need it. Then we can define the actual addition as

```
Global Instance plus_slopes : Plus slopes
  := (fun k l => ((plus_slopes_function k l)
  ; ((plus_slopes_bound k l) ; (IsBound_plus_slopes k l)))).
```

Since for all operations with slopes we need to prove that the stated bound is indeed a bound, we follow this line of definitions for all other operations on slopes. Sometimes we will however also define a `helpfunction` if we cannot state the definition of the `function` immediately or if this simplifies the definitions. An example is the absolute value function, where we take the odd extension in `abs_slopes_function` of the function we defined in `abs_slopes_helpfunction`.

```
Definition abs_slopes_helpfunction (l : slopes) : Z->Z
  := (fun n => (int_abs_Z (pr_slope l n))).

Definition abs_slopes_function ( l : slopes) : Z -> Z
  := odd_extension (abs_slopes_helpfunction l).
```

The only exception to this format of defining an operation on slopes is the multiplicative inverse, which we will discuss in the next section.

9.4 Multiplicative inverse

Since we saw in Lemma 5.53 that the multiplicative inverse only exists merely for slope, we cannot define it exactly as operations such as addition and multiplication. Normally we namely define the function and its bound together with a proof that this is indeed a bound to the function, before defining the resulting slope. In this case however, the function and bound only exists merely so we need to change our approach in this case.

If we take a closer look at Lemma 5.53 and the proof of Mizrahi, we see that we need the truncation to be able to conclude from $0 < \lambda$ that there exists $m : \mathbb{N}_+$ such that $d_\lambda < \lambda(m)$ and similarly from $\lambda < 0$ that there exists $m : \mathbb{N}_+$ such that $\lambda(m) < -d_\lambda$. After this step, we do not use the truncation in the goal anymore to strip truncations in intermediate steps. Hence if we add the assumption that there exists such an $m : \mathbb{N}_+$, we can construct the multiplicative inverse without needing to truncate its existence.

We chose to only define the function and bound of a multiplicative inverse for positive slopes. For Eudoxus reals we then infer a multiplicative inverse for negative reals from the positive case. A first reason for this, is that we are ultimately interested in a division for the Eudoxus reals, and not necessarily for slopes. A practical reason for the choice to postpone the multiplicative inverse for negative entries, is that we work more easily with Eudoxus reals than with slopes because of all the results available in the HoTT library. Therefore we can derive the multiplicative inverse of a negative Eudoxus real from the positive case without developing extra theory. In the case of slopes we would for instance have to show in addition that $\kappa \cdot (-\lambda) \sim (-\kappa) \cdot \lambda$ for all $\kappa, \lambda : \mathcal{S}$, which for the Eudoxus reals is a result contained in the library.

As a last remark, Mizrahi assumes when defining the inverse that without loss of generality the original slope is well-adjusted and hence in particular odd. To avoid complicating the proof, we hence assume a slope to be odd when defining its inverse. Then when showing that for all positive slopes, there merely exists a multiplicative inverse, we define the inverse of a slope as the inverse of the odd slope equivalent to the original slope, which exists by Lemma 5.8.

Hence we define the multiplicative inverse of an odd slope for which there exists $m : \mathbb{N}_+$ such that $d_\lambda < \lambda(m)$ as follows.

```
Definition recip_slopes_pos_odd (l : slopes)
  (p: exists m, and (0<m) (pr_bound l < pr_slope l m) )
  (l_odd:IsOdd_slope l)
  : slopes
  := (recip_slopes_pos_function l p
    ; (recip_slopes_pos_bound l p
    ; isbound_recip_slopes_pos_odd l p l_odd)).
```

Then it is shown in a following lemma that if we have a positive slope, we have a mere multiplicative inverse.

```
Lemma recip_slopes_pos_exists (l:slopes) (p:0<l)
  : merely (exists k :slopes, equiv_slopes (l*k) 1).
```

9.5 Instances

One might have noticed that we defined addition on slopes in `plus_slopes` as an `Instance` of type `Plus slopes`, whereas we also could have defined it as a term of type `slopes -> slopes -> slopes` using `Definition`. The advantage of defining it as an `Instance` is that when Coq does a type class search it finds only instances and not definitions. We discuss two examples where this proves useful.

A first example is that if we write `k + l` to add two slopes `k l : slopes`, Coq will search for an instance of `Plus slopes`. Hence, if we have defined `plus_slopes` as an instance, Coq will recognize `k + l` as short for `plus_slopes k l`. If we have defined `plus_slopes` using `Definition` this gives an error and we then have to write `plus_slopes k l` instead of `k + l`.

A second example is that if we want to prove that `--a = a` for an Eudoxus real `a`, we can apply `negate_involutive` which proves this statement under the condition that we are working in a group. Hence if have defined an instance of `IsGroup e_reals`, the goal is immediately solved.

The lemma `negate_involutive` is just an example of all the results in the HoTT library. We only defined a field and commutative ring in Section 4, but the library also contains the definition for example of a semiring, monoid and group. For all of these definitions, the library has implemented results following from the definition. That the negation is involutive is for instance not a requirement for a group, but it is shown that this follows from the requirements in `Classes/theory/groups`. Similarly, there are also orderings defined with less requirements than a full pseudo semiring order for which also theory is developed in the library.

We can define an instance as a `Local Instance` or a `Global Instance`. In the first case, Coq only finds the instance in a type class search within the same section, module or file. Hence when a file is imported in another file, the local instances of the first file will not be found in a type class search in the second file. Their definitions will however be available outside the section, module or file it was defined. If we define an instance as a global instance, Coq also finds the instance outside the current module, section or file.

If we do not specify whether an instance is local or global, Coq will sometimes make it local and sometimes global [31]. Hence we follow the convention of the HoTT library to always specify this. In general, we define operations such as addition and multiplication as `Global Instance`s, since we want to use them in multiple files and frequently. An example of a `Local Instance` is the following, which states that the strict ordering on Eudoxus reals is a psuedo semiring order.

```
Local Instance pseudo_srorder_e_reals
  : PseudoSemiRingOrder lt_e_reals.
```

We did not define `pseudo_srorder_e_reals` as a `Global Instance`, since in the same file we prove that we have a *full* pseudo semiring order. The latter implies the former, so it suffices to make the following a `Global Instance`.

```
Global Instance full_pseudo_srorder_e_reals
  : FullPseudoSemiRingOrder le_e_reals lt_e_reals.
```

9.6 Induction

We often rely on the use of induction to prove statements for all $n : \mathbb{N}$ or $n : \mathbb{Z}$. However, sometimes we need to prove a slightly different statement with induction than we directly need to be able to prove the induction step. We discuss two examples here.

