# Learning how to implement Mathematical Proofs in Agda

Andrew Sneap - 1980030

MSc - Maths & Computer Science
Third Year Project 2020-2021
Supervised by Martín Escardó and Co-supervised by Todd Waugh Ambridge

**Abstract**

Proof assistants such as Agda are becoming increasingly popular, used to formalise existing proofs, and sometimes to verify the correctness of contentious proofs. Within the framework of dependent type theory, I construct the Integers, Rational numbers and Dedekind Reals, choosing to work constructively. I prove a number of properties and theorems of these sets, noting the differences when writing proofs without the use of proof assistants. Finally, I discuss how my implementation of the Dedekind Reals can be further developed, with the aim of proving that the Dedekind Reals form a complete archimedean field.

# Contents

# 1 Introduction

The goal of my project is to learn how to implement mathematical proofs in the dependently typed functional programming language Agda. The secondary goal of my project is to implement the Dedekind Real numbers in Agda. By working towards the second goal of the project, I will learn the techniques used to write mathematical proofs in Agda, taking note of the differences in approach and rigour that arise when writing proofs in a constructive type-theoretical setting. I build upon the work of Escardó's developmental Agda library, TypeTopology [5]. In this development, the Naturals Numbers have been defined, along with Addition and Order of Natural Numbers.

I begin by describing the relationship of some of the concepts used in Mathematics and Type Theory. I go on to formalise a number of properties and theorems of the Natural Numbers, including distributivity and Euclid's algorithm. I encounter surprising difficulty when attempting some of these proofs, and note the challenges involved in translating seemingly complete mathematical proofs. I move on to the Integers and Rational, proving properties of each, and note the difficulty of working with more complicated mathematical structures.

I then define the Dedekind Reals, and prove that that there is an embedding from the Rationals to the Dedekind Reals. I define the field axioms, and discuss the steps necessary to prove that the Dedekind Reals form a complete archimedian field, as well as potential extensions of the formalisation developed in this project.

# 2 Background

There are a number of proof assistants, including Coq, Lean and Agda. My supervisor suggested Agda as a suitable choice, and his strong background of univalent mathematics written in Agda was a valuable resource for me. At it's core, Agda is a dependently typed programming language, so the first step required me to develop my knowledge of dependent type theory, and more specifically Agda. I used a number of reading materials to accomplish this, including [1], [11] and [3]. I give here a brief description of the connections between concepts type theory and mathematics and logic, assuming basic knowledge of each. We write $x : A$ to mean that term $x$ has type $A$.

- Falsity can be encoded as the type that has no terms. Truth is encoded as the type that has one term. They are sometimes known as the Void and Unit types. Consider the ordered set $\mathbb{N}$, and the standard relation operator $<$. Then the type of $5 < 3$ has no elements, and the type of $3 < 5$ has a single unique element.

- The set $\mathbb{N}$ is a type with type constructors $zero : \mathbb{N}$ and $succ : \mathbb{N} \to \mathbb{N}$, so that terms of $\mathbb{N}$ are of the form $zero$, or $succ\ x$ for some $x : \mathbb{N}$. But every element has a type, including $\mathbb{N}$. We say that $\mathbb{N} : Set_0$, or in TypeTopology $\mathbb{N} : U_0$, where $U_0, Set_0$ are known as Universes. Universes are also terms, and follow a hierarchical structure; $U_0 : U_1$, $U_1 : U_2$. We cannot have $U_n : U_n$, as this is paradoxical and would allow us to prove anything.

- In classical mathematics, propositions are statements to be asserted as true or false, for example "5 is a prime number". As described by Escardó in [4], in type theory, propositions are types for which any two inhabitants are equal. When we write a proposition in type theory, we need to prove that it is indeed a proposition. This guarantees that any two proofs of a proposition are equal. In type theory, a full proof of a proposition consists of a proof that the proposition *is* a proposition, and an algorithm with produces an inhabitant of that proposition.

- The notion of equality is captured by an Identity type. Equality is a datatype relation of the form $\_ \equiv \_ : A \to A \to A$, with a single constructor $refl : (a : A) \to a \equiv a$. A complete description may be seen in [1], or [13]. Informally, this type says that each element $a : A$ is equal to itself. Much of challenge in proofs is constructing elements of this type.

- The logical language used in mathematics, for example "and", "or", "there exists", "if and only if" all have analogous representations within Type Theory. In this project, we choose to work constructively, which in particular is revealed through the use of the Sigma type, which in proofs requires us to produce a witness, and a proof that the witness satisfies the dependent function.

The highly recommended IDE for writing Agda code is Emacs. There is the availability of an extension in Visual Studio Code, but after a brief period using VS Code I switched to back to Emacs, encountering many highlighting issues.

Writing Agda code in Emacs is an enjoyable experience. One can write code interactively; by placing a '?' in place of a term, compiling the code replaces the ? with a hole. When writing proofs, one can write in ?'s liberally. Within the context of a hole, Emacs will tell you the type of the term it expects to be entered into the hole. Furthermore, for complicated holes that may be composed of multiple terms, one can write in partial terms, potentially with more ?'s, and refine the proof. For holes which have only one possible solution, by solving the constraints of a problem Agda can fill in the hole by itself. There is a rudimentary auto-solver, which can work for very simple proofs, but it is not sound, and can fill the whole with incorrect terms in some situations.

For the sake of brevity, and to respect word limits, the majority of proofs in this are omitted. I point out any proof already contained in Type Topology, and my proofs are all linked and pass the type check within Agda, unless otherwise stated. Note that the notation within this report is informal, and is not the same as the more rigorous notation of the Agda, which uses Unicode extensively.

# 3 Natural Numbers

When constructing proofs mathematically, we must have a system within which to work. We begin by stating the Peano Axioms, as described in [7], and [8]. We use 0 as in the initial element, as in [7].

**Definition 3.0.1** (Peano Axioms)**.** $\mathbb{N}$ is a set with the following properties

(P1) 0 is an element of $\mathbb{N}$

(P2) There exists a successor function $S : \mathbb{N} \to \mathbb{N}$

(P3) For all $m, n \in \mathbb{N}$, $m = n$ if and only if $S(n) = S(m)$

(P4) For all $m \in \mathbb{N}, S(m) = 0$ is false

(P5) Let $A \subset \mathbb{N}$. If $0 \in A$, and $m \in \mathbb{N} \to S(m) \in \mathbb{N}$, then $A = N$.

*Remark.* Note that the 5th axiom refers to a subset $A$ of $N$. The subset $A$ can be described as a property of $\mathbb{N}$, and elements of $A$ can be described as natural numbers that satisfy the property $A$. For example, if $A$ is the set of prime numbers, then if $n \in A$, then n is a prime number. The principle of natural induction says that if it can be shown that if a property is true for a base case $n = 0$, and we can show that if the properties holding for some $k \in \mathbb{N}$ implies that the property holds for $n = k + 1$, then the property holds $\forall n \in \mathbb{N}$.

We encounter the first difference here. In classical mathematics, the principle of Natural induction is assumed as an axiom. In Agda, we can construct the principle of Natural induction as a function.

## 3.1 Addition and Properties

Addition is defined recursively as follows:

1. $\forall m \in \mathbb{N},\ m + 0 = m$
2. $\forall m, n \in \mathbb{N},\ m + S(n) = S(m + n)$

With addition defined, well-known properties of addition of the Natural Numbers can be proved using the Peano Axioms. The following proofs are already implemented in the Type Topology library.

**Lemma 3.1.** $\forall m \in \mathbb{N},\ 0 + m = m$

*Proof.* By induction on $m$.

- Base Case : $0 + 0 = 0$ by definition of addition
- Inductive Hypothesis : Let $k \in \mathbb{N}$, and assume that $0 + k = k$
- Inductive Step :

$$\begin{aligned} 0 + S(k) &= S(0 + k) && \text{by definition of addition} \\ &= S(k) && \text{by the inductive hypothesis} \end{aligned}$$

□

**Lemma 3.2.** $\forall m, n \in \mathbb{N},\ S(m) + n = S(m + n)$

*Proof.* Omitted. □

**Theorem 3.3** (Commutativity of Addition of Natural Numbers)**.** *Let* $a, b \in \mathbb{N}$. *Then* $a + b = b + a$

*Proof.* By induction on $b$.

