Formalising Real Number Computation and Real Analysis

Andrew Sneap - 1980030

MSci - Maths & Computer Science
Fourth Year Project - 40 Credits
Supervised by Martín Escardó and Co-supervised by Todd Waugh Ambridge
Overleaf Word Count: 9912

Abstract

Computer verified formalisation of mathematics is important. An investigation into the efficiency of exact real arithmetic in comparison to floating point arithmetic would be useful, in a world where disasters have been cause by floating point arithmetic. This project begins work on this investigation, by formalising Dedekind reals and proving properties of the reals, including arithmetic locatedness, and that the reals are a metric space. I conclude by discussing the progress, and how this project will be extended, with a view towards exact computation with real numbers.

Contents

1 Literature Review

The mathematical framework within which to work is an important choice. Zermelo-Fraenkel set theory (ZFC) is dominant in the modern world, characterised by the axiom of choice. The axiom says that given a potentially infinite collection of inhabited sets, we can construct a new set by choosing one element from each set in the collection [Bel21]. The axiom gives rise to proofs that are otherwise not provable in the system. The choice axiom has been shown to be equivalent to the theorem that every vector space has a basis. It is understandable why many mathematicians are reluctant to give up this axiom; it provides key results that are nice to have, so abandoning choice is not an appealing prospect.

Constructive mathematics is a framework in which every proof provides a witness. This is advantageous computationally, since constructive proofs provide an algorithm to obtain an explicit witness of the proof. It is usually impossible to find an algorithm for a proof which uses the axiom of choice. Constructive mathematicians may argue that the axiom should not be used, since it is not constructive, and what use is a proof if you cannot provide an example of its application?

In some constructive frameworks, by adding the axiom of choice to we obtain classical mathematics. One might view constructive mathematics as embedded in classical mathematics, although this is not possible for some flavours of of constructive mathematics, for example Brouwer's intuitionism. Constructive mathematics has drawbacks. Proofs which rely on using the axiom of choice must be reformulated to avoid choice, sometimes resulting in weaker theorems. Constructive arguments may turn out to be more complex. On the other hand, the HoTT Book [Uni13, Introduction - Constructivity] discusses how this is not usually a problem. Definitions can be adapted, theorems can be made constructed, and constructive variants of proofs usually shed light on or provide new insights onto the crux of a proof.

Classical mathematics has had much more development, since constructive mathematics is a young field. There has been a lot of research in the area, since Brouwer begun pioneering intuitionism in 1907 [Att20]. For example, Bishop made major contributions to constructive analysis [BB87] followed by Troelstra [Tro73], while progress is being made by multiple authors in constructive algebra [LQ15].

Type theory is another young field living in the intersection of mathematics, computer science and logic. In the early 20th century, Russel introduced a system known as type theory to avoid the growing number of paradoxes in mathematics [Coq18]. There has since been a huge amount of research in the field, and correspondences have been established between type theory, mathematics and logic. In 1980, Howard explicitly stated this correspondence [How80], which has since been extended to include category theory interpretations of type theory, and an even younger field known as homotopy type theory (see [Uni13]).

One alternative foundation of mathematics is known as intuitionistic type theory, or Martin-Löf type theory (MLTT) after its founder Per Martin Löf. This type theory is aiming to be to constructive mathematics what ZFC is to classical mathematics [DP20]. One model of MLTT is known as univalent type theory, introduced by Voevodsky, which is characterised by the univalence axiom.

The goal of univalent type theory is pure; as Grayson put it, univalent foundations may provide "a language for mathematics invariant under equivalence and thus freed from irrelevant details and able to merge the results of mathematicians taking different but equivalent approaches" [Gra18]. The applications of such a foundation are clear; using proof assistants such as Agda and Coq to verify the *correctness* of a proof eliminates this burden for mathematicians. Moreover, while the theory is borne out of constructive thinking, it is not limited to constructive mathematics. By adding certain axioms to the system, one can work in a classical mathematical foundation, allowing one to verify the

correctness of classical proofs.

The power of this system goes even further, considering the consequence of writing propositions as types. By constructing an inhabitant of a type, we have a proof of the type. But writing this proof in a proof assistant allows us to extract a program to compute it, which is unquestionably useful to mathematicians. By writing proofs in such a language we can produce algorithms which are verifiably correct. Widely accepted incorrect proofs [httb] are not unheard of.

My development will allow for exact computation of the reals up to an arbitrary interval of error. I can therefore guarantee the correctness of computation with the reals. This is an important feature, since computation drives the modern world, and numerical errors in calculation can have catastrophic consequences. In 1996, the Ariane 501 exploded at an estimated cost of $300 million [Dow97], due to an overflow of floating point arithmetic. More examples can be found, the Patriot Rocket disaster for one, resulting in the loss of life of 28 individuals [Off]. Stevenson discusses in "A critical look at quality in large-scale" simulations [Ste99] how these problems arise, and how important validation is in the context of software. Also discussed is the dichotomy of the *cost* of validation.

Numerical analysis is the study of algorithms with approximate solutions to problems. To consider one example, I am studying numerical linear algebra as a module this year, and the two most important factors when considering algorithms is the time complexity of the algorithm, and the stability of the algorithm. If the algorithm is unstable, then the solution may be far away from the true solution, due to the buildup of errors in the floating point arithmetic during the algorithm. By using exact computation, we can avoid all instability, giving the answer to whichever degree of precision the user requires.

Exact computation is not free. As discussed in [Ste99], "time is money", and there are many scenarios in which approximations are more suitable. Efficiency is important, but for applications such critical nuclear power plant software, or involved in the design of the structures in the plant, safety-critical code may be worth the price. It is worth investigating the efficiency of exact computation against floating point arithmetic.