9.6.1 The multiplication of two nonnegative slopes

In the proof of Lemma 5.30 we showed that if we multiply two nonnegative slopes the result is again nonnegative. We showed this for two slopes $\lambda, \mu : \mathcal{S}$ with respective lower bounds S_λ and S_μ, by stating that we could prove by induction on S_μ that for all $n : \mathbb{N}$ we have $\min(S_\lambda, \lambda(S_\mu), \lambda(S_\mu + 1), \lambda(S_\mu + 2), \ldots, \lambda(-1)) \leq \lambda(\mu(n))$.

However, in Coq, at least in the HoTT library, there does not yet exist a function which produces the minimum of a decidable subset of $\mathbb{Z}$. Hence when we formalize this result we construct this value while applying induction. At first glance, this means that for all $m : \mathbb{Z}$ we want to prove that if $m \leq \mu(n)$ holds for all $n : \mathbb{N}$ that we then have $S(m) \leq \lambda(\mu(n))$ for all $n : \mathbb{N}$, where $S(m)$ is the largest integer such that $S(m) \leq S_\lambda$, $S(m) \leq \lambda(m)$, $S(m) \leq \lambda(m+1)$, $\ldots$, $S(m) \leq \lambda(-1)$. However, we will explain in this section why we cannot prove this statement by applying induction on m.

The induction principle for the integers includes three cases. In the case where m equals 0, the claim follows immediately from the proof that S_λ is a lower bound for λ. In the second case we have to prove the statement for $m+1$ given the induction hypothesis for m. However, if for all $n : \mathbb{N}$ it holds that $m+1 \leq \mu(n)$ then definitely $m \leq \mu(n)$ and we can apply the induction hypothesis. We finish the proof by the observation that $S(m) = S(m+1)$.

In the third case we have to prove the statement for $m-1$ given the induction hypothesis for m. We recall that the induction hypothesis states that if for all $n : \mathbb{N}$ it holds that $m \leq \mu(n)$, then for all $n : \mathbb{N}$ we have $S(m) \leq \lambda(\mu(n))$. Intuitively, to prove $S(m-1) \leq \lambda(\mu(n))$ for all $n : \mathbb{N}$, we would take $n : \mathbb{N}$ and decide whether $m \leq \mu(n)$ or $m - 1 = \mu(n)$. In the first case we would apply the induction hypothesis and note that $S(m-1) \leq S(m)$. In the second case we would prove that $\lambda(\mu(n))$ is bounded from below by $\min(S(m), \lambda(m-1))$, which is exactly $S(m-1)$.

However, we cannot apply the induction hypothesis in the first case, because it requires that for all $n : \mathbb{N}$ we have $m \leq \mu(n)$. We cannot prove this because we are only given that for all $n : \mathbb{N}$ we have $m - 1 \leq \mu(n)$.

Hence we need to adapt the statement we want to proof to assert that we can apply our induction hypothesis when we want to prove that $S(m) \leq \lambda(\mu(n))$ for all $n : \mathbb{N}$. We do so by moving the double occurrence of 'for all $n : \mathbb{N}$' to the front of the statement. Hence the statement becomes 'for all $m : \mathbb{Z}$ and all $n : \mathbb{N}$, if $m \leq \mu(n)$ holds then we have $S(m) \leq \lambda(\mu(n))$'. This is a statement we *can* prove, by the earlier mentioned case distinction between $m \leq \mu(n)$ or $m - 1 = \mu(n)$.

Then the statement that there exists a lower bound of $\lambda(\mu(n))$ for all $n : \mathbb{N}$, follows by applying the result to $m = S_\mu$. We know after all that for all $n : \mathbb{N}$ we have $S_\mu \leq \mu(n)$.

In Coq the intermediate statement is proven in `nonneg_mult_compat_slopes_helpfunction` and the final result in `nonneg_mult_compat_slopes`. The intermediate statement looks as follows.

```
Definition nonneg_mult_compat_slopes_helpfunction
  (k l : slopes) (k_nonneg : 0 ≤ k) (l_nonneg: 0 ≤ l)
  : forall m:Z, merely (exists S:Z,
  forall n:Z, 0≤n -> m ≤ pr_slope l n -> S ≤ pr_slope k (pr_slope l n)).
```

Note the possible confusion, since this discussion was about λ and μ and the function refers to `k` and `l`, where λ and `l` do not fulfill the same role. A second small difference is here the occurrence of merely to match the truncated inputs `0 ≤ k` and `0 ≤ l`.

9.6.2 Defining a rational slope

Another example where we have to be careful with the statement we apply induction to is the definition of rational slopes. In Definition 5.2 we defined a positive rational slope as a slope $\lambda : \mathcal{S}$ for which there exists $p, q : \mathbb{Z}$ such that $p > 0$, $q > 0$ and $\lambda(n) = \min\{k : \mathbb{N} | qk \geq pn\}$ for $n > 0$. Again, we have to construct the minimum ourselves in Coq, but we will do so for the case where the condition is $qk \geq p$. The right-hand side here is p instead of pn, since we will later vary p to be pn for different values of $n > 0$. Just as in the previous section, we first implement the following definition which proves a slightly different statement.

```
Definition qslopes_helpfunction (p q:Z) (p_pos:0<p) (q_pos:0<q)
  : forall n:Z, 0≤n -> exists M:Z, and (p ≤ q*M) (and (M<p -> p ≤ q*n)
  (forall k : Z, 0≤k≤n -> p≤q*k -> M ≤ k)).
```

We see that this function constructs for all $n : \mathbb{N}$ a value $M : \mathbb{Z}$, where unfortunately n does not fulfill the same role as before this definition. The idea is that M is the minimal value k in the interval $[0, n]$ satisfying $p \leq qk$. If the interval does not contain an M such that $p \leq qM$, then we set M to be p since $p \leq qp$. We capture this idea by stating three conditions on M. The first states that this M satisfies the condition $p \leq qM$. The third condition guarantees that M is indeed the minimal value in the interval $[0, n]$ if the interval contains an M such that $p \leq qM$. The second condition tells us that if M is not equal to p, that then $p \leq qn$. This condition is not necessary to capture that M is the minimum, but we use it

to prove the other two conditions when applying induction on n. We use it for instance in combination with the third condition to conclude that if $M < p$ that then $M \leq n$.