- Base Case :

$$\begin{aligned} a + 0 &= a && \text{by the definition of addition} \\ &= 0 + a && \text{Lemma 3.1} \end{aligned}$$

- Inductive Hypothesis : Let $k \in \mathbb{N}$, and $a + k = k + a$
- Inductive Step :

$$\begin{aligned} a + S(k) &= S(a + k) && \text{by the definition of addition} \\ &= S(k + a) && \text{by the inductive hypothesis} \\ &= S(k) + a && \text{by Lemma 3.2} \end{aligned}$$

□

I now list a number of results I have proved in Agda.

**Theorem 3.4** (Associativity of Addition of Natural Numbers)**.** *Let* $x, y, z \in \mathbb{N}$*. Then* $(x+y)+z = x+(y+z)$

**Corollary 3.4.1** (Addition and Successor)**.** *Let* $x, y \in \mathbb{N}$*. Then*

- $x + S(y) = S(x + y)$
- $S(x) + y = S(x + y)$

**Corollary 3.4.2** (Addition Cancellable)**.** *Let* $x, y, z \in \mathbb{N}$*, then*

- *If* $x + z = y + z$*, then* $x = y$
- *If* $z + x = z + y$*, then* $x = y$

**Corollary 3.4.3** (Sum to Zero)**.** *Let* $a, b \in \mathbb{N}$*. If* $a + b = 0$*, then* $b = 0$*.*

## 3.2 Order

Ordering on the natural numbers can be defined in two equivalent ways, with the first described in [8], and the second inductively.

1. Let $m, n \in \mathbb{N}$. Then define $m \leq n : \exists k \in \mathbb{N}$ such that $m + k = n$
2. - $0 \leq n$ : True
   - $S(m) \leq 0$ : False
   - $S(m) \leq S(n) : m \leq n$

   In either case, we have $m \geq n = n \leq m$

Strict Ordering can then be defined as an embedding of the above definition:

$$m < n : S(m) \leq n,\ m > n : m < n$$

I build on the work of Escardó in [5] on the ordering of Natural Numbers.
Following is a list of results I have proved on the ordering of natural numbers.

**Theorem 3.5.** *Let* $x, y, w, z \in \mathbb{N}$*. Then*

- $x < y$*, or* $x = y$*, or* $y < x$ *[Naturals Order Trichotomous]*
- $x \leq y$ *or* $y \leq x$ *[$\leq$-Dichotomy]*
- *if* $x = y$*, then* $x \leq y$
- *if* $x < y$*, then* $x \neq y$
- *if* $x < y$*, and* $w < z$*, then* $x + w < y + z$
- *if* $x < y$*, then* $x < y + z$
- $x \leq y$ *or* $y \leq x$
- *if* $x < y$*, then* $\exists k \in \mathbb{N}$ *such that* $S(k) + x = y$
- *Let* $S \subset \mathbb{N}$*, then if* $\exists x, y \in S$ *are both least elements of* $S$*, then* $x = y$

I needed to implement the trichotomy of natural numbers to prove the result that multiplication of natural numbers is cancellable in theorem 3.6. With my rudimentary experience of Agda at this point, this proved to be challenging. The first three cases are trivial (Agda's Auto-Solver will provide easy proofs). The problem then reduces to handling $(S(x) < S(y))$ or $(S(x) = S(y))$ or $(S(y) < S(x))$. With hindsight, the solution here is to recursively call the function and use trichotomy for $x$ and $y$. This is proving trichotomy inductively using Agda's built-in pattern matching. At this time I failed to see this correspondence, and wrote the function using the constructed principle of natural induction.

## 3.3 Multiplication

I now list an incomplete list properties of multiplication that I have proved in Agda (full list in html).

**Theorem 3.6.** *Let $x, y, z \in \mathbb{N}$, then*

| | |
|---|---|
| $x * 0 = 0$ | *[Zero Right Is Zero]* |
| $0 * x = 0$ | *[Zero Left Is Zero]* |
| $x * 1 = 0$ | *[Mult Right ID]* |
| $1 * x = 0$ | *[Mult Left ID]* |
| $x * (y + z) = x * y + x * z$ | *[Distributivity of Mult over Naturals Left]* |
| $(x + y) * z = x * z + y * z$ | *[Distributivity of Mult over Naturals Right]* |
| $x * y = y * x$ | *[Commutativity of Multiplication]* |
| $(x * y) * z = x * (y * z)$ | *[Associativity Of Multiplication]* |
| *If $z > 0$, and $x * z = y * z$, then $x = y$* | *[Multiplication Cancellable]* |

## 3.4 Division

Division is defined as follows:

**Definition 3.6.1** (Division of Natural Numbers)**.** Let $x, y \in \mathbb{N}$. Then define $x \mid y$ if $\exists k \in \mathbb{N}$ such that $x * k = y$.

The following proof is an adapted version of the division theorem from [6], page 12. I needed to adapt the proof in [6] as it defines division for integers, however at this stage I am still working with natural numbers, and hence need to be careful when working with subtraction which is not closed on the natural numbers.

**Theorem 3.7.** *Let $a, d \in \mathbb{N}$. Then there exists a unique $q, r \in \mathbb{N}$ such that $a = q * d + r$ and $r < d$.*

*Proof.* Let $q \in \mathbb{N}$ be the largest such that $q \leq \frac{a}{d}$. Let $r = a - q * d$. Then $a = q * d + r$, and $r \in \mathbb{N}$.

It must be shown than $r < d$. For a contradiction, suppose that $r \geq d$. Then $\frac{r}{d} \geq 1$, and $q + 1 \leq q + \frac{r}{d} = \frac{a}{d}$. But $q + 1 \geq q$, and q was chosen to be maximal such that $q \leq \frac{a}{d}$, so we have a contradiction and $r < d$.

To prove that $q$ and $r$ are unique, suppose that $q, q', r, r' \in \mathbb{N}$ such that

$$a = q * d + r = q' * d + r', r, r' < d$$

If $q = q'$, then we are done. Suppose that $q \neq q'$. Without loss of generality, suppose that $q < q'$. We now consider the cases $r = r'$ and $r \neq r'$. If $r = r'$ then we have

$$q * d + r = q' * d + r$$

, but we also have that

$$q * d + r < q' * d + r$$

, and we have a contradiction. Now suppose that $r \neq r'$. Then we have that either $r < r'$ or $r' < r$. If $r' < r$, then $\exists k \in \mathbb{N}$ such that $q * d + k = q' * d$, and since $q < q'$, $\exists m \in \mathbb{N}$ such that $k = m * d$. But $k < d$, and $k = m * d > d$, so we have a contradiction. A similar argument gives a contradiction for $r' < r$, and so by contradiction we have that $q = q'$, which gives $r = r'$ and we have proved uniqueness. □

The proof of the division theorem was my first major obstacle of the project. I had two major difficulties when attempting this proof.

Firstly, I needed a method to find constructively "the largest $q \in \mathbb{N}$ such that $q \leq \frac{a}{d}$". Note that these lecture notes do not provide any hint as to how one may find such a $q$, and do not even provide any evidence that such a $q$ actually exists. This is just one of many instances where authors make assumptions in proofs that may be intuitive, but do not hold trivially. There is a lack of rigour that makes it difficult to easily translate the proof into Agda, which demands every proof be fully rigorous. My project supervisor had provided a function for finding the minimum element satisfying a property (given an inhabitant of the property within a bound), by reduction to bounded minimisation. On his suggestion, I attempted to adapt his work into a function for bounded maximisation. Unfortunately, at this time my Agda knowledge was not at level that allowed me to complete this adaptation.