With the above in mind, writing a constructive library for the Dedekind reals is an exciting project for me. It poses various challenges, one being that I cannot simply copy and paste classical proofs from literature, and with a background in classical mathematics it forces me to think in a constructive way. I have to take care with definitions to ensure they hold constructively, and adapt proofs where necessary to avoid classical arguments. I have the added benefit in that my supervisor has a library of univalent type theory [Esc19], and so my work will contribute to an active development in a young field at the intersection of mathematics and computer science.

2 Background

2.1 Personal Background

I am a fourth year student undertaking a 40 credit MSci project. I also completed a 40 credit project in my third year, as required for my joint Mathematics and Computer Science degree: `https://adsneap.github.io/CSProject`. In my third year project, I learned how to implement mathematical proofs in Agda. The secondary goal, which was not achieved due to time constraints, was to show that the Dedekind reals are a Dedekind-complete Archimedean ordered field. I defined the Dedekind reals, by first constructing the integers and rationals.

The two projects are built on top of Escardo's Type Topology library [Esc19]. I love working in the intersection of computer science and mathematics, and so formalising mathematics using type theory as the framework is very appealing to me, and the main reason I chose to continue working in this area.

2.2 Agda

This library is written in the dependently type programming language Agda. Similarly to Coq and Lean, Agda may be used as a proof assistant. At it's core, Agda is a functional programming language, and shares many similarities with Haskell. Like Haskell, Agda is strongly typed, but differs from

Haskell in that it allows for dependent types. Consequently, Agda is a suitable language for proving mathematical theorems.

Mathematical statements are represented by types, and functions which produce an output of a type are seen as proofs of this type. Dependent types give us the flexibility to express properties of programs [Unkb]. Strong typing means that there is no room for ambiguity. It is enforced that every term has a type, so there is no scope for problematic calls to functions with inputs of inappropriate types. These two properties of Agda together are powerful. We can ensure that a program developed in Agda conforms to a specification by imposing the specification within the type of the program.

To give an explicit example, we can write the specification that a list is sorted. In Haskell, you can write a function which sorts a list, but in Agda we can go further and *prove* that the function sorts the list. This extends to mathematical statements. We can write the statement "there exists a natural number between 70 and 100, which is divisible by 24" as a type, and a proof of this statement gives an algorithm for finding such a number which is guaranteed to be correct.

In this project, I am building on top of my supervisor's Type Topology library, which is a library of "various new theorems in univalent mathematics written in Agda" [Esc19]. The library is deliberately minimal and assumes as little as possible for each proof, using "principles of HoTT/UF, ...and classical mathematics as explicit assumptions for the theorems ...that require them". This library provides me with the mathematical language in which I write my proofs. I write proofs in Agda, in MLTT. By the Curry-Howard correspondence, we have a mathematical interpretation of the proofs.

2.3 Mathematics in Type Theory

In type theory, every object has a type. We declare types in the following way:

```
data ℤ : 𝒰₀ ̇ where
  pos    : ℕ → ℤ
  negsucc : ℕ → ℤ
```

The type ℤ defines the integers. There are two constructors, pos and negsucc which are functions that allow us to construct integers. For example, by applying the natural number zero to the function pos, we obtain the integer pos zero which has type ℤ, denoted as pos zero : ℤ. The integers live in the lowest universe, which in TypeTopology is denoted by $\mathcal{U}_0$. Universes are sometimes called Set. A universe is a type whose elements are also types.

I will now introduce the types that correspond to the language of mathematics, described by the Curry-Howard correspondence.

- The unit type 𝟙 has a unique element ⋆ : 𝟙. We can interpret this type as an encoding of truth.
- The empty type 𝟘 has no inhabitants. A function not-true : A → 𝟘 is a proof that the type A has no inhabitants. If it is a statement, it is a proof that the statement is not true.
- The plus type _+_ which has two constructors, inl : X → X + Y, and inr : Y → X + Y. This encapsulate the disjoint union of two types, and equivalently logical disjunction.
- The sigma type, which can be expressed in the form Σ x : X , b. This is a dependent type, because b may depend on x. Logically, this may be viewed as the statement that there exists some x, such that b(x) is true.
- The cartesian product _×_ is a special case of the sigma type, where b does not depend on x, and can be expressed as X × B. This is viewed as conjunction.
- The arrow → represents implication.
- Equality is captured by the rather subtle identity type _≡_. There is a single constructor refl : (x : X) → x ≡ x. This can be read as "for all x of type X, by definition, x is equal to x".

I also make use of three assumptions within this development, which cannot be proved within this particular framework.

- Function extensionality is the assumption that functions which are pointwise equal, then they are equal.
- Propositional extensionality is the assumption that if two types P and Q are propositions (statements, or subsingletons), and P → Q, and Q → P, then P ≡ Q.
- Propositional truncation allows us to truncate types. Working constructively, we always provide inhabitants for our proofs. There are situations where we want to "forget" such a witness, because we do not want to know the particular witness, but do want to know that and inhabitant of the type exists.

Assuming the above types does not pose a problem to computation. Algorithms may be produced without assumptions, but they are needed to make observations about the properties of the algorithms, and in particular the correctness of the algorithm. Given a specification, you could say "here is a value x, and assuming functional extensionality holds here is the proof that x satisfies the specification".

To introduce how proofs work before delving into the realm of numbers, first consider the following example.

```
data List (X : 𝒰₀ ̇) : 𝒰₀ ̇ where
  []  : List X
  _::_ : X → List X → List X
map : {X Y : Type} → (X → Y) → List X → List Y
map f [] = []
map f (x :: xs) = f x :: map f xs
```

We have defined here the type of lists, and a map function which maps each element of a list to a new element produced by a function f. We can prove properties about the map function.

```
map-id-preserves-list : {X : Type} (xs : List X) → map id xs ≡ xs
map-id-preserves-list [] = refl
map-id-preserves-list (x :: xs) = ap (x ::_) (map-id-preserves-list xs)

map-composition-proof : {X Y Z : Type} (g : Y → Z) (f : X → Y) (xs : List X)
  → map (λ x → g (f x)) xs ≡ map g (map f xs)
map-composition-proof g f [] = refl
map-composition-proof g f (x :: xs) = ap (g (f x) ::_) (map-composition-proof g f xs)
```

The first is a proof that mapping a list with the identity function doesn't change the list. The proof is inductive, first considering the base case of an empty list, and then the inductive case using an inductive hypothesis. The second proves mapping the composition of two functions is the same as mapping each function successively on a list.