We first limit ourselves to define the minimum in the interval $[0, n]$, because this enables us to define M inductively. The three requirements together are strong enough to allow us to prove the induction step. If we would have tried to immediately construct M such that $p \leq qM$ and such that for all $k : \mathbb{Z}$ satisfying $p \leq qk$ that it holds that $M \leq k$, the second requirement would have forced us in general to define M as 0 but this would have been in contradiction with the first.

Since we know that the minimum is at most p, it follows that $\min\{k : \mathbb{N} | qk \geq p\}$ equals $\min\{k : \mathbb{N} | (k \leq p) \times (qk \geq p)\}$. Hence we define $\min\{k : \mathbb{N} | qk \geq p\}$ as the first projection of `qslopes_helpfunction` evaluated in p. Then we need to state `lt_le _ _ p_pos` as the last argument, since `lt_le` infers from `0 < p` that `0 ≤ p`.

```
Definition Qpos_slopes_function (p q:Z) (p_pos:0<p) (q_pos:0<q)
  := pr1 (qslopes_helpfunction p q p_pos q_pos p (lt_le _ _ p_pos)).
```

Then we can define a positive rational slope as an odd slope of which the associated function equals $n \mapsto \min\{k : \mathbb{N} | qk \geq pn\}$. Here we insert `pos_mult_compat _ _ p_pos n_pos` as argument since we need a proof that `p*n` is positive when both `p` and `n` are positive.

```
Definition IsPosRationalSlope (l : slopes)
  := and (IsOdd_slope l)
  (exists (p q:Z) (p_pos:0<p) (q_pos:0<q),
  forall n:Z, forall n_pos: 0<n,
  pr_slope l n =
  Qpos_slopes_function (p*n) q (pos_mult_compat _ _ p_pos n_pos) q_pos).
```

Then we define a slope to be rational if it consists of the zero function or if it a positive rational slope, or if its negation is a positive rational slope.

```
Definition IsRationalSlope (l : slopes)
  := sum (pr_slope l = fun _ :Z => 0)
  (sum (IsPosRationalSlope l) (IsPosRationalSlope (-l))).
```

This is slightly different from Definition 5.2, since we simplified the $p = 0$ case to the associated function being the zero function. At first sight the inclusion of a negative slope also differs, but if the slope is a rational slope with $p < 0$ then the negation of the slope is a rational slope with $-p > 0$. Hence this does not cause a difference between the definitions.

Then we can finally define the types of (positive) rational slopes.

```
Definition Qpos_slopes := exists (l:slopes), IsPosRationalSlope l.

Definition Qslopes := exists (l:slopes), IsRationalSlope l.
```

9.7 Coercion

When we are given a rational slope, we want to be able to work with it as if it were a slope and we will see in this section that coercions enable this. More precisely, even though we defined operations as addition and multiplication on the type of slopes, we also want to be able to apply these operations to rational slopes. We could, if `l : Qslopes`, perform these with `pr1 l`, where `pr1` is the projection to the first element of a dependent pair. However, for convenience Coq allows us to work with `l` itself by the use of implicit coercions. Then if Coq finds a term which is not well typed, it searches for (a combination of) available coercions that make the term well typed.

This entails that we should first define a coercion `Qslopes -> slopes` before we can work with rational slopes as if they are just slopes. However, for this we can just take `pr1`, because this maps a term in `Qslopes` to a term of `slopes`. Hence we first define a function `Qslopes_to_slopes` of type `Qslopes -> slopes`.

```
Definition Qslopes_to_slopes : Qslopes -> slopes := pr1.
```

Then to state that his is a coercion we use the combination of `Coercion` and `>->` as follows.

```
Coercion Qslopes_to_slopes : Qslopes >-> slopes.
```

After stating this coercion, we are able to write `k + l` where `k : slopes` and `l: Qslopes`. We cannot however write `l + k`, since then Coq searches for a coercion from `slopes` to `Qslopes` which does not exist. In such cases we have to write `plus_slopes l k`, because by the type of `plus_slopes` Coq knows it has to search for a coercion from `Qslopes` to `slopes`.

In a similar way we also define a coercion from `Qpos_slopes` to `Qslopes`. Then we can write `k + m` where `k : slopes` and `m: Qpos_slopes`.

9.8 Implicit arguments

When a declaration requires multiple arguments, Coq can sometimes infer arguments from other arguments. Hence we do not need to explicitly state these arguments, and we can define them as so-called implicit arguments.

To see how this works, we start with an example of the identity function. It is defined in the HoTT library in `Classes/interfaces/canonical_names` as follows.

```
Definition id {A : Type} (a : A) := a.
```

When we want to apply `id` to some element `a` of a type `A`, we write `id a` and not `id A a`. The reason for this lies in the use of the brackets in `{A : Type}` and `(a : A)`. The curly brackets namely indicate a implicit argument and the round brackets an explicit argument. This means that Coq infers the implicit argument `A` from `a` when we write `id a`. If we want to specify an implicit argument explicitly we can do so by making all implicits explicit by prefixing `@` as for example `@id A a`. If we want to specify just one implicit we can use `:=` to write for example `id (A:= A) a`.

There are two forms of implicit arguments: maximally and non-maximally inserted. The first is denoted by curly brackets as in the definition of the identity function and the second is denoted by square brackets as `[A: Type]`. The difference is visible when we only partially apply a function. Coq namely inserts non-maximally inserted implicit arguments only before an argument we give explicitly. On the other hand maximally inserted implicit arguments are always inserted by Coq if they are the next argument to insert [15]. If we go back to the example of our identity function, Coq considers the argument `A` inserted if we check the type of `id`, but not if we would have used square brackets.

In our implementation we only use explicit arguments in definitions, lemmas and other declarations. The only exceptions are the context, which we will discuss in the next section, and the definition of slopes. In the latter we include the implicit argument `` `{Integers Z} `` to make sure we work with slopes as functions from and to the integers. Without this implicit argument, the type `slopes` takes as arguments a type `Z` together with elements of `Plus Z`, `Negate Z` and `Le Z`. Hence when we have defined addition, negation and the non-strict ordering on `slopes`, the type `slopes` itself satisfies these arguments. Therefore at that point, Coq states that the type of `slopes` is the type `{l : slopes -> slopes & IsSlope l}` instead of `{l : Z -> Z & IsSlope l}`. We will see in the next section why we used the `\` before this argument.