Secondly, I had to work with subtraction on the natural numbers in order to prove the that quotient and remainder posited by the theorem are unique. I had to realign my understanding of subtraction to match concept of subtraction in Agda. Instead of a function that "takes" one number away from another, it is necessary to think of subtraction in terms of addition. That is, informally, $a - b := \exists x \in \mathbb{N}$ such that $a + x = b$. The type theoretical implementation of subtraction is no different to how it works in classical mathematics, but reframing how subtraction works allows me to more easily use the information contained in subtraction, in terms of using subtraction in pen and paper proofs.

With my attempts to prove the division theorem using bounded maximisation failing at this time, I managed to implement a proof of division theorem using natural induction. In this case, I found the proof by experimenting with induction on both $a$ and $d$ and case splitting on the k value in the inductive step. Agda was a crucial guide; by setting up the induction in various ways, Agda's goals can provide insight into how to complete the proof. It should be noted that in other cases, using Agda in this way can have the opposite effect. By only considering the goals, it is easy to lose focus on the proof. There have been several occasions where I have tried to prove a false statement as a result of not constructing the proofs on pen and paper first.

After my work on the integers and rationals, I was able to return to naturals division with more experience. I was now able to find a maximal element satisfying a property, given a bound and proof of inhabitance of the property. I was also able to implement a proof of the division theorem that more closely follows the proof given in [6]. I was able to clean up my first proof of uniqueness of the division theorem, prove that my two proofs of the division algorithm agree, and further prove that my two proofs for uniqueness agree.

In this section, I also proved some properties of division, including one surprisingly difficult property.

**Theorem 3.8.** *Let $x, y \in \mathbb{N}$. Suppose $x \mid y$ and $y \mid x$. Then $x = y$.*

Informally, the proof on pen and paper is shown as follows:

*Proof.* Since $x \mid y$ and $y \mid x$, $\exists a, b \in \mathbb{N}$ such that $a * x = y$ and $b * y = x$, and we have that

$$a * b * y = y$$

Since $a, b$ are natural numbers, the only possibility is that $a * b = 1$, and so $a = 1$ and $b = 1$, and therefore $x = y$.

□

This kind of reasoning is widely used in mathematics, where intuitive results are taken as granted, without proof, or reference to a proof. The last line of the proof is true, but it is not *trivially* true, and certainly does not translate directly into an Agda proof. In Agda, my proof that division is anti-symmetric requires that multiplication is cancellable. My proof that multiplication is cancellable requires a proof that multiplication is cancellable with respect to order, as well as the proof that natural numbers are a trichotomy with respect to order.

It is often surprising the foundations required to complete "trivial proofs". In classical mathematics, many details are usually omitted or left to the reader. It makes sense to omit such details, it isn't viable for a mathematician to verify that every aspect of the foundation of mathematics when they write high level proofs. With omissions, we lose rigour, and it would still be nice to know for sure that *every* assumption made is *actually* true, regardless of it's triviality. Using a proof checker such as Agda without postulating assumptions, we can be sure that a proof is fully rigorous. We can also see the truth of any sub-proofs used in a proof.

## 3.5 Highest Common Factor

We define the highest common factor (HCF) adapted from [6].

**Definition 3.8.1.** Let $a, b \in \mathbb{N}$. Then the highest common factor $h$ of $a$ and $b$ is the largest natural number $h$ that is a common factor, denoted $h = hcf(a, b)$.

Informally, the notion of "largest natural number" in the case of the HCF is equivalent to saying that the HCF can be divided by any given common divisor of two natural numbers, which is how we translate this concept into Agda. The HCF was a challenging part of this project.

Recall that a full proof of a proposition consists of an algorithm which produces an inhabitant of the proposition, and a proof that the proposition is a subsingleton. In this case, the proposition is the existence of a HCF, and the algorithm is one that produces the HCF. The standard technique to find the HCF is Euclid's Algorithm. My challenge, therefore, was to translate Euclid's algorithm into Agda, and prove that it outputs the HCF, which took a substantial amount of time.

We now state an adapted lemma from [6] without proof.

**Lemma 3.9.** *Let $a, b, q, r \in \mathbb{N}$. Suppose that $a = q * b + r$. Then $hcf(a, b) = hcf(b, r)$.*

**Theorem 3.10.** *Euclid's Algorithm, adapted from [6]*

*Let $a, b \in \mathbb{N}$, with $a \geq b$. Let $q, r$ be the quotient and remainder obtained by applying theorem 3.7 to $a$ and $b$. Then $a = q * b + r$*

*Let $h$ be the output of the following recursive function:*

$$hcf(a,\ b) = \begin{cases} hcf(b, r) & r \geq 0 \\ b & r = 0 \end{cases}$$

*Then $h = hcf(a, b)$.*

*Proof.* The algorithm terminates at any step where $r_m = 0$, for some $m$th step of the algorithm. If this is case, then we have that $a_{m-1} = a_m * q_m$, so $a_{m-1} \mid a_m$ by the definition of division, and therefore $hcf(a_m, a_{m-1}) = a_{m-1} = h$. Consider step $k$, where $1 \leq k < m$. We have that $a_{k-1} = a_k * q_k + a_{k+1}$. In the initial case, we have that $a_1 = a_2 * q_1 + r_1$, where $a_1 = a, a_2 = b$. We now have a chain of equalities, and

$$h = a_m = hcf(a_{m-1}, a_m) = \cdots = hcf(a_{k-1}, a_k) = \cdots = hcf(a_1, a_2) = hcf(a, b).$$

Note that $a_1 > a_2 > \cdots$, so we must eventually reach some step where $a_{k+1} = 0$.

□

The first challenge is the algorithm, in that Agda requires any function called recursively to have an argument strictly smaller than the original argument (for two or more arguments this may happen lexicographically, see [12]). If we construct a function identical to the one above in Agda, it will compute our candidate for the HCF (regardless of the correctness of the function), but Agda's termination checker will flag this function. Agda cannot be sure that this function terminates, it can not verify without help that this algorithm reduces at each step, and I could not translate the above proof into Agda using pattern matching or the principle of induction without failing the termination check.

Using suggestions from my supervisor and the Agda club, I knew I needed to use Strong induction along with the second condition of theorem 3.7 that produces a remainder strictly below the divisor. Fortunately, in the TypeTopology, strong induction is already implemented and named course of values induction. A lecture from Cornell University [9], based on [10], provides a proof of Euclid's algorithm using Strong induction. I was able to use this as a framework to implement the HCF into Agda, which also required a lemma concerning a property of division.

**Lemma 3.11.** *Let $a, b, c, d \in \mathbb{N}$. Suppose $a * b = a * c + d$. Then $a \mid d$.*

*Proof.* Omitted. □

This lemma serves as an another example of a proof (HCF) requiring intermediate results that are not trivial. I completed this proof with just a pen and paper draft, but whilst the proof is not difficult it is certainly time-consuming and requires intermediate results itself (from theorem 3.5). It is generally difficult to predict that a result such as lemma 3.11 will be necessary. In this project, I use a top down approach to proofs, in that I begin by writing in the type of the proof I am working on (sometimes with the same proof in classical mathematics at hand). Generally, the proof will need intermediary results, and many of these are common and already implemented (for example commutativity, or division anti-symmetric). I can complete the proof by writing in the type of the intermediate proof, and then subsequently prove the intermediate result. This approach is necessary for a project like this, as one could spend an inordinate amount of time proving properties of, say, multiplication or division that are ultimately not necessary for the final goal.

This is similar to the problem that a classical mathematician faces identified earlier in this report. How much mathematics should one assume to serve as foundation to work in? Using this approach in Agda, the answer is that nothing is assumed. Ultimately, a mathematician would be able to use these formalised proofs and know they are working with a concrete foundation, and be able to easily obtain the proof of any intermediate results they use in their work. Of course, there are some minor drawbacks that will become apparent in due course when proving properties of more complex structures.