These proofs are simple, but the idea is powerful. We could continue this way, building up a library which provides list sorting algorithms, but also provides proof that the algorithms actually produce sorted lists, which provides peace of mind to any end users of such algorithms.

2.4 Viewing Code

Any code displayed in this report is hyperlinked, and refers to my github pages rendering of the code. This is the preferred method of viewing the code. The HTML code is produced by the Agda compiler using my so called "literate Agda" files. Agda will only produce the HTML code if the files compile successfully, so there is an implicit verification that all of the code referred to in this report is correct.

I have verified that my code compiles with Agda version 2.6.2, using TypeTopology commit: https://github.com/martinescardo/TypeTopology/tree/e2e635f61fe2ac6faeecad5880cbbb9842248e03. The code may be navigated using emacs, which can be installed by following the instructions on the Agda installation webpage: https://agda.readthedocs.io/en/latest/getting-started/installation.html. This is not necessary, and I recommend viewing the code using the HMTL rendering. The index of this rendering lists the files I have created during this project. For completeness, the files are also listed below.

- AndrewIndex
- NaturalsOrderExtended
- NaturalsMultiplication
- NaturalsDivision
- HCF
- IntegersB
- IntegersAbs
- IntegersAddition
- IntegersOrder
- IntegersDivision
- IntegersHCF
- IntegersMultiplication
- IntegersNegation
- ncRationals
- ncRationalsOperations
- ncRationalsOrder
- FieldAxioms
- Rationals
- RationalsAbs
- RationalsAddition
- RationalsExtension
- RationalsField
- RationalsLimits
- RationalsMinMax
- RationalsMultiplication
- RationalsNegation
- RationalsOrder
- DedekindReals
- DedekindRealsAddition
- DedekindRealsProperties
- MetricSpaceAltDef
- MetricSpaceRationals
- MetricSpaceDedekindReals
- ContinuousExtensionTheorem
- FieldRationals

The progress history of my code can be seen in the commit history of the main branch of my Reals repository: https://github.com/adsneap/Reals.

3 Dedekind Reals

3.1 Goals

In my third year project, I defined the Dedekind reals, and showed that there is an embedding from rationals to reals. The definition is given.

$$\text{inhabited-left} : (L : \mathcal{P}\ \mathbb{Q}) \to \mathcal{U}_0^{\cdot}$$
$$\text{inhabited-left}\ L = (\exists\ p : \mathbb{Q}\ ,\ p \in L)$$

$$\text{inhabited-right} : (R : \mathcal{P}\ \mathbb{Q}) \to \mathcal{U}_0^{\cdot}$$
$$\text{inhabited-right}\ R = (\exists\ q : \mathbb{Q}\ ,\ q \in R)$$

$$\text{rounded-left} : (L : \mathcal{P}\ \mathbb{Q}) \to \mathcal{U}_0^{\cdot}$$
$$\text{rounded-left}\ L = (x : \mathbb{Q}) \to (x \in L \Leftrightarrow (\exists\ p : \mathbb{Q}\ ,\ (x < p) \times p \in L))$$

$$\text{rounded-right} : (R : \mathcal{P}\ \mathbb{Q}) \to \mathcal{U}_0^{\cdot}$$
$$\text{rounded-right}\ R = (x : \mathbb{Q}) \to x \in R \Leftrightarrow (\exists\ q : \mathbb{Q}\ ,\ (q < x) \times q \in R)$$

$$\text{disjoint} : (L\ R : \mathcal{P}\ \mathbb{Q}) \to \mathcal{U}_0^{\cdot}$$
$$\text{disjoint}\ L\ R = (p\ q : \mathbb{Q}) \to p \in L \times q \in R \to p < q$$

$$\text{located} : (L\ R : \mathcal{P}\ \mathbb{Q}) \to \mathcal{U}_0^{\cdot}$$
$$\text{located}\ L\ R = (p\ q : \mathbb{Q}) \to p < q \to p \in L \vee q \in R$$

```
isCut : (L R : 𝒫 ℚ) → 𝒰₀̇
isCut L R = inhabited-left L
          × inhabited-right R
          × rounded-left L
          × rounded-right R
          × disjoint L R
          × located L R

ℝ : 𝒰₁̇
ℝ = Σ (L , R) : 𝒫 ℚ × 𝒫 ℚ , isCut L R
```

Note that in the definition of the reals, we use of the truncated sigma type ∃ and truncated disjunction type ∨. This is necessary, because we cannot decide if arbitrary real numbers are equal.
The natural goal is to show that this definition of the Reals, equipped with the usual addition and multiplication operators satisfies the constructive field axioms, and further show that the field is complete and Archimedean. This describes the reals as we view them intuitively, as a continuous line with no gaps stretching infinitely in two directions. This can be broken up nicely into subgoals, listed below.

- Show that the Dedekind reals are Archimedean
- Define order, apartness relations
- Define addition, multiplication and inverse operations
- Show that the Dedekind reals equipped with the above satisfy the constructive ordered field axioms
- Show that the Dedekind reals are Dedekind-complete

3.2 Order and Apartness Relations

The definitions for order and non-strict order are intuitive. I define them as in the HoTT Book [Uni13, Section 11.2.1].

```
_<ℝ_ : ℝ → ℝ → 𝒰₀̇
x <ℝ y = ∃ q : ℚ , (x < q) × (q < y)
```

The notion behind strict order is that if we can find some rational between two reals x and y, then $x < y$. Note the use of the overloaded operators; for strict order we have two distinct relations which compare rationals and reals. The first says that q is in the upper cut of x, and the second that says q is in the lower cut of y. This is useful syntactic sugar, since it is more intuitive to think of a rational being less or greater than a real number, rather than belonging to the lower or upper cut of a real.