9.9 Sections and context

A further use of implicit arguments is that we can use them to add assumptions. This can be done within a section, which we open by `Section contents` and close by `End contents` if our section is called `contents`. In a section we have multiple options to make assumptions such as `Context`, `Hypothesis` and `Let`. We will however only use `Context` since it enables us to use implicit generalization in our assumptions [16].

An example without implicit generalization is if we for instance want to add the assumption that `A` denotes a type. Then we can write `Context {A : Type}` after we opened the section, which adds `A : Type` to the context.

Coq disables implicit generalization by default [15]. Hence if we want that Coq implicitly generalizes `A` in our next example, we first need to specify that Coq is allowed to do so. This means that we should declare it as a generalizable variable by the use of `Generalizable Variable A`.

Then we can write `Context `{Naturals A}` to add the assumption that `A` is a construction of the naturals numbers. Hence we do not only add the assumption `H : Naturals A` to the context, but also an apartness relation, addition, multiplication and all other arguments for the typeclass `Naturals` which is defined in `Classes/interfaces/naturals`. The extra `` ` `` indicates the use of implicit generalization here, which enables Coq to generate and insert these implicit arguments, such as addition [38].

We can also extend our assumptions for certain parts of our context. A construction of the natural numbers does for instance not require the apartness relation to be trivial, which is the requirement that

two elements are apart if and only if they are not equal. We can add the assumption that we have a trivial apartness relation on our natural numbers by `Context `{TrivialApart A}`. However, this assumption will add a new apartness relation on `A` to the context and an assumption that this new apartness relation is trivial. If we do not want to add a new apartness relation, but only assume that the apartness relation we already had is trivial, we write instead `Context `{!TrivialApart Z}`. The extra `!` ensures that Coq does an instance search and hence finds the earlier defined apartness relation.

If we want to be able to explicitly refer to the proof that `A` is a construction of the naturals we write `Context `{H:Naturals A}`. Then `H` of type `Naturals A` is added to the context.

9.9.1 Our context

Since we are adding assumptions when we use `Context`, we should be careful that we do not add statements which are not true or cannot be proven. In our implementation we have (a subset of) the following declarations at the beginning of a section after generalizing `N` and `Z`.

```
Context `{Funext} `{Univalence}.
Context {Z: Type} {Zap: Apart Z}
  {Zplus : Plus Z} {Zmult : Mult Z}
  {Zzero : Zero Z} {Zone : One Z} {Zneg : Negate Z}
  {Zle : Le Z} {Zlt : Lt Z} {ZtoRing : IntegersToRing Z}.
Context {Zint : Integers Z} `{Naturals N}.
Context `{!TrivialApart Z} `{!IsApart Z}.
Context `{Meet Z} `{Join Z} `{!LatticeOrder Zle}.
```

The first line adds univalence and functional extensionality, since they are implemented in Coq as typeclasses. The next five lines assumes that `N` and `Z` are constructions of the natural numbers and integers respectively. We list all the operations on `Z` explicitly to be able to refer to `Zle` later on without the need to use the automatically generated name by Coq. The next line adds the assumptions that the apartness function on `Z` is a trivial apartness relation, which means that the relation satisfies the requirements of Section 5.8, and is equivalent to not being equal. These two assumptions are not required for a type to be an implementation of the integers. However, we can reasonably add these assumptions since the implementation `natpair_integers` shows that we can define an apartness relation on the integers that is trivial if we have a trivial apartness relation on the naturals[17]. This latter statement is shown in the `peano_naturals` implementation of the naturals [18]. The last line adds the assumptions that we have a meet and join with respect to `Zle` such as in Section 5.10. This could be implemented since the non-strict ordering is decidable on the integers.

Hence we only add assumptions to the context which could also be constructed.

9.10 Structure of the Coq files

In this section we will shortly discuss what files our formalization contains and what results they contain. We organized our implementation in Coq by creating multiple files, all dedicated to a certain topic. We tried to keep our results on slopes and Eudoxus reals separated. Of course, we often need statements about slopes when we want to prove something for Eudoxus reals, but the files about slopes do not refer to the Eudoxus reals.

First of all we have the file `int_orders` which contains lemmas on the integers, which were not available in the HoTT library. These mostly concern the ordering on the integers, but also the related absolute value, meet and join.

Then we take a look at files concerning slopes. We first define slopes in the file `slopes`, which also includes the definition of odd and well-adjusted slopes. Furthermore, we define here the operations from Section 5.3 such as addition and multiplication. To show that these operations are commutative or have an inverse, we often use the concentration lemma. Hence in `concentration` we prove the concentration lemma where we use properties of the optimal Euclidean divisor from `oed`. We also show some results related to slopes defined as an odd extension. In `slopes_ringlike` we are then able to show that slopes are a commutative ring up to equivalence. Without the need for 'up to equivalence' we implement the slopes as a semigroup with respect to multiplication, and as a monoid and commutative semigroup with respect to addition.

[17] The apartness relation is constructed in `Z_is_apart'` and the triviality is established in `Z_trivial_apart'`. Both lemmas can be found in `HoTT/Classes/implementations/natpair_integers`.

[18] See `nat_trivial_apart` in `HoTT/Classes/implementations/peano_naturals`.

Next we concern ourselves with the ordering on slopes. We first define in `qslopes` the rational slopes so that in `order_slopes` we can define the orderings on slopes and prove most of the statements of Section 5.4. We also state in `order_slopes` the Archimedean property and the cotransitivity of the strict ordering. Since we needed an ordering to be able to define the multiplicative inverse of a slope, this is also the file where we define this operation. Here also a few lemmas of Mizrahi can be found related to the ordering of slopes.

In the last two files on slopes, which are `abs_slopes` and `lattice_slopes`, we define respectively the absolute value, and join and meet on slopes. We also prove in these files some properties of the operations.

Then we can look at the files concerning the Eudoxus reals. First of all we have a file `ereals` which contains the definition of the Eudoxus reals, proves that this is a ring and defines the absolute value. We cannot continue with showing that it is a field, without defining an order which we hence do in `order_ereals`. In this file we show that the non-strict ordering is a so-called `SemiRingOrder` and the strict order is a `StrictOrder`, which are both weaker notions than a full pseudo semiring order. We also show here the Archimedean property, with the remark that we have not yet implemented the rationals, but for the status of the formalization we refer to Section 9.11. Results on the absolute value in combination with orderings are also proven in this file, just as that $0 < 1$ in the type of Eudoxus reals.