The second challenge proving that the HCF is a proposition. More precisely, we want to prove that the statement "Given $a, b \in \mathbb{N}$, there exists a $d \in \mathbb{N}$ such that $d$ is the HCF of $a$

and $b$" is a proposition. This is an dependent pair. To prove equality of a dependent pair, then given two instances of the pair such that the first projections are equal, we need to know that the second projections (the type family, or dependent function) are also equal (see [13]). In the context of this example, this means that given $a, b \in \mathbb{N}$, we need to show that any two HCF's $h$ and $h'$ of $a$ and $b$ are equal, and also that the statement "$h$ is a common factor of $a, b$" is a proposition. This statement itself consists of an independent pair of statements, "$h$ is a common factor of $a, b$", and "given any common factor $c$ of $a, b$, then "$c \mid h$". The second statement here requires the use of Function Extensionality.

The motivation of Function Extensionality (FE) is that we want to be able to decide when two functions are equal. Note that there is a distinction between intensionally equal functions, and extensionally equal functions. We say that two functions are extensionally equal if they are pointwise equal, or informally that for $f, g : A \to B$, then $f(x) = g(x)$ for $x : A$. The distinction is necessary because two functions may be pointwise equal, but not intensionally equal (indeed one can easily construct two intensionally different functions which have the same outputs at each point). It has been shown (as stated in [4]) that it is not provable or disprovable within the framework of MLTT that pointwise-equal functions are equal. As a result, FE is an assumption we must make in order to proceed with the proof that HCF is a proposition. There are three points I would like to make on FE:

- We do not lose anything by assuming FE. Rather, we assume FE as necessary for proofs, which agrees with the notion above that for any proof, we assume only as much as we need to complete that particular proof. Later, we may choose to assume FE (and other versions of extensionality) for whole modules, antithetically and mainly for bookkeeping reasons, when the majority of functions in a module require this assumption.

- FE can be obtained as a result of either Univalence or from an interval type (see [13]). The Univalence Axiom, very informally, posits that equivalence is equivalent to identity. A detailed explanation of the connections between Constructive Mathematics, Homotopy Theory, Topology, Type Theory and Category Theory is beyond the scope of this report. This project is about learning how to implement mathematical proofs in Agda. The Cubical Agda library obtains FE as a result of the interval type.

- I use the implementation of FE already developed in [5] in this project, which obtains FE as a result of the Univalence axiom. Luckily, I am able to use this as a framework within which to implement mathematical proofs. Without such a framework, much have this project would have been impossible in the same time period. It is much easier to use the tools provided than it is to re-invent the wheel, and I certainly wouldn't have been able to implement the required Univalent Foundations necessary for the HCF alone, let alone what is required for the rationals. For the purposes of learning how to implement mathematical proofs as in Agda, we can treat Function Extensionality as a black box axiom that allows us to prove that pointwise-equal functions are equal, respecting that this black box is rooted in the Univalent Foundations of Mathematics.

Assuming FE, the HCF problem reduces to showing that division is a proposition. One can see from the definition that given $a, b$, division is a dependent pair of a Natural Number $k$, and a proof that $a * k = b$. In this case, we need to know that equality of natural numbers is a proposition, i.e that given $x, y \in \mathbb{N}$, the statement "$x = y$" is a proposition. If we generalize this idea to some arbitrary type $A$ instead of $\mathbb{N}$, what we have is the idea of types as sets. As stated in [4], "a type is defined to be a set if there is at most one way for any two of its elements to be equal". In this case, in Type Topology it has already been proven that $\mathbb{N}$ is a set, and therefore it follows that equality of the naturals is a proposition. In [13], it's noted that this

notion of the set is not the same as the sets used in classical mathematics, it's more similar to the sets used in structural mathematics and category theory.

Note that we can only prove that HCF is a proposition when we restrict the potential HCF to be greater than zero. As per the definition of the highest common factor, $h$ is required to be a common factor of $a$ and $b$, which requires that $h$ is a factor of $a$. "$0 \mid a$" is not a proposition in type theory. There are infinitely many $x \in \mathbb{N}$ such that $0 * x = b$, and therefore we cannot guarantee uniqueness of x for $0 \mid a$. This is not a problem, as we generally do not divide by 0.

# 4 Integers

There are different ways to implement the Integers. Naively, one could use the Natural Numbers with a positive or negative sign, but this implementation introduces the ambiguity of positive and negative zero. To handle the ambiguity, an easy solution is to use the constructors pos and negsucc, where for $x \in \mathbb{N}$, we think of negsucc $x$ as - S(x). This approach is the one used by the Cubical Agda library, and we adopt the same approach in this project.

Alternatively, one could define the integers as a Set Quotient as in [13], as a pair of Natural Numbers and an equivalence relation. There are disadvantages and advantages to each approach, some proofs may be easier to implement depending on the chosen approach. I chose the Inductive approach as it was more intuitive to me at the time, however with hindsight it might have been a better choice to use the Set Quotient implementation. Since my definition of the Rationals are as a Set Quotient, working on the Integers as a Set Quotient might have provided a better foundation for me to work on the Rationals as a Set Quotient, and as an extension of the project it might be beneficial to re-implement the integers to explore if the Set Quotient integers result in a cleaner implementation of the Rationals.

At this point in the project, with Natural Numbers, HCF, being Coprime and a basic definition of the Integers ready to go, I naively wanted to work on the Rationals numbers directly. It quickly became apparent that this was not feasible. The rationals proved to be a major challenge, to be discussed later, but the first necessary work involved reproducing many of the proofs I completed for the Naturals, now for the Integers.

Proving properties of the integers is not much more difficult than proving properties of the naturals numbers, with the key difference being the extra negative case to consider. Note that functions with an integer domain usually have to pattern match on at least two cases (positive and negative), with extra cases if considering zero is necessary. A particularly egregious example is the proof that the integers are trichotomous with respect to order, which considers 16 cases. Another distinction of proofs on the integers is the use of the induction principle. Proofs that require induction now require an extra step function to handle the integers less than 0. I found two different ways to handle induction for the Integers, with an example given here.

- Define a principle of Induction for the Integers with two step functions. The first step function is equivalent to the step function of natural induction, while the second function informally has the form: If $k \in \mathbb{Z}$, and $A\ k$ holds for some property $A$, then $A\ p(k)$ holds, where p(k) denoted the predecessor of k.

- The second approach is to use the Principle of Natural Induction twice. Create two auxiliary functions, where the target for induction is replaced with only positive integers in the first auxiliary function, and by only negative integers in the second auxiliary function. Prove the property for the two auxiliary functions, and the final proof then follows trivially.

The first way is the intuitive extension of the inductive principle for natural numbers and subjectively may be more aesthetic than the second way, but the second way seems to result

in more concise proof overall, at the expense of having to write two extra auxiliary functions for each induction. The Cubical Agda library contains some properties of addition, but has no implementation of multiplication and it's properties. To prove properties of addition, the Cubical Agda library utilises an abtracted version of induction for particular properties, as seen in [14]. As an extension of this project to simplify the code, I could rewrite many of my proofs by abstracting proofs of some properties and trying to reduce similar code, I don't think is necessary. I don't foresee too many occasions to use the abstracted code, and subjectively it feels less readable.

I have implemented following definitions and theorems related to the integers in Agda.

**Definition 4.0.1.**

- Addition - $+$
- Multiplication - $*$
- Negation - $(-)$
- Absolute Value - $||$
- Successor and Predecessor - $succ\mathbb{Z}$ and $pred\mathbb{Z}$
- Positivity Property
- Negativity Property
- "Is zero" property
- "Greater than zero"property
- Order - $<, \leq$

The following is not a complete list for brevity.