With this definition, we can now prove that strict order of real numbers is transitive.

```
ℝ<-trans : (x y z : ℝ) → x < y → y < z → x < z
ℝ<-trans x y z x<y y<z = ∥∥-functor I (binary-choice x<y y<z)
  where
    I : (Σ k : ℚ , x < k × k < y)
      × (Σ l : ℚ , y < l × l < z)
      → Σ m : ℚ , x < m × m < z
    I ((k , (x<k , k<y)) , l , (y<l , l<z)) = k , (x<k , k<z)
      where
        k<l : k < l
        k<l = disjoint-from-real y k l (k<y , y<l)
        k<z : k < z
        k<z = rounded-left-c (lower-cut-of z) (rounded-from-real-L z) k l k<l l<z
```

The proof is intuitive. First, because we are writing a function between truncated types, we can use ∥∥-functor to escape the truncation, and use the information hidden by the truncation. Now we can refer to explicit values for rationals in between reals, which are used to prove the transitivity property.

It is not always possible to use ∥∥-functor, for example functions which return a rational or real number, because these are not truncated types. We can only escape from a truncation if the function returns a propositional type. This is intuitive mathematically, since propositions are types with a single inhabitant, so we cannot "forget" this.

The Archimedean property has many equivalent definition, but I choose to define it in the same way as the HoTT book [Uni13, Theorem 11.2.6].

```
ℝ-archimedean : (x y : ℝ)
              → x < y
              → ∃ q : ℚ , (x < q) × (q < y)
ℝ-archimedean x y l = l
```

My reason for choosing this over any other definition (you can view a few variations here [22]), is that the proof is automatic, simply by the definition of strict order. An extension of this project could define more variations of the property and verify they are equivalent for this definition of real numbers.

The apartness relation encapsulates when two reals are not equal.

```
_♯_ : (x y : ℝ) → 𝓤₀ ̇
x ♯ y = x < y ∨ y < x

apartness-gives-inequality : (x y : ℝ) → x ♯ y → ¬ (x ≡ y)
apartness-gives-inequality x y apart e = ∥∥-rec 𝟘-is-prop I apart
  where
    I : ¬ (x < y ∔ y < x)
    I (inl l) = ∥∥-rec 𝟘-is-prop III II
      where
        II : x < x
        II = transport (x <_) (e ⁻¹) l
        III : ¬ (Σ q : ℚ , (x < q) × (q < x))
        III (q , x<q , q<x) = ℚ<-not-itself-from-ℝ q x (q<x , x<q)
    I (inr r) = ∥∥-rec 𝟘-is-prop III II
      where
        II : y < y
        II = transport (y <_) e r
        III : ¬ (Σ p : ℚ , (p > y) × (p < y))
        III (p , y<p , p<y) = ℚ<-not-itself-from-ℝ p y (p<y , y<p)
```

The converse is not true. We cannot decide which of two non-equal reals is larger constructively, because this is equivalent to using the axiom of choice. One of the classical field axioms states that every number not equal to 0 has a multiplicative inverse, but this is not sufficient constructively.

The defined apartness relation is sufficient to preserve the property that a number is not equal to 0, and this small modification to the field axioms allows us to proceed constructively. Troelstra and Dalen say [TD88, Page 256] that "apartness is a positive version of inequality...". This is analogous to how a classically one might say that a set is "non-empty", whereas constructively one would say that a set is "inhabited".

I have also defined non strict order, and proved some more properties of order of reals numbers. The types are listed here without proof.

```
_≤ℝ_ : ℝ → ℝ → 𝓤₀ ̇
x ≤ℝ y = (q : ℚ) → q < x → q < y
ℝ≤-trans : (x y z : ℝ) → x ≤ y → y ≤ z → x ≤ z
ℝ<-≤-trans : (x y z : ℝ) → x < y → y ≤ z → x < z
ℝ-less-than-or-equal-not-greater : (x y : ℝ) → x ≤ y → ¬ (y < x)
ℝ-less-than-not-greater-or-equal : (x y : ℝ) → x < y → ¬ (y ≤ x)
ℝ-not-less-is-greater-or-equal : (x y : ℝ) → ¬ (x < y) → y ≤ x
ℝ≤-<-trans : (x y z : ℝ) → x ≤ y → y < z → x < z
ℝ-less-than-not-itself : (x : ℝ) → x ≮ x
ℝ-zero-less-than-one : 0ℝ < 1ℝ
```

```
ℝ-zero-apart-from-one : 0ℝ ♯ 1ℝ
embedding-preserves-order : (p q : ℚ) → p < q → ι p < ι q
weak-linearity : (x y z : ℝ) → x < y → x < z ∨ z < y
```

3.3 Operations

At this stage, we have to decide how to implement operators for real numbers. At the very least we need addition and multiplication. We would like to have negation, to find the additive inverse of a real. We hope to avoid division, since I avoided an explicit implementation this when formalising my rationals. In the long term I would like to add division as an operator for both reals and rationals, for the purposes of exact computation.