In `apart_ereals` we then define the apartness relation on the Eudoxus reals and in `lattice_ereals` we construct the meet and join. We also show in the latter how the absolute value relates to the join. Next in `full_order_ereals` we prove that we have a full pseudo semiring order, which includes proving the binary extensionality of multiplication from Section 5.11. We also define here the multiplicative inverse since its definition uses the result that the strict ordering on the Eudoxus reals is in particular a strict semiring order. This result is used to construct the inverse of a negative real from the inverse of a positive real, as already mentioned in Section 9.4. In the final file, `field_ereals` we finally conclude that the Eudoxus reals are an ordered field.

9.11 Status of the formalization

In the Coq files we prove that the Eudoxus reals are an ordered field, but due to time constraints we have a few gaps. A lemma or other declaration can namely also be closed by `Admitted` instead of `Defined` or `Qed`. This adds the declaration to the context without having to prove it. This introduces some weak points in the proof that the Eudoxus reals are indeed an ordered field, but we fully expect to be able to prove the statements which are now admitted. We already indicated in Section 5 which results were formalized, but here we will gather this information and give an overview of what elements of the proof are missing.

The first file in which proofs are missing is the `oed` file, in which the definition of the optimal euclidean divisor is stated together with some properties. However, the optimal euclidean divisor is not defined and neither are its properties proven. There are two main reasons for leaving this part unproven. The first is that the Euclidean divisor is not yet implemented in the HoTT library, so that we could not base the construction of the *optimal* Euclidean divisor on this. The second is that Mizrahi [35, Sec. 4.2] gives detailed proofs of these properties and they just use standard properties of the integers, so we expect no problems in implementing these proofs.

In `order_slopes` we have also admitted three statements and a consequence from Mizrahi [35, Pro. 4.50]. Two of these are used in the construction of the multiplicative inverse and the other one is used to bound a slope pointwise if it is apart from zero. Again, Mizrahi gives a detailed proof of all these statements and we do not expects problems when formalizing these statements in homotopy type theory.

The last remaining proofs are related to the rationals, since we already omitted in Section 5.12 the full proof that our definition of $\mathbb{Q}$ is indeed a construction of the rational numbers. This means that we cannot use Definition 5.68 to simplify the dealings with rational slopes and in particular we cannot use the decidability of the orderings for these slopes. Hence we did not prove any of the results related to sequences from Section 5.6, and we did not formalize Lemma 5.44 and 5.45 about the Archimedean property for slopes. This implies that we also were not able to show the cotransitivity in the rational case from Lemma 5.49 and we were not able to prove Lemma 5.51 which states that if $\neg(\mu < \lambda)$ we have $\lambda \leq \mu$ for all $\lambda, \mu : \mathcal{S}$.

Since we also have not defined the rationals as a subset of the Eudoxus reals, we were not able to state the Archimedean property for the Eudoxus reals using the rationals. Hence we proved the Archimedean property except that we stated that the intermediate value was a real and not necessarily a rational. That is, we stated that if $a < b$ for $a, b : \mathcal{E}$ that we could construct $c : \mathcal{E}$ such that $a < c < b$ instead

of $c : \mathbb{Q}$. Once we have defined the rationals as subset of the Eudoxus reals, we should be able to easily modify the proof to construct $c : \mathbb{Q}$ since in the current proof the constructed c is the image of a rational slope under classof. For now, we indicate the lack of the rationals in the formulation by adding `notrat` to the name of the Archimedean property. Hence the Archimedean property of the Eudoxus reals looks in Coq as follows.

```
Lemma arch_e_reals_notrat ( x y : e_reals)
  : x<y -> merely (exists z:e_reals, x<z<y).
```

The last proof that is missing is located in `full_order_ereals` and states that if an Eudoxus real is positive, its multiplicative inverse is also positive. In the library this is shown in `pos_recip_compat` but this requires that we already have proven that the Eudoxus reals are an ordered field. Other than that, the simplest proof would probably use that multiplying both sides of a strict inequality with a positive real reflects the strict ordering. That is, if $a : \mathcal{E}$ and we know $0 < a$, then $0 < 1/a$ follows from $0 \cdot a < (1/a) \cdot a$ which is equivalent to $0 < 1$. However, this property of reflecting the order is only included in the library once we have shown that the ordering on the Eudoxus reals is a full pseudo semiring order, which is (indirectly) exactly what we are trying to show. Hence we could try to show this result for slopes. Easier is however to note that we only use this result for rational numbers. Therefore once we have constructed the rationals as subset of the Eudoxus reals, we should rephrase this lemma to concern only rationals and then this lemma follows easily.

10 Future work

In Section 7 we concluded that the completion of the Eudoxus reals is an initial Cauchy complete Archimedean ordered field and in Section 9 we discussed our formalization of this result in Coq. These results were however both not complete, so in this section we will give an overview of what still remains to be done.

$\mathbb{Q}$ is a construction of the rationals In Section 5.12 we explained why the type we referred to as the rational numbers and denoted by $\mathbb{Q}$ was indeed a construction of the rationals. We did however not prove this in full detail and so this remains to be done.

Elements of the formalization of Section 5 In Section 9.11 we already discussed in detail all elements missing in the Coq formalization. However, in short, next to the results related to the rationals, also the construction and properties of the optimal euclidean divisor are not formalized, just as three statements from Mizrahi. Other than these results, Section 5 is completely formalized in Coq.

The completion of the Eudoxus reals is an Archimedean ordered field We used Gilbert's work on premetric spaces to conclude that the completion of the Eudoxus reals is Cauchy complete. However, we have to prove ourselves that our completion is an Archimedean ordered field. In order to show this, we expect that we can adapt the construction of the HoTT book and Gilbert for the rationals to the Eudoxus reals.

We explained this in Section 7.3, but the main idea is that we can define all the operations on the completion as Lipschitz extensions and that hence identities for the Eudoxus reals transfer to its completion. We can additionally define the strict ordering such that the Archimedean property of the Eudoxus reals immediately implies that its completion is Archimedean.

When the proofs of Gilbert and the HoTT book are adapted to the Eudoxus reals, it would also be interesting to see whether the algebraic structure on a completion can be defined in the case of more general underlying types. In particular it would be interesting to see whether there exists a sufficient requirement for a type for its completion to be an Archimedean ordered field.

The formalization of Section 7 We formalized none of the results of Section 7, but we expect that a future formalization can be based on the formalization of the Cauchy reals by Gilbert. An exception is the proof that the Eudoxus reals are an *initial* Cauchy complete Archimedean ordered field, since Gilbert has not formalized this for the Cauchy reals and we have defined our embedding differently. We can however use Gilbert's formalized result that any Cauchy complete Archimedean ordered field can be endowed with a closeness relation to form a premetric space.