**Theorem 4.1.** *Let $x, y, w, z \in \mathbb{Z}$. Then*

- $x + (pos\ 0) = x = (pos\ 0) + x$
- $succ\mathbb{Z}(x) + y = succ\mathbb{Z}(x + y) = x + succ\mathbb{Z}(y)$
- $pred\mathbb{Z}(x) + y = pred\mathbb{Z}(x + y) = x + pred\mathbb{Z}(y)$
- *If* $succ\mathbb{Z}(x) = succ\mathbb{Z}(y)$, *then* $x = y$
- *If* $pred\mathbb{Z}(x) = pred\mathbb{Z}(y)$, *then* $x = y$
- $x + y = y + x$
- $(x + y) + z = x + (y + z)$
- *If* $z + x = z + y$, *then* $x = y$
- $x * (pos\ 0) = pos\ 0 = (pos\ 0) * x$
- $x * (pos\ 1) = x = (pos\ 1) * x$
- $-(x + y) = (-x) + (-y)$
- $(x + y) * z = (x * z) + (y * z)$

- $x * y = y * x$
- $-(-x) = x$
- $(-x) * y = -(x * y) = x * (-y)$
- $(x * y) * z = x * (y * z)$
- *Addition of positive integers is equivalent to Addition of Natural Numbers*
- *Multiplication of positive integers is equivalent to Addition of Natural Numbers*
- $(x + (-x)) = (pos\ 0)$
- *If* $x < y$ *and* $y < z$*, then* $x < z$
- $x < y$ *or* $x = y$ *or* $y < x$
- *If* $(pos\ 0) < z$*, and* $x < y$*, then* $x * z < y * z$
- *If* $(pos\ 0) < z$*, and* $x * z < y * z$*, then* $x < y$
- *If* $z < (pos\ 0)$*, and* $x < y$*, then* $y * z < x * z$
- *If* $z < (pos\ 0)$*, and* $x * z < y * z$*, then* $y < x$
- *If* $z \neq (pos\ 0)$*, and* $x * z = y * z$*, then* $x = y$

It was already proven in TypeTopology that $\mathbb{N}$ is a set, now we prove that $\mathbb{Z}$ is a set. We adopt the same strategy that Escardó used in [5]: Firstly, prove that $\mathbb{Z}$ is discrete, and then use the proof that discrete types are sets.

# 5 Rationals

The rational numbers were a much more difficult implementation compared to the integers and natural numbers. Similarly to the integers, we can define the rational numbers in multiple different ways.

**Definition 5.0.1** (Rational Numbers)**.** The Rationals, denoted $\mathbb{Q}$ are defined as a set quotient, an independent integer and natural number pair $x \in \mathbb{Z}, y \in \mathbb{N}$, with the relation that $|x|$ is coprime to $y$.

The rational numbers pose a number of challenge not present in the previous sections.

- Note that the natural number represents the number above it, to avoid the division by 0. This incrementation by one affects complicates definitions and proofs of the rationals, for example addition of two rationals must first correct the denominators, apply the addition and then decrement the resultant denominator.
- With both the Natural Numbers and Integers in play, we have to be fully rigorous with the operators. Addition on the Naturals and Addition on the integers are distinct operators, we cannot overload the operator as a classical mathematician might want. This usually means converting between Naturals and Integers as necessary within proofs. With addition and multiplication of rationals being defined in terms of addition and multiplication of integers and natural numbers, is it common to deal with expressions with 4 terms or more. When using equational reasoning, this usually results in required manipulation to arrange the terms in the correct order, which bloats many proofs with trivial re-arrangement of terms.

- Equality of terms of the rationals is more difficult than equality of Naturals or Integers. Proving properties of the rationals sometimes requires proving equality of a dependent pair. Usually, this equality must be proved after an operator has been applied to two rationals, and the result has been normalised by a "$to\mathbb{Q}$" operation, which brings a (potentially non-canonical) rational number to it's lowest terms. A large proportion of the remainder of the project was spent tackling this problem.
- Most proofs require a lot of information to be carried, specifically keeping track of the normalisation factor when reducing a rational to it's lowest terms. The types of terms withnin a proof become more complicated, and in general proofs of properties of the rationals are less readable than proofs on the naturals and integers. It is very necessary to have a clear idea of the structure of a proof (pen and paper drafts are very useful).

To compartmentalise the challenge, I defined a version of the rationals without the condition that the integer and natural number pair are in their lowest terms. The idea behind this is that I could implement proofs of the (potentially non-canonical) rationals, with the hope that it is then easier to extend these proofs to the canonical rationals.

**Definition 5.0.2.** The (potentially) non-canonical rationals are defined as an independent pair $\mathbb{Q}_n = \{(x, y) \colon x \in \mathbb{Z}, y \in \mathbb{N}\}$

The following definitions and results were completed in a top down process, where I completed these functions as they were necessary for the rationals, however they are listed here for brevity.

**Definition 5.0.3** (Addition of free rationals)**.** Let $(x, a), (y, b) \in \mathbb{Q}_n$, where $x, y \in \mathbb{Z}, a, b \in \mathbb{N}$. Then define

$$(x, a) + (y, b) = ((x\ \mathbb{Z}^*\ (pos\ S(b)))\ \mathbb{Z}^+\ (y\ \mathbb{Z}^*\ (pos\ S(a))), pred(S(a)\ \mathbb{N}^*\ S(b))$$

**Definition 5.0.4** (Multiplication of free rationals)**.** Let $(x, a), (y, b) \in \mathbb{Q}_n$, where $x, y \in \mathbb{Z}, a, b \in \mathbb{N}$. Then define

$$(x, a) * (y, b) = (x\ \mathbb{Z}^*\ y, pred(S(a)\ \mathbb{N}^*\ S(b))$$

**Definition 5.0.5** (Order of free rationals)**.** Let $(x, a), (y, b) \in \mathbb{Q}_n$, where $x, y \in \mathbb{Z}, a, b \in \mathbb{N}$. Then define

$$(x, a) < (y, b) = (x\ \mathbb{Z}^*\ (pos\ S(b))\ \mathbb{Z}^<\ (y\ \mathbb{Z}^*\ (pos\ S(a))$$

**Theorem 5.1.** *Let $p, q, r \in \mathbb{Q}_n$. Then,*

| | |
|---|---|
| $p + q = q + p$ | *[Addition Commutative on free rationals]* |
| $(p + q) + r = p + (q + r)$ | *[Addition Associative on free rationals]* |
| $p * q = q * p$ | *[Multiplication Commutative on free rationals]* |
| $(p * q) * r = p * (q * r)$ | *[Multiplication Associative on free rationals]* |

With a goal of implementing addition and multiplication on the canonical rationals, we need a function that reduces a rational number to it's lowest terms.

**Definition 5.1.1.** Let $q \in \mathbb{Q}_n$. Then define $to\mathbb{Q}(q)$ to be the function from $\mathbb{Q}_n \to \mathbb{Q}$ that sends a free rational number to it's canonical representation in its equivalence class.

This definition is implemented using a function that divides two numbers by their HCF, with the proof that this results in a pair of coprime natural numbers.

**Lemma 5.2.** *Let $x, y \in \mathbb{N}$. Then there exists $h, x, y \in \mathbb{N}$ such that $h * a = x$, $h * b = y$ and $x$ is coprime to $y$.*

We can now define addition, multiplication and order of rational numbers.

**Definition 5.2.1.** Let $((x, a), p), ((y, b), q) \in \mathbb{Q}$, where $x, y \in \mathbb{Z}$, $a, b \in \mathbb{N}$, and $p, q$ are proofs that the respective rational number is in it's lowest terms. Then,

| | |
|---|---|
| Addition of rationals | $((x, a), p) + ((y, b), q)) = to\mathbb{Q}\ ((x, a)\ \mathbb{Q}_n^+\ (y, b))$ |
| Multiplication of rationals | $((x, a), p) * ((y, b), q)) = to\mathbb{Q}\ ((x, a)\ \mathbb{Q}_n^*\ (y, b))$ |
| Order of rationals | $((x, a), p) < ((y, b), q)) = (x, a)\mathbb{Q}_n^<(y, b)$ |

Recall that in our representation of the Rationals, it must contain a proof that the rational is in it's lowest terms, and addition of rational numbers does not necessarily preserve this property. For addition and multiplication, we must therefore normalise the resultant rational number. Notice that to define order, we can simply ignore the proof that the rationals are in the lowest terms, as being in lowest terms has no impact on the evaluation of the order of two rational numbers.

With these functions on the free rationals, there still remains the challenge of handling equality of the canonical rationals, $\mathbb{Q}$. As a specific example, consider associativity of addition on $\mathbb{Q}$. On the left hand side, we add p and q and normalise the result, and then add r to this and normalise the subsequent result. On the right hand side, the normalisation of terms happen from right to left. The challenge, therefore, is to prove that the result of addition is not affected by when the normalisation of the rational occurs. This was a particularly difficult challenge for me, and required heavy input of both my supervisor and co-supervisor. Furthermore, it resulted it many of the functions defined previously. The following describes the setup required to be able to complete a proof of associativity of the rationals.