In my first naive attempt, I attempted to directly implement addition using the cuts defined in the HoTT book.

```
L-z R-z : 𝒫 ℚ
L-z p = (∃ (r , s) : ℚ × ℚ , r ∈ L-x × s ∈ L-y × (p ≡ r ℚ+ s)) , ∃-is-prop
R-z q = (∃ (r , s) : ℚ × ℚ , r ∈ R-x × s ∈ R-y × (q ≡ r ℚ+ s)) , ∃-is-prop
```

Inhabitedness, roundedness and disjointedness can be proved fairly easily; each follows from the respective property on x and y, for example inhabitedness of z follows from inhabitedness of x and y. Inhabitedness and disjointness are shown as examples.

```
inhabited-left-z : ∃ q : ℚ , q ∈ L-z
inhabited-left-z = ∥∥-rec ∃-is-prop δ (binary-choice inhabited-left-x inhabited-left-y)
  where
    δ : (Σ p : ℚ , p ∈ L-x) × (Σ q : ℚ , q ∈ L-y) → ∃ z : ℚ , z ∈ L-z
    δ ((p , l-x) , q , l-y) = ∣ (p ℚ+ q) , (∣ (p , q) , l-x , l-y , refl ∣) ∣

inhabited-right-z : ∃ q : ℚ , q ∈ R-z
inhabited-right-z = ∥∥-rec ∃-is-prop δ (binary-choice inhabited-right-x inhabited-right-y)
  where
    δ : (Σ p : ℚ , p ∈ R-x) × (Σ q : ℚ , q ∈ R-y) → ∃ z : ℚ , z ∈ R-z
    δ ((p , r-x) , q , r-y) = ∣ (p ℚ+ q) , (∣ (p , q) , (r-x , r-y , refl) ∣) ∣


disjoint-z : disjoint L-z R-z
disjoint-z p q (α , β) = ∥∥-rec (ℚ<-is-prop p q) δ (binary-choice α β)
  where
    δ : (Σ (r , s) : ℚ × ℚ , r ∈ L-x × s ∈ L-y × (p ≡ r ℚ+ s))
        × (Σ (r' , s') : ℚ × ℚ , r' ∈ R-x × s' ∈ R-y × (q ≡ r' ℚ+ s'))
        → p < q
    δ (((r , s) , l-x , l-y , e₁) , ((r' , s') , r-x , r-y , e₂)) = goal
      where
        I : r < r'
        I = disjoint-x r r' (l-x , r-x)

        II : s < s'
        II = disjoint-y s s' (l-y , r-y)

        goal : p < q
        goal = transport₂ _<_ (e₁ ⁻¹) (e₂ ⁻¹) (ℚ<-adding r r' s s' I II)
```

Locatedness can not be proven so easily. We want to show that for any $a < b$, either there exists $p < x$, $u < y$ such that $a = p + u$, or that there exists $x < q$, $y < v$ such that $b = q + v$. As a naive first

approach, since the other conditions follow from their respective conditions on x and y, we look at the locatedness of x and y. We are given $a < b$, so we obtain that $a < x$ or $x < b$, and $a < y$ or $y < b$. One might stare at these facts for a long time before realising that this is not enough information by itself to prove the locatedness of z. Bauer and Taylor remark in "The Dedekind reals in abstract Stone duality" [BT09, Remark 11.14] that "locatedness always seems to be the most difficult, because we need to calculate the result arbitrarily closely".

The problem is that we don't have enough information about how *close* a and b are to either x or y. We can move arbitrarily small distances to the left of a, since x is rounded and for all $\epsilon > 0$, $a - \epsilon < a$, but we cannot guarantee that $\epsilon < y$. By locatedness of y, we can find that $a - \epsilon < y$ or $y < a$. In the first case we need that either $\epsilon < x$, and in the second we have no extra useful information, so in neither case we cannot proceed.

To tackle this problem, I used an idea from the same paper [BT09, Proposition 11.15]. This proposition introduces the idea of arithmetic locatedness of the reals.

```
ℝ-arithmetically-located : (z : ℝ)
                         → (p : ℚ)
                         → 0ℚ < p
                         → ∃ (x , y) : ℚ × ℚ , (x < z) × (z < y) × 0ℚ < (y - x) × (y - x) < p
```

The intuition behind arithmetic locatedness of the reals is that we can locate a real number x to any degree of precision. For any $\epsilon > 0$, we can find an interval (p, q) with x located somewhere in this interval. With arithmetic locatedness of the reals, we could apply lemma 11.16 [BT09] to complete the addition operator. By stating arithmetic locatedness in Agda, leaving the proof as a hole, I managed to finish the implementation of addition directly.

I cannot claim to have formalised addition of reals without proving arithmetic locatedness, so this was clearly the next task. A full discussion can be found later in this report, but for now it suffices to say that the first attempt was not successful. With the experience of implementing the embedding from rationals to reals, and addition of reals sans the proof of arithmetic locatedness, I decided it was worth trying a different approach to implement the operations.

Addition is just one operator, and it proved to be difficult to solve. For the purpose of proving that the Dedekind reals satisfy the field axioms, addition, multiplication and negation would suffice, but it's expected that problems similar to arithmetic locatedness would arise. With a view towards expanding this project in the future, where we might want to implement many more functions on the reals, it would be a good idea if we had a frameworks to produce operations and functions without having to define these cuts directly, without having to construct proofs that each cut satisfies the definition of a Dedekind cut.

3.4 Extensions

Extensions of functions are an intuitive concept. Suppose we have sets X, Y and Z, with $X \subset Y$, and a function $f : X \to Z$. A function $g : Y \to Z$ is considered an extension of f if g agrees with f for every $x \in X$. We would like to implement addition, multiplication and inverses on the reals. We notice that we already have these operations for rationals, and that the rationals are embedded in the reals. The question is, then, is it possible to extend these functions from the rationals to the reals?

By imposing some conditions on the functions we want to extend, we can find a unique extension to the new space. In this section I will discuss two extensions. My supervisor provided me with a reference to Simmons, "Introduction to Topology and Modern Analysis", with the goal of working towards Theorem D [Sim83, Page 78].