11 Conclusion

The main goal of this thesis was to gain a better understanding of higher inductive types. In order to accomplish this, we initially tried to construct the real numbers as a higher inductive type but later used instead a higher inductive-inductive type.

In our first approach we used a quotient type to construct the Eudoxus reals from the type of slopes. Since a quotient type is set truncated we only needed to work with paths between terms, and not higher order paths. Moreover, since a lot of statements we proved were mere propositions, path constructors were always respected when using the induction principle. Only when we defined operations on the Eudoxus reals as addition, we needed to take the path constructors explicitly into account. In such cases we used a path constructor to reduce the required equality of Eudoxus reals to an equivalences of slopes. At this point we could often follow the reasoning of A'Campo in his proofs from set theory.

A significant difference arose between our construction and A'Campo's when we considered the completeness of the Eudoxus reals. We were namely not able to replicate his proof of the least-upper-bound-property because we did not want to use the law of excluded middle. Instead we proved that the type of slopes is Cauchy complete. The HoTT book stated that we would need the axiom of choice to be able to deduce from this that the Eudoxus reals were Cauchy complete. We indeed ran into this problem when we wanted to lift sequences of Eudoxus reals to slopes, because we only knew the mere existence of a representative slope for one Eudoxus real at a time.

Hence we used a higher inductive-inductive type to define the completion of the Eudoxus reals. We concluded that it was an initial Cauchy complete Archimedean ordered field and that it was equal to the Cauchy reals. While working with this type we did not directly use the induction principle, since it implied an uniqueness result of continuous functions defined on a completion. We used this result to reduce equalities involving continuous functions on the completion, to equalities on the Eudoxus reals itself. Hence even though we proved statements for the completion, we often only had to take elements of the Eudoxus reals into account and not the limit points.

The HoTT book also used this completion to construct the Cauchy reals, and stated that this construction was preferable over other approaches. We did not explicitly compare these approaches, but we did see some of its advantages. We were namely able to conclude that the completion was Cauchy complete without the need to use the axiom of choice, which was not possible when we worked with the quotient type. We also did not have to work with an equivalence relation, as would have been necessary when working with setoids. A more explicit comparison could be made in future work.

We end this conclusion with an aspect of the completion of the Eudoxus reals we do not yet fully grasp. We namely know that in the classical setting the Eudoxus reals are Cauchy complete and that A'Campo states for example how we can construct a slope representing $\sqrt{2}$. We do however not know what kind of elements the completion adds compared to the Eudoxus reals or the type of slopes. In particular, we do not fully understand whether these extra elements enable us to derive the Cauchy completeness of the completion or whether it is its associated induction principle.

A Tactics in Coq

In this section we will explain the tactics we used in the formalization. Some are part of Coq itself and others are defined in the HoTT library, but we do not make a distinction here. Since we did not use all tactics available, this list is not (nearly) complete. A complete list of all tactics in Coq can be found at `https://coq.inria.fr/refman/coq-tacindex.html`, but we are not aware of such a list for tactics defined in the HoTT library.

We do not intend to fully explain all possibilities of the tactics listed or explain their inner workings. Instead we try to give a new user of Coq an informal starting point with what we hope are useful examples. In Section A.1 we additionally state a few options to have more control over the used tactics.

`apply` If `E: A -> G` and the goal is `G`, then `apply E` replaces the goal by `A`. It also works if the result has more premises, such as `E : A -> B -> G`, then for each premise `A` and `B` a goal is generated. The error `Unable to find an instance for the variables` appears if Coq cannot find an instance to apply the theorem with so that it cannot generate goals accordingly. An example is when we use some transitivity property without specifying with what 'intermediate value'. In these cases we can write `apply E with a` or, when more instances are missing, `apply E with a b c`. We also mention in this appendix the variation `rapply`, but other variations can be found in the style guide of the HoTT library [31].

`assert` If we want to prove an intermediate result `H` we can write `assert H` or `assert (E:H)` when we want to name the term. Then a goal `H` is added as first goal to prove. If the proof of `H` is short, we can opt to write `assert H by ([...])`, where we state instead of `[...]` the proof as just one command, possibly using `;` and `,` from Section A.1. Then the goal `H` is not generated, but immediately added to the local context since we have already proven it. This tactic is similar to `cut`.

`auto` This is a tactic which can solve easy goals as `G1 * G2` if `E1 : G1` and `E2 : G2` are in the context. Note that the tactic only solves easy type-theoretic goals and does for instance not know that if we add 1 to a positive integer, the result is positive. This tactic solves more goals than `trivial`.

`case` If `E : A+B` then `case E` corresponds to making a case distinction. Hence Coq replaces the current goal `G` by a goal `A -> G` and a goal `B -> G`.

`change` If `A` and `B` are definitionally equal, then `change A with B` replaces any occurance of `A` in the goal by `B`. An example is for instance `change (pr_bound l) with (pr_bound (-l))`, since the bound of a slope `-l` is defined as `pr_bound l`.

`contradiction` If in the context we have a term `H :~E`, we can write `contradiction H` or just `contradiction` to replace the current goal by `E`. This can also be applied to `A <> B` since `A <> B` is defined as `~(A = B)`. In simple cases such as `a <> a` no new goal is generated as Coq then automatically finishes the proof.

`cut` If the current goal is `G` and we write `cut E`, Coq replaces the current goal by a goal `E -> G` and a goal `E`. Especially in small cases, also `assert` can be used.

`destruct` If `E` is an inductive type and we write `destruct E`, Coq replicates the current goal `G` for each constructor of `E`. Hence if `E : A + B`, Coq generates a goal `G` where `E : A` and a goal `G` where `E : B`. We can combine this with an introduction pattern as `destruct E as [E1 | E2]`. Then `E1 : A` and `E2 : B`. Since `E1` and `E2` occur for different subgoals they can also both be named `E1` or even `E`. If `E : A & B`, where we have just one constructor, then Coq does not replicate the goal but replaces `E : A & B` by `proj1_sig : A` and `proj2_sig : B`. We can again name these terms ourselves, but now we have to use the introduction pattern of a dependent pair as `destruct E as (E1, E2)`.

`exact` If the goal is `G` and we know of a term `g:G` we can write `exact g` to solve the goal. An extension to this is `refine`.