I needed to introduce an equivalence relation of the rationals. That is, the notion of two free rational numbers $(x, a), (y, b)$ being equivalent.

**Definition 5.2.2.** Let $(x, a), (y, b) \in \mathbb{Q}_n$. Then define $(x, a) \approx (y, b)$ as $x\ \mathbb{Z}^*\ (pos\ S(b)) = y\ \mathbb{Z}^*\ (pos\ S(a))$.

I also needed to prove the following lemma:

**Lemma 5.3.** *Let $(x, a), (y, b) \in \mathbb{Q}_n$, then $(x, a) \approx (y, b)$ if and only if $to\mathbb{Q}\ (x, a) = to\mathbb{Q}\ (y, b)$.*

The right to left implication can be shown without too much difficulty. The other direction required more work. Firstly, it required an extension of the function that normalises a rational number ($to\mathbb{Q}$), and secondly, it required an auxilliary proof:

**Lemma 5.4.** *Let $p, q \in \mathbb{Q}_n$. Suppose $p \approx q$, and both $p$ and $q$ are in their lowest terms. Then $p = q$.*

It required an extension of $to\mathbb{Q}$ in order to keep track of the data that was previously being lost to normalisation. When dividing by the highest common factor, it is necessary to keep track of this highest common factor. This was necessary because the assumed equivalence relation applies to free rational before normalisation, and without the data relating the free rational to it's canonical form, the equivalence relation becomes useless. Extending proofs doesn't pose a huge difficulty, and was required several times in this project, with a second occurrence discussed shortly.

The proof of the lemma was more difficult than the extension. In order to prove that $p = q$, I needed a further auxilliary proof.

**Lemma 5.5.** *Let $a, b, c \in \mathbb{N}$. Suppose $a$ is coprime to $b$, and suppose that $a \mid (b\ \mathbb{N}^*\ c)$, then $a \mid c$.*

Discussion with Alex Rice in the Agda Club lead to a pen and paper proof of this lemma, but it required Bezout's Lemma. At this stage, I had only implemented the HCF on the natural numbers, and not the integers. Clearly, Bezout's lemma requires the use of integers and the aforementioned extension became necessary at this point, however there was some time lost due to my approach at this stage of the project. Without fully considering that Bezout's Lemma required only the Extended Euclidean Algorithm, I implemented Division and HCF for the Integers. This was time consuming, and ultimately didn't provide the information I needed. It was a similar situation to the $to\mathbb{Q}$ loss of data, I would not be able to implement Bezout's Lemma without access to the data from the proof of HCF on natural numbers. I then re-implemented the HCF on the naturals, by duplicating the code from my previous proof of the Natural Numbers, and extending it to implement Euclid's Extended Algorithm in order to implement Bezout's Lemma. An extension of the project could tidy up duplicate code related to the HCF. The extra code for division and HCF of integers is correct and type-checks, but is not relevant to the remainder of this project. With Bezout's Lemma implemented, I was able to work my way back up the chain and prove the left to right implication is true.

Even with this equivalence relation in place, I was still unable to directly prove associativity of addition of canonical rational numbers. I needed to implement some more proofs related to the equivalence relation. In the case of the associativity, we want to prove that the left hand side is equal to the right hand side, where each side is a rational number. In the case of natural numbers, we could use equational reasoning to directly prove the equality of each side. We cannot directly prove the equality on the rationals using equational reasoning. Recall that we defined the rationals as a set quotient, which is implemented as a dependent pair. The idea is that we can use a version of equational reasoning, but instead using the equivalence relation, and then elevate equivalence to equality using lemma 5.3.

**Theorem 5.6.** *Let $p, q, r \in \mathbb{Q}_n$. Then*

| | |
|---|---|
| $p \approx p$ | *[$\approx$ Reflexive]* |
| *If* $p \approx q$, *then* $q \approx p$ | *[$\approx$ Symmetric]* |
| *If* $p \approx q$, *and* $q \approx r$, *then* $p \approx r$ | *[$\approx$ Transitive]* |
| $p \approx to\mathbb{Q}_n\ (to\mathbb{Q}\ p)$ | *[Free rational $\approx$ Canonical rational]* |
| *If* $p \approx q$, *then* $(p\ \mathbb{Q}_n^+\ r) \approx (q\ \mathbb{Q}_n^+\ r)$ | *$\approx$ respects addition* |
| *If* $p \approx q$, *then* $(p\ \mathbb{Q}_n^*\ r) \approx (q\ \mathbb{Q}_n^*\ r)$ | *$\approx$ respects multiplication* |

With all of this machinery in place (which took several weeks to implement), I could finally implement a proof of the following theorem, which requires lemma 5.6, lemma 5.4 and lemma 5.3.

**Theorem 5.7.** *Let $p, q \in \mathbb{Q}_n$. Then*

$$to\mathbb{Q}\ (p\ \mathbb{Q}_n^+\ q) = (to\mathbb{Q}\ p) + (to\mathbb{Q}\ q)$$

The final piece of the puzzle for associativity of addition of canonical rationals was a function that extracts a free rational from a canonical rational (intuitively we just take the canonical rational itself), to be able to apply the above theorem to intermediary rational numbers in a proof.

**Lemma 5.8.** *Let $q \in \mathbb{Q}$. Then there exists $q' \in \mathbb{Q}_n$ such that $q = to\mathbb{Q}_n\ q'$*

This was a surprising amount of work required to be able to prove a property as simple as associativity of the canonical rational numbers. To recap, it required the extension of HCF in order to obtain Bezout's lemma, the proof that addition on the free rationals is associative, a new equivalence relation on the rationals and properties of this equivalence relation, and a proof that the $to\mathbb{Q}$ function can be applied before or after addition of free rationals for the same result. Fortunately, these foundations resulted in the process of proving the other basic properties being more streamlined.

**Theorem 5.9.** *Let $p, q, r \in \mathbb{Q}$. Then*

- $p + q = q + r$
- $(p + q) + r = p + (q + r)$
- $p * q = q * p$
- $(p * q) * r = p * (q * r)$

Whilst lengthy, it is trivial to implement transitivity of order of rational numbers, and trichotomy of rational numbers, in comparison to the above properties.

**Theorem 5.10.** *Let $p, q, r \in \mathbb{Q}$.*

- *Suppose that $p < q$, and $q < r$. Then $p < r$*
- *$p < r$, or $p = q$, or $q < p$*

There are more properties of the rationals defined in the section below, however there is scope for more properties of the rationals to be implemented. At the very least, most of the properties from theorem 4.1 have an analogous proof for the rationals. I have not implemented all of these functions in this project. As mentioned previously, I have adopted a top-down approach for this report, which means that I have implemented the proofs strictly necessary for the goal of implenting the Dedekind Reals in Agda. Whilst the framework is in place to complete these functions, it will be a lengthy process, and some of the functions may not be necessary to the goal. Furthermore, there may be another strategy for implementing these functions that avoids some of the difficulty of working with the $to\mathbb{Q}$ function, and the equivalence $\approx$, which will be discussed in a later section. With the rationals implemented with some basic properties, I was ready to consider the Dedekind reals in Agda.

# 6 Reals

The Reals numbers are the biggest surprise to me, in terms of formal mathematical representation. The rational numbers are intuitively formalised, whereas it is not easy to intuitively describe the formal description of real numbers. This is surprising because the real numbers are the most familiar field to a mathematician, and many of the courses in an undergraduate degree commonly use the real numbers as the basis of the course, but the construction of the reals is usually unmentioned, most likely due to this lack of intuitiveness.

In mathematics research, usually the real numbers are assumed and hence, also assumed is the foundational mathematics required to construct the reals, shown in this project. There is nothing inherently wrong with assuming these foundations, but is an interesting note that many mathematicians might have never seen, or thought about the construction of the reals, especially within applied mathematical fields such as physics.