Theorem (Continuous Extension Theorem)**.** *Let X be a metric space, let Y be a complete metric space, and let A be a dense subspace of X. If f is a uniformly continuous mapping of A into Y, then f can be extended uniquely to a uniformly continuous mapping g of X into Y.*

In our case, X and Y are both the reals, and A is the rationals. The idea is that we can take continuous extensions from the rationals to the reals, and uniquely extend them to functions from

the rationals to the reals. By taking addition, multiplication and inverse functions defined for the rationals and extending them, we avoid having to directly implement these operations. There is an implicit hope that since I have already proven that the rationals are field, it will follow easily that the extensions of these functions are also a field.

To prove the continuous extension theorem in Agda, I need to define each concept mentioned in the theorem. Metric spaces, continuity, Cauchy and convergent sequences and completeness of a metric space all need to be defined in Agda. These are all familiar concepts to me from my studies in classical mathematics, but I was not familiar with these concepts in the constructive sense, and would go on to discover the challenges that the constructive formalisation posed.

The advantage of having such a theorem is clear. A completed proof allows us to extend *any* continuous rational function to the reals, and although this does not encapsulate all of the functions one might want to study in mathematics, it is certainly a good foundation and would allow us to look at polynomials, exponentials, logarithms and other interesting functions.

The continuous extension theorem is a daunting prospect to consider solving. The Simmons proof makes use of the strictly classical theorem that every real number is the limit of some converging sequence of rationals. It is not possible to prove this constructively, because the proof would require the axiom of choice. I could not find any examples of constructive adaptations of this proof in literature, so it was difficult for me to envision how the proof might work.

I wanted to familiarise myself with the notion of extensions, and so as a proof of the concept of the extension of functions, I will now discuss a simpler extension, which I will call the monotonic extension theorem.

3.5 Monotonic Extension Theorem

The continuous extension theorem imposes the condition of continuity onto functions. This is quite general, which is an attractive property in mathematics. Monotonicity is less general than continuity. A proof that we can extend monotonic functions is less exciting, especially because even multiplication is not a monotonic function, but nevertheless would be good to have even as a proof of concept that functions can be extended to the reals. The monotonic extension theorem was suggested to me by my supervisor, who sketched out a pen and paper proof for me.

We want to prove the following: if f is a rational valued function on the rationals, and f is a monotonic function, and there exists a function g where f and g are bijective, then there exists an unique extension $\hat{f}$ where $\hat{f}$ is a real valued function on the reals.

We begin by proving a lemma. If f and g are bijective, and f is strictly monotone, then

```
bijection-preserves-monotone : (f g : ℚ → ℚ) → 𝓤₀ ̇
bijection-preserves-monotone f g = ((p q : ℚ) → p < q ⇔ f p < f q)
                                 → ((r : ℚ) → (g (f r) ≡ r) × (f (g r) ≡ r))
                                 → ((p q : ℚ) → p < q ⇔ g p < g q)


bijective-and-monotonic : (f : ℚ → ℚ)
                        → (g : ℚ → ℚ)
                        → bijection-preserves-monotone f g
bijective-and-monotonic f g f-preserves-order f-g-bijection = γ
  where
    γ : (p q : ℚ) → p < q ⇔ g p < g q
    γ p q = ltr , rtl
      where
        apply-order-preversation : g p < g q ⇔ f (g p) < f (g q)
        apply-order-preversation = f-preserves-order (g p) (g q)

        ltr : p < q → g p < g q
        ltr l = (rl-implication apply-order-preversation) i
          where
```

```
      i : f (g p) < f (g q)
      i = transport₂ _<_ (pr₂ (f-g-bijection p) ⁻¹) (pr₂ (f-g-bijection q) ⁻¹) l

  rtl : g p < g q → p < q
  rtl l = transport₂ _<_ (pr₂ (f-g-bijection p)) (pr₂ (f-g-bijection q)) i
    where
      i : f (g p) < f (g q)
      i = (lr-implication apply-order-preversation) l
```