`exists` If our goal is to construct an element of a dependent pair defined by `B : A -> Type`, we can write `exists a` if `a : A` to change the goal to `B(a)`

`eexists` This tactic works the same as `exists`, but if it cannot find an instantiation of an object it creates an extra goal for this, instead of giving an error as `exists` would. This can for instance come up if we write `exists (E _)` and `E` Coq cannot find the hypothesis we denoted by `_` in the context.

`hnf` The `hnf` tactic reduces the goal to a standard form. If we for instance have to prove that `LeftIdentity (.*.) mon_unit`, the tactic `hnf` will reduce this to `forall y : A, mon_unit * y = y`, where the type `A` is inferred from the operation and `mon_unit`.

`induction` If `a : A` where `A` is an inductive type, then `induction a` will apply the induction principle associated to `A`.

`intro` If the goal is of the form `forall g : G1, G2 g` or `G1 -> G2` then writing `intro` will introduce an element `g:G1` and replace the goal by respectively `G2 g` or `G2`. We can name the to be introduced variable `E` by writing `intro E`. An extension to this is `intros`.

`intros` This tactic repeats `intro` until there are no variables left to introduce. If we want to introduce for instance four variables and only name the first and last we can write `intros E1 ?? E2`. If the variable we will introduce is a term of `A + B`, we can write `intros [E1|E2]` to duplicate the current goal into one where we have an element `E1 : A` in the local context and one where we have `E2 : B` in the local context. These names could also be the same. If the to be introduced variable is of type `a = b`, then writing `intros []` will replace all occurances of `b` in the goal by `a`.

`left` If the goal is of the form `A + B`, this tactic will replace the goal by `A`. See also the tactic `right`.

`pose proof` If we want to add a term `E : A` to the local context we can write `pose proof E`. We can name the term `a` by writing `pose proof E as a`.

`rapply` Works the same as `apply` but is less likely to give unification errors it should not have. More variations to apply can be found in the style guide of the HoTT library [31].

`refine` This tactic is like `exact`, but now we can also use holes in our term if needed. Hence we can write `refine (g _ _)` if `g _ _` is a term of the goal `G`. Coq will then replace the goal by the subgoals associated with the holes denoted by `_`.

`reflexivity` Solves a goal of the form `A = B` if `A` and `B` are definitionally equal. A special case is `A = A`. At least in the case of the integers, this also solves goals of the form `A`$\leq$`A`.

`revert` If we have `a:A` in the local context we write `revert a` to change a goal `G` to `A -> G`. This can for instance be useful to get the goal in a form to which Coq applies an induction principle as we want without repeating the complete goal in the induction principle.

`rewrite` If `E : a=b` then `rewrite E` will replace all occurrences of `a` in the goal by `b`. When we write `rewrite <- E`, all occurrences of `b` are replaced by `a`.

`right` If the goal is of the form `A + B`, this tactic will replace the goal by `B`. See also the tactic `left`.

`simpl` Simplifies the goal. If the goal for instance contains `pr1 (a & B a)` then this will be simplified to `a`.

`split` If the goal is of the form `A * B`, writing `split` replaces the goal by a goal `A` and a goal `B`. As a special case we can also use it to split goals which are typeclasses. An example is a goal `IsSemiGroup A`, which `split` replaces by `IsHSet A` and `Associative (.*.)`, which are exactly the requirements for `A` to be a semigroup.

`strip_truncations` This applies the induction principle for truncated types to terms in the context. Hence it removes the truncation in a term in the local context if the goal is sufficiently truncated. This is often easier than applying `Trunc_ind`, but might not work in more involved cases when Coq cannot infer that the goal is indeed truncated. We use this tactic often in combination with `apply tr` to remove the truncation in the goal afterwards.

`symmetry` If the goal is of the form `R a b` and `R` is symmetric, then `symmetry` changes the goal to `R b a`.

`transitivity` If the goal is of the form `R a b` and `R` is transitive, then `transitivity c` changes the goal to a goal `R a c` and a goal `R c b`.

`trivial` Solves easy goals, but less than `auto`. Examples of goals it solves are `Unit` and `A = A`, but also any goal appearing already in the local context. In contrast to `reflexivity` it does however not solve the goal `A`≤`A`.

`unfold` If a term `A` appearing in the goal is defined as `B`, writing `unfold A` will replace `A` by `B` in the goal.

A.1 Applying tactics

Next we list some options to specify how and where tactics should be applied.

However, we first want to shortly mention `Set Printing Parentheses` which does not seem to fit in this list, but has proven its worth. It makes Coq print all brackets, which is especially useful when working with associativity. This option can also be turned on in the settings of an IDE.

`all:` Applies the following tactic to all goals.

`in *|-, |-*, E` If we do not want to apply a tactic (just) to the goal, we can with `in` specify where to apply the tactic. Useful variations are `in *` to apply the tactic to the goal and all hypotheses, `in * |-` to apply the tactic to all hypotheses, `in E1,E2` to apply the tactic to hypotheses `E1` and `E2`, and `in E1,E2 |- *` to apply the tactic to the goal and hypotheses `E1` and `E2`.

`repeat` Repeats the following the tactic until it can no longer be applied.

`try` Coq tries to apply the following tactic, but does not give an error if it cannot apply the tactic successfully.

{} If a tactic generates multiple subgoals, we can write { to focus on the first subgoal. If we want to go instead to the 5th subgoal for instance we can write `5:{`. When we then have proven the subgoal, we unfocus it by }.

`,` If we want to apply a tactic to multiple terms we can write the tactic once and write the terms seperated by `,`. We can use this for instance to shorten `rewrite A. rewrite B` to `rewrite A,B`.

`;` The tactic to the right of the `;` is applied to each subgoal generated by the tactic on the left. If we for instance write `transitivity c` and then want to apply `trivial` to both goals generated by `transitivity`, we can write `transitivity c; trivial`.

`[ | ]` We use this in combination with `;` if we want to apply a certain tactic to the first subgoal and a different tactic to the second subgoal. An example would be `transitivity c; [apply E | trivial]`, where `apply E` is applied to the first goal and `trivial` to the second.

References

[1] Norbert A'Campo. A natural construction for the real numbers, 2003. ArXiv:math/0301015.

[2] Thorsten Altenkirch, Paolo Capriotti, Gabe Dijkstra, Nicolai Kraus, and Fredrik Nordvall Forsberg. Quotient inductive-inductive types. In *International Conference on Foundations of Software Science and Computation Structures*, pages 293–310. Springer, Cham, 2018.