Formally, the real numbers are a complete archimedean ordered field. There are two common constructions, as described at the start of chapter 11 in [13], namely the Cauchy reals and

the Dedekind reals. Within classical mathematics, we could view the two constructions as isomorphic, but without the law of excluded middle or countable choice it is not known whether the constructions coincide. We choose to construct the Dedekind Reals, which is not finished at the time of writing, however in the long term there is the possible extension of constructing the Cauchy Reals in Agda.

Of course, if we want to show that the Dedekind reals form an complete archimedean ordered field, we need to formalise these concepts too. The field is a well known concept in Mathematics (for a definition see [2]), and is not difficult to translate into Agda. A field can be thought of as an abstraction of mathematical structures which have *nice* properties. With a description of the structure of a field, and the properties it has, we can prove properties that a field has in general, and any mathematical structure which satisfies the field axioms can inherently possess properties we prove for fields. A field is known as an ordered field if it is equipped with a relation operation, and satisfies a further two properties than a field requires. The difficulty lies in proving that a mathematical construction is indeed a field. I have partially shown the the rational numbers are an ordered field, there are still holes to be filled.

An ordered field is archimedean if it satisfies the Archimedean principle. For the reals, this can have the following form, as stated in [13].

**Theorem 6.1** (Archimedean Principle of R)**.** *Let $x, y \in \mathbb{R}$. Suppose $x < y$. Then there exists $q \in \mathbb{Q}$ such that $x < q < y$.*

The notion of completeness of an ordered field depends on the ordered field itself, but if the field satisfies one version of the completeness axiom, then it satisfies all of them. The notion of completeness of a Dedekind field is stated as in [13], corollary 11.2.16. In the general sense, we are saying that any non-empty subset of $\mathbb{R}$ with an upper bound has a *least* upper bound.

**Theorem 6.2.** *The Dedekind reals are Dedekind complete: for every real-valued Dedekind cut $(L, U)$ there exists a unique $x \in \mathbb{R}$ such that $L(y) = (y < x)$ and $R(y) = (x < y)$.*

Note that this notion of completeness is different to the notion of Cauchy completeness. Cauchy completeness can be generalised to completeness of a metric space, which is the notion of a metric space without any holes, or more formally a metric space in which every Cauchy approximation has a limit, ([13], Definition 11.5.2).

These distinctions are beyond the scope of this project. As discussed shortly, the Dedekind Reals have not yet being fully implemented as a result of this project, however implemented the Cauchy reals within Agda and considering these distinctions could be included as an extension of this project.

We now introduce the Dedekind reals, with one slight variation to the [13] definition of a Dedekind Cut, and notational differences. Here, we can understand the notation $L(p)$ to mean that the rational number $p$ satisfies some property $L$.

**Definition 6.2.1.** A Dedekind Cut is a pair $(L, R)$ of properties of rational numbers such that the following properties hold:

- Inhabited: $\exists\ p, q \in \mathbb{Q}$ such that $L(p) and R(q)$.
- Rounded: Let $p, q \in \mathbb{Q}$. Then

$$L(p) \iff \exists\ p' \in \mathbb{Q} \text{ such that } (p < p') \wedge L(p')$$

$$R(q) \iff \exists\ q' \in \mathbb{Q} \text{ such that } (q' < q) \wedge R(q')$$

- Disjoint: Let $p, q \in \mathbb{Q}$. Suppose that $L(p)$ and $R(q)$. Then $p < q$

- Located: Let $p, q \in \mathbb{Q}$. Suppose that $p < q$. Then $L(p)R(q)$

I briefly discuss the intuition of each property, which I used to help guide the implementation of these properties in Agda. We can picture a cut as a point on the real line, where everything below the point is contained in $L$, and everything above in $R$, where the point is not contained in either $L$ or $R$. Inhabitedness says that for any cut we choose, there exists a rational on either side of the cut. Roundedness says that if we choose a rational number in either side of the cut, we can find a rational that is closer to the cut on that same side. Disjointness says that any rational in the left cut is strictly less than any rational in the right cut. Locatedness says that we can place at least one side of any interval of rational numbers in the left or right cut. The HoTT Book [13] has a slight variation of the disjointness property that says that a rational number is never located in both sides of a cut. As part of an extension of this project, it could be shown that in the presence of the other conditions these two variations of disjointness are equivalent.

The Dedekind Reals are the set of all Dedekind Cuts, that is the set of predicate pairs that that satisfy the properties of a Dedekind Cut. The HoTT Book [13] goes on to further state, without proof, that there is an embedding $\mathbb{Q} \to \mathbb{R}$. For $x \in \mathbb{Q}$, we choose the two properties for the cut as

$$L_x(p) = (p < r) \text{ and } R_x(q) = (x < q)$$

I was able to define the Dedekind Reals and implement this embedding from $\mathbb{Q} \to R$, with some difficulty. Disjointedness and locatedness follows quickly from transitivity of order of rationals, and trichotomy of rationals respectively. Inhabitedness required two extra proofs of the rationals.

*Remark.* Note that in the definition of the Dedekind Reals, instead of using a Sigma type, we use the truncated Sigma type. Recall that we work constructively, so when we use a Sigma type we must give a witness that satisfies the dependent function. The truncated Sigma "forgets" this witness. The only information a truncated Sigma type contains is the inhabitedness of a type, and logically we can view it as existence. Full explanations can be seen in [13] and [4]. Proposition Truncation must be assumed in the same manner as Function Extensionality, and the discussion of FE applies to Propositional Truncation.

**Theorem 6.3.** *Let $x \in \mathbb{Q}$. Then*

| | |
|---|---|
| *There exists $p \in \mathbb{Q}$ such that $p < x$* | *[$\mathbb{Q}$ has no least element]* |
| *There exists $q \in \mathbb{Q}$ such that $x < q$* | *[$\mathbb{Q}$ has no maximum element]* |

The proofs of these required a result from theorem 5.6, that a free rational is equivalent to it's canonical representation. There is general difficulty of proving properties of the rational numbers, in terms of using both natural numbers, integers and proofs of their respective properties all at the same time. This is more notationally intense than one might see in usual proofs in classical mathematics, but Agda requires completeness, every step must be shown.

Roundedness demonstrates this general difficulty. If we are given $p, q \in \mathbb{Q}$ and we want to find $x \in \mathbb{Q}$ such that $p < x < q$, intuitively we take $x$ as the midpoint of $p$ and $q$. Using this intuition, I implemented the following lemma.

**Lemma 6.4.** *Let $p, q \in \mathbb{Q}$, and suppose that $p < q$. Then there exists $x \in \mathbb{Q}$ such that $p < x < q$.*

This was an extremely lengthy proof. There is no technique used in this proof that is any more difficult than anything previously used in this project, but it exemplifies the tedium that can arise when formally proving mathematical proofs within the setting of constructive type-theory. This embedding $\mathbb{Q} \to \mathbb{R}$ concludes my formalisation's in this project. I was not able to finish my implementation of the Dedekind Reals within the permitted time.

# 7 Future Work

There are several immediate results that can be worked on in the short term. We would like to show that the Dedekind Reals are an Archimedean Field. This can be seen by showing that the Dedekind Reals both satisfy the Field Axioms, and the Archimedean principle.

The HoTT book [13] requires the following operations in order to satisfy the field axioms: addition, multiplication, inverse, minimum, maximum, the relations: less-than, less-than-or-equal and apartness. We also need the constants 0 and 1. We can prove the Archimedean principle for the Dedekind Reals using the definition of the less-than relation. I have defined the Field axioms in Agda, and wrote in the functions, with holes, that prove that the Dedekind Reals form an Archimedean field. The field axioms are defined easily in Agda, but proving that the Dedekind Reals satisfy the field axioms is a lengthy task. Consider addition of the Dedekind Reals. We define addition as in [13].

**Definition 7.0.1** (Addition of Dedekind Reals)**.** For $z \in \mathbf{R}$, let $L_z, R_z$ refer to the lower and upper cuts of z respectively.

Let $x, y \in \mathbf{R}$. Then define $x + y$ as:

$$L_{x+y}(p) := \exists(r, s \in \mathbb{R}) \text{ such that } L_x(r) \wedge L_y(s) \wedge p = r + s$$

$$R_{x+y}(q) := \exists(r, s \in \mathbb{R}) \text{ such that } R_x(r) \wedge R_y(s) \wedge q = r + s$$

The difficulty lies in proving that the definition is a cut. Solving this directly, this amounts to showing that $L_{x+y}(p), R_{x+y}(q)$ are inhabited, rounded, disjoint and located, which is not trivial. There is a process to follow; one must construct a proof outside of Agda and have a good understanding of how it works. It is not enough to simply write in the type that you need and hope that the proof follows easily, as one might do for some proofs of natural numbers and integers. With the length of lemma 6.4 in mind, it is clear there will be a lot of work involved in using all of information contained in a Dedekind Cut, along with implementing any required auxiliary proofs involving rational numbers. This process must be followed for all of the operations above, although the relations and constants are defined trivially. The first task following the conclusion of this project will be to implement all of the operations, and subsequently prove that they are an archidemean field.

Following this, I will show that the Dedekind reals are Dedekind complete, as defined in 6.2. I anticipate following the strategy shown in [13], section 11.2.3. At this point, with experience working on the operations of the Dedekind Reals, I do not think that proving completeness of the Dedekind Reals will pose much of a challenge, but of course there is always the opportunity for the formalisation to be surprising.

## 7.1 Cleaning up Code, and Equivalence Relation

Over the course of the project, I re-organised and re-wrote many proofs to be shorter and more elegant as I grew more experienced with Agda, and the style of Escardó's [5]. This is an ongoing process as the work develops, and in particular the proofs of the rationals could be condensed to be much clearer. I believe that if I explored developing more functions of the equivalence relation I introduced in definition 5.2.2, I would be able to eliminate code that re-occurs for functions of the rationals. Cleaning the code and further development of the equivalence will happen in tandem with work on the Dedekind reals.

There is also scope for re-implementing the Integers and the Rationals as a set-quotient. This would be more tedious than difficult, and could result in some proofs of the rationals becoming extremely short. It may also extend some other proofs. I am interested to see if the set-quotient variants (which are more closely related to the equivalence relation I defined on

the rationals) would reduce the overall tedium of working with the rational numbers, however this is a side goal. The different implementations are isomorphic, and wouldn't have an effect on the correctness of proofs.

### 7.2 Extension of uniformly continuous functions on a dense set

As mentioned, implementing the operations of the real numbers directly will not be easy. It is worth exploring an alternative method of defining these operations. If a function is uniformly continuously on a dense subset, then it has a continuous extension to the whole set. We have the rational numbers, which are dense can be embedded in the reals. By implementing the continuous extension theorem, it might be easier to implement operations and properties of the reals by extending the functions already implemented on the rationals. Exploring this alternative method will happen as part of proving that the Dedekind Reals form a complete archimedean field. I have written, with holes, the functions that will prove that the rationals form an archimedean ordered field, in anticipation of needing the field properties of rationals if the implementation of the continuous extension theorem is successful (note that the rationals are not complete).

### 7.3 Unit Interval satisfies the Convex Body Axioms

Following completion of the proof that the Dedekind Reals form a complete archimedean field, an extension of the project could show that the unit interval $[0, 1]$ satisfy the convex body axioms.

## 8 Conclusion

The goal of this project was to learn how to implement mathematical proofs within Agda. I thought the best way to accomplish this would be to work towards an implementation of the Dedekind Reals. I began by working on simple mathematical proofs, developing a deeper understanding of Agda, and more generally Type Theory. I encountered difficult challenges, including implementing HCF, working with more complex structures such as the rationals, and implementing proofs where the pen and paper proofs were not rigorous enough to translate directly.

In the process of formalising the construction of "simple" mathematical structures, I gained a deeper respect of the level of rigour required to write proofs of even more basic properties. I have noticed the difference between the completeness of the requirements of a proof assistant, and the assumptions usually made within the field of mathematics.

I have appreciated the assistance that Agda provides, with interactive proof solving in the form of holes in the code. Partial refinement of solutions, and providing the type of the goal can be extremely useful in compartmentalising proofs into easier to solve sub-proofs. I have noted that sometimes the goal type can lead to a focus on the wrong direction when attempting a proof, but with a pen and paper guide this can be minimised. It is certainly possible to write in the type of a proof, without checking that a proof actually exists.

Working towards the Dedekind Kind Reals, I formalised properties of order and addition of Natural Numbers, multiplication and it's properties, division, the extended euclidean algorithm. I have formalised integers and it's operations, proving that it is a group with respect to addition. I have partly formalised the rationals, including addition, multiplication and order, but some more work is required to prove that the rationals are a field.

I have learned about the connections between Type Theory and Mathematics, and noted the use of Univalent Type Theory in the form of Function Extensionality. I have discussed

the notion of assuming only as much as one needs when writing Mathematical Proofs, albeit breaking this notion for ease of use of FE and Propositional Truncation in the Dedekind Reals module.

With the formalisation of the Dedekind Reals incomplete, I have laid out a path forward to finish the implementation, which will involve a good amount of work, considering the complex structure of the Reals. Due to the vastness of the fields of Mathematics and Type Theory the scope for extending this project is wide, which gives me options to explore in my fourth year project.

# References


[1] Ana Bove and Peter Dybjer. "Dependent Types at Work". In: *Language Engineering and Rigorous Software Development: International LerNet ALFA Summer School 2008, Piriapolis, Uruguay, February 24 - March 1, 2008, Revised Tutorial Lectures*. Ed. by Ana Bove et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 57–99. ISBN: 978-3-642-03153-3. DOI: 10.1007/978-3-642-03153-3_2. URL: https://doi.org/10.1007/978-3-642-03153-3_2.

[2] *Definition:Field Axioms*. URL: https://proofwiki.org/w/index.php?title=Definition:Field_Axioms.

[3] Peter Dybjer. *An Introduction to Programming and Proving in Agda (incomplete draft)*. 2018. URL: http://www.cse.chalmers.se/~peterd/papers/AgdaLectureNotes2018.pdf.

[4] Martın Hötzel Escardó. "Introduction to Univalent Foundations of Mathematics with Agda". In: *CoRR* abs/1911.00580 (2019). arXiv: 1911.00580. URL: http://arxiv.org/abs/1911.00580.

[5] Martın Hötzel Escardó. *Various new theorems in constructive univalent mathematics written in Agda*. https://github.com/martinescardo/TypeTopology/. Agda development. Feb. 2019.

[6] Simon Goodwin. *1AC: Algebra 1*. 2019.

[7] A. G. (Alan G.) Hamilton. *Numbers, sets and axioms : the apparatus of mathematics / A.G. Hamilton.* eng. Cambridge, 1982.

[8] N. Kumar. "CONSTRUCTION OF NUMBER SYSTEMS". In: 2009.

[9] *Lecture 30: Number bases, Euclidean GCD algorithm, and strong induction*. URL: http://www.cs.cornell.edu/courses/cs2800/2017sp/lectures/lec30-strongind.html.

[10] Eric Lehman. *Mathematics for Computer Science*. London, GBR: Samurai Media Limited, 2017. ISBN: 9789888407064.

[11] Ulf Norell. "Dependently Typed Programming in Agda". In: *Advanced Functional Programming: 6th International School, AFP 2008, Heijen, The Netherlands, May 2008, Revised Lectures*. Ed. by Pieter Koopman, Rinus Plasmeijer, and Doaitse Swierstra. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 230–266. ISBN: 978-3-642-04652-0. DOI: 10.1007/978-3-642-04652-0_5. URL: https://doi.org/10.1007/978-3-642-04652-0_5.

[12] "Termination Checking". In: (). URL: https://agda.readthedocs.io/en/v2.6.0.1/language/termination-checking.html.

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

[14] Andrea Vezzosi and Mörtberg. *Cubical Adga*. URL: https://github.com/agda/cubical/blob/master/Cubical/Data/Int/Properties.agda.