We can now define a function which produces an extension $\hat{f}$ given monotonic function f with a bijection g.

```
f→f̂ : (f g : ℚ → ℚ)
    → ((p q : ℚ) → p < q ⇔ f p < f q)
    → ((r : ℚ) → (g (f r) ≡ r) × (f (g r) ≡ r))
    → ℝ → ℝ
f→f̂ f g f-order-preserving f-g-bijective
  ((L , R) , inhabited-left-x , inhabited-right-x , rounded-left-x , rounded-right-x , disjoint-x , located-x)
    = (left , right) , inhabited-left' , inhabited-right' , rounded-left' , rounded-right' , disjoint' , located'
  where
    x : ℝ
    x = ((L , R) , inhabited-left-x , inhabited-right-x , rounded-left-x , rounded-right-x , disjoint-x , located-x)

    left : 𝒫 ℚ
    left p = (g p ∈ L) , ∈-is-prop L (g p)
    right : 𝒫 ℚ
    right q = g q ∈ R , ∈-is-prop R (g q)

    inhabited-left' : inhabited-left left
    inhabited-left' = ∥∥-functor I inhabited-left-x
      where
        I : Σ p : ℚ , p ∈ L → Σ p' : ℚ , p' ∈ left
        I (p , p-L) = (f p) , transport (_∈ L) (pr₁ (f-g-bijective p) ⁻¹) p-L

    inhabited-right' : inhabited-right right
    inhabited-right' = ∥∥-functor I inhabited-right-x
      where
        I : Σ q : ℚ , q ∈ R → Σ q' : ℚ , q' ∈ right
        I (q , q-R) = f q , transport (_∈ R) (pr₁ (f-g-bijective q) ⁻¹) q-R

    rounded-left' : rounded-left left
    rounded-left' k = I , II
      where
        I : k ∈ left → ∃ p : ℚ , (k < p) × p ∈ left
        I k-L = ∥∥-functor iii ii
          where
            i : f (g k) ≡ k
            i = pr₂ (f-g-bijective k)
            ii : ∃ q : ℚ , g k < q × q ∈ L
            ii = (pr₁ (rounded-left-x (g k))) k-L
            iii : Σ q : ℚ , g k < q × q ∈ L → Σ p : ℚ , k < p × p ∈ left
            iii (q , (l , q-L)) = f q , vii , vi
              where
                iv : g k < q → f (g k) < f q
                iv = pr₁ (f-order-preserving (g k) q)
                v : g (f q) ∈ L
                v = transport (_∈ L) (pr₁ (f-g-bijective q) ⁻¹) q-L
                vi : g (f q) ∈ L
                vi = transport (_∈ L) (pr₁ (f-g-bijective q) ⁻¹) q-L
                vii : k < f q
                vii = transport (_< f q) i (iv l)
```

```
    II : ∃ p : ℚ , k < p × p ∈ left → k ∈ left
    II e = ∥∥-rec (∈-is-prop left k) i e
      where
        i : Σ p : ℚ , k < p × p ∈ left → k ∈ left
        i (p , (l , p-L)) = iv | (g p) , iii , p-L |
          where
            ii : k < p ⇔ g k < g p
            ii = bijective-and-monotonic f g f-order-preserving f-g-bijective k p
            iii : g k < g p
            iii = (pr₁ ii) l
            iv : ∃ p' : ℚ , g k < p' × p' ∈ L → g k ∈ L
            iv = pr₂ (rounded-left-x (g k))

rounded-right' : rounded-right right
rounded-right' k = I , II
  where
    I : k ∈ right → ∃ q : ℚ , q < k × q ∈ right
    I k-R = ∥∥-functor ii i
      where
        i : ∃ q : ℚ , q < g k × q ∈ R
        i = pr₁ (rounded-right-x (g k)) k-R
        ii : Σ p : ℚ , p < g k × p ∈ R → Σ q : ℚ , (q < k) × q ∈ right
        ii (p , (l , p-R)) = (f p) , (transport (f p <_) iv iii) , transport (_∈ R) (pr₁ (f-g-bijective p) ⁻¹) p-R
          where
            iii : f p < f (g k)
            iii = (pr₁ (f-order-preserving p (g k))) l
            iv : f (g k) ≡ k
            iv = pr₂ (f-g-bijective k)
    II : ∃ q : ℚ , q < k × q ∈ right → k ∈ right
    II e = ∥∥-rec (∈-is-prop right k) i e
      where
        i : Σ q : ℚ , q < k × q ∈ right → k ∈ right
        i (q , (l , q-R)) = iv | (g q) , (iii , q-R) |
          where
            ii : q < k ⇔ g q < g k
            ii = bijective-and-monotonic f g f-order-preserving f-g-bijective q k
            iii : g q < g k
            iii = (pr₁ ii) l
            iv : ∃ q : ℚ , q < g k × q ∈ R → g k ∈ R
            iv = pr₂ (rounded-right-x (g k))

disjoint' : disjoint left right
disjoint' p q l = (pr₂ I) II
  where
    I : p < q ⇔ g p < g q
    I = bijective-and-monotonic f g f-order-preserving f-g-bijective p q
    II : g p < g q
    II = disjoint-x (g p) (g q) l

located' : located left right
located' p q l = III
  where
    I : p < q ⇔ g p < g q
    I = bijective-and-monotonic f g f-order-preserving f-g-bijective p q
    II : p < q → g p < g q
    II = pr₁ I
    III : g p ∈ L ∨ g q ∈ R
    III = located-x (g p) (g q) (II l)
```

And finally, we can prove that this function preserves the behaviour of f.

```
diagram-commutes : (f g : ℚ → ℚ)
                 → (f-order-preserving : ((p q : ℚ) → p < q ⇔ f p < f q))
                 → (f-g-bijective : ((r : ℚ) → (g (f r) ≡ r) × (f (g r) ≡ r)))
                 → (q : ℚ)
                 → (f→𝔣 f g f-order-preserving f-g-bijective ∘ ι) q ≡ (ι ∘ f) q
diagram-commutes f g f-order-preserving f-g-bijective q = ℝ-equality' ((𝔣 ∘ ι) q) ((ι ∘ f) q) I II III IV
  where
    𝔣 : ℝ → ℝ
    𝔣 = f→𝔣 f g f-order-preserving f-g-bijective

    I : (a : ℚ) → g a < q → a < f q
    I a b = transport (_< f q) ii i
      where
        i : f (g a) < f q
        i = (pr₁ (f-order-preserving (g a) q)) b
        ii : f (g a) ≡ a
        ii = pr₂ (f-g-bijective a)
    II : (a : ℚ) → a < f q → g a < q
    II a b = transport (g a <_) ii i
      where
        i : g a < g (f q)
        i = (pr₁ (bijective-and-monotonic f g f-order-preserving f-g-bijective a (f q))) b
        ii : g (f q) ≡ q
        ii = pr₁ (f-g-bijective q)
    III : (a : ℚ) → q < g a → f q < a
    III a b = transport (f q <_) ii i
      where
        i : f q < f (g a)
        i = (pr₁ (f-order-preserving q (g a))) b
        ii : f (g a) ≡ a
        ii = pr₂ (f-g-bijective a)
    IV : (a : ℚ) → f q < a → q < g a
    IV a b = transport (_< g a) ii i
      where
        i : g (f q) < g a
        i = (pr₁ (bijective-and-monotonic f g f-order-preserving f-g-bijective (f q) a)) b
        ii : g (f q) ≡ q
        ii = pr₁ (f-g-bijective q)
```

This proof of concept is successful. I proved that given a monotonic rational function with a bijection, I can extend it to the reals. Unfortunately, montonicity is a very strong condition. There are many functions which are not monotone, with multiplication being a particular example. This theorem is therefore not strong enough for me to obtain the operations I need to prove that the Dedekind reals are a field, and so we must look back to the continuous extension theorem.

As discussed before, in order to prove the continuous extension theorem, it's first necessary to formalise every concept included in the proof. We turn our attention to metric spaces.

4 Metric Spaces

We immediately run into a problem when trying to define a metric space. A metric space is simply a set (which may be empty), with a distance function known as a metric. There are many metrics one could choose, however, usually a metric d would be defined on a set X with $d : X \times X \to \mathbb{R}$. One of the axioms of a metric space is the triangle inequality, which requires that $d(x, z) \leq d(x, y) + d(y, z)$

for $x, y, z \in X$. But $d(x, y) + d(y, z)$ is addition the real numbers. The motivation to implement the continuous extension theorem was to *avoid* directly implementing addition of the reals, so we must consider a different approach.

We replace the real valued distance function with an equivalent definition which instead uses a rational to bound some distance function on the set.

```
m1a : {𝒰 : Universe} → (X : 𝒰 ̇) → (B : X → X → (ε : ℚ) → 0ℚ < ε → 𝒰₀ ̇) → 𝒰 ̇
m1a X B = (x y : X) → ((ε : ℚ) → (l : 0ℚ < ε) → B x y ε l) → x ≡ y

m1b : {𝒰 : Universe} → (X : 𝒰 ̇) → (B : X → X → (ε : ℚ) → 0ℚ < ε → 𝒰₀ ̇) → 𝒰 ̇
m1b X B = (x : X) → ((ε : ℚ) → (l : 0ℚ < ε) → B x x ε l)

m2 : {𝒰 : Universe} → (X : 𝒰 ̇) → (B : X → X → (ε : ℚ) → 0ℚ < ε → 𝒰₀ ̇) → 𝒰 ̇
m2 X B = (x y : X) → (ε : ℚ) → (l : 0ℚ < ε) → B x y ε l → B y x ε l

m3 : {𝒰 : Universe} → (X : 𝒰 ̇) → (B : X → X → (ε : ℚ) → 0ℚ < ε → 𝒰₀ ̇) → 𝒰 ̇
m3 X B = (x y : X) → (ε₁ ε₂ : ℚ)
                   → (l₁ : 0ℚ < ε₁)
                   → (l₂ : 0ℚ < ε₂)
                   → ε₁ < ε₂
                   → B x y ε₁ l₁
                   → B x y ε₂ l₂

m4 : {𝒰 : Universe} → (X : 𝒰 ̇) → (B : X → X → (ε : ℚ) → 0ℚ < ε → 𝒰₀ ̇) → 𝒰 ̇
m4 X B = (x y z : X) → (ε₁ ε₂ : ℚ)
                     → (l₁ : (0ℚ < ε₁))
                     → (l₂ : (0ℚ < ε₂))
                     → B x y ε₁ l₁
                     → B y z ε₂ l₂
                     → B x z (ε₁ + ε₂) (ℚ<-adding-zero ε₁ ε₂ l₁ l₂)

metric-space : {𝒰 : Universe} → (X : 𝒰 ̇) → 𝒰₁ ⊔ 𝒰 ̇
metric-space X =
  Σ B : (X → X → (ε : ℚ) → 0ℚ < ε → 𝒰₀ ̇) , m1a X B × m1b X B × m2 X B × m3 X B × m4 X B
```

The distance function allows us to define the distance between some x and y in terms of a positive ϵ sized ball. The first axiom m1a says that, for any x and y in the set, if they are ϵ close for *any* rational ϵ, then they are equal. This is equivalent to the standard formulation of this axioms which says that if $d(x, y) = 0$, then $x = 0$. The other conditions above are similarly analogous to the standard formulations of metric space axioms.

With this alternative definition, we can construct metric spaces which are defined in terms of only rational numbers, avoiding the use of addition of reals. Recalling the continuous extension theorem in Simmons, we require that both the rationals and the reals are metric spaces. We can see how the alternative axioms are used in practice by proving that the rationals are a metric space.

4.1 Rationals Metric Space

The distance relation for the rationals is defined using the standard metric.

```
B-ℚ : (x y ε : ℚ) → 0ℚ < ε → 𝒰₀ ̇
B-ℚ x y ε l = ℚ-metric x y < ε

ℚ-metric : ℚ → ℚ → ℚ
ℚ-metric p q = abs (p - q)
```

To define this, it was necessary to extend my implementation of the rationals to define the absolute value function, and formalise many properties of the absolute value function, which are listed here without proof.

```
abs-of-pos-is-pos : Fun-Ext → (p : ℚ) → 0ℚ ≤ p → abs p ≡ p
abs-of-pos-is-pos' : Fun-Ext → (p : ℚ) → 0ℚ < p → abs p ≡ p
ℚ-abs-neg-equals-pos : Fun-Ext → (q : ℚ) → abs q ≡ abs (- q)
ℚ-abs-inverse : Fun-Ext → (q : ℚ) → (abs q ≡ q) ∔ (abs q ≡ - q)
ℚ-positive-not-zero : Fun-Ext → (x a : ℕ) → ¬ (toℚ (pos (succ x) , a) ≡ 0ℚ)
ℚ-abs-is-positive : (q : ℚ) → 0ℚ ≤ abs q
ℚ-abs-zero-is-zero : Fun-Ext → (q : ℚ) → abs q ≡ 0ℚ → q ≡ 0ℚ
ℚ-abs-≤ : Fun-Ext → (q : ℚ) → (- (abs q) ≤ q) × (q ≤ abs q)
ℚ-abs-≤-unpack : Fun-Ext → (q ε : ℚ) → abs q ≤ ε → (- ε ≤ q) × (q ≤ ε)
ℚ≤-to-abs : Fun-Ext → (x y : ℚ) → (- y ≤ x) × (x ≤ y) → abs x ≤ y
ℚ<-to-abs : Fun-Ext → (x y : ℚ) → (- y < x) × (x < y) → abs x < y
ℚ-abs-<-unpack : Fun-Ext → (q ε : ℚ) → abs q