[3] Thorsten Altenkirch and Ambrus Kaposi. Type theory in type theory using quotient inductive types. *ACM SIGPLAN Notices*, 51(1):18–29, 2016.

[4] Rob Arthan. The Eudoxus real numbers, 2004. ArXiv:math/0405454.

[5] Steve Awodey and Michael A Warren. Homotopy theoretic models of identity types. In *Mathematical proceedings of the Cambridge philosophical society*, volume 146, pages 45–55. Cambridge University Press, 2009.

[6] Andrej Bauer. Five stages of accepting constructive mathematics. *Bulletin of the American Mathematical Society*, 54(3):481–498, 2017.

[7] Andrej Bauer, Jason Gross, Peter LeFanu Lumsdaine, Michael Shulman, Matthieu Sozeau, and Bas Spitters. The HoTT library: a formalization of homotopy type theory in Coq. In *Proceedings of the 6th ACM SIGPLAN Conference on Certified Programs and Proofs*, pages 164–172, 2017.

[8] Yves Bertot. Private inductive types: Proposing a language extension. `https://coq.inria.fr/files/coq5_submission_3.pdf`, April 2013.

[9] Errett Bishop. *Foundations of constructive analysis*, volume 60. McGraw-Hill New York, 1967.

[10] Errett Bishop and Douglas Bridges. *Constructive analysis*, volume 279. Springer Science & Business Media, 2012.

[11] Auke Booij. *Analysis in Univalent Type Theory*. PhD thesis, University of Birmingham, 2020.

[12] Douglas S Bridges. Constructive mathematics: a foundation for computable analysis. *Theoretical computer science*, 219(1-2):95–109, 1999.

[13] Ulrik Buchholtz and Kuen-Bang Hou. Cellular cohomology in homotopy type theory. *Logical Methods in Computer Science*, Volume 16, Issue 2, June 2020.

[14] Evan Cavallo. Synthetic cohomology in homotopy type theory. Master's thesis, Carnegie Mellon University, 2015.

[15] Coq. Implicit arguments. `https://coq.inria.fr/refman/language/extensions/implicit-arguments.html`. Accessed: 26-02-2021.

[16] Coq. Implicit arguments. `https://coq.inria.fr/refman/language/core/sections.html`. Accessed: 26-02-2021.

[17] Coq. The module system. `https://coq.inria.fr/refman/language/core/modules.html`. Accessed: 02-03-2021.

[18] Coq. Variants and the match construct. `https://coq.inria.fr/refman/language/core/variants.html`. Accessed: 05-01-2021.

[19] Gabe Dijkstra. *Quotient inductive-inductive definitions*. PhD thesis, The University of Nottingham, 2017.

[20] James Douglas, Rony Kirollos, Ben Odgers, Ross Street, and Nguyen Hanh Vo. The efficient real numbers. `http://science.mq.edu.au/~street/EffR.pdf`, 2004. Accessed: 09-03-2021.

[21] Peter Dybjer and Anton Setzer. A finite axiomatization of inductive-recursive definitions. In *International Conference on Typed Lambda Calculi and Applications*, pages 129–146. Springer, 1999.

[22] Peter Dybjer and Anton Setzer. Indexed induction–recursion. *The Journal of Logic and Algebraic Programming*, 66(1):1–49, 2006.

[23] Fredrik Nordvall Forsberg. *Inductive-inductive definitions*. PhD thesis, Swansea University, 2013.

[24] Fredrik Nordvall Forsberg and Anton Setzer. A finite axiomatisation of inductive-inductive definitions. *Logic, Construction, Computation*, 3:259–287, 2012.

[25] Gaëtan Gilbert. Formalising real numbers in homotopy type theory. *Proceedings of the 6th ACM SIGPLAN Conference on Certified Programs and Proofs - CPP 2017*, 2017.

[26] Thomas Hales. A proof of the Kepler conjecture. *Annals of mathematics*, pages 1065–1185, 2005.

[27] John Harrison. Constructing the real numbers. In *Theorem Proving with the Real Numbers*, pages 13–38. Springer, London, 1st edition, 1998.

[28] Ambrus Kaposi and András Kovács. Signatures and induction principles for higher inductive-inductive types. *Logical Methods in Computer Science*, Volume 16, Issue 1, February 2020.

[29] András Kovács and Ambrus Kaposi. Large and infinitary quotient inductive-inductive types. *Proceedings of the 35th Annual ACM/IEEE Symposium on Logic in Computer Science*, May 2020.

[30] Robbert Krebbers and Bas Spitters. Type classes for efficient exact real arithmetic in Coq. *Logical Methods in Computer Science*, Volume 9, Issue 1, February 2013.

[31] The HoTT library. Conventions and style guide. `https://github.com/HoTT/HoTT/blob/master/STYLE.md`, 2020. Accessed: 08-01-2021.

[32] Dan Licata. Running circles around (in) your proof assistant; or, quotients that compute. `https://homotopytypetheory.org/2011/04/23/running-circles-around-in-your-proof-assistant/`, April 2011. Accessed: 03-02-2021.

[33] Peter LeFanu Lumsdaine and Mike Shulman. Semantics of higher inductive types, 2017. ArXiv:1705.07088.

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

[35] Leila Mizrahi. Thoroughly formalizing an uncommon construction of the real numbers. Master's thesis, University of Zurich, 2015.

[36] Russell OConnor. A monadic, functional implementation of real numbers. *Mathematical Structures in Computer Science*, 17(1):129–159, 2007.

[37] Walter Rudin. *Principles of mathematical analysis*, volume 3. McGraw-Hill, New York, 1964.

[38] Bas Spitters and Eelis Van Der Weegen. Type classes for mathematics in type theory. *Mathematical Structures in Computer Science*, 21(4):795–825, 2011.

[39] Ross Street. An efficient construction of the real numbers. *Gazette Australian Math. Soc*, 12:57–58, 1985.

[40] Ross Street. Update on the efficient reals. `http://web.science.mq.edu.au/~street/reals.pdf`, 2003. Accessed: 09-03-2021.

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

[42] Vladimir Voevodsky. A very short note on the homotopy λ-calculus. `https://www.math.ias.edu/~vladimir/Site3/Univalent_Foundations_files/Hlambda_short_current.pdf`, 2006. Accessed: 09-03-2021.

[43] Vladimir Voevodsky, Benedikt Ahrens, Daniel Grayson, et al. UniMath – a computer-checked library of univalent mathematics. `https://github.com/UniMath/UniMath`. Accessed: 18-12-2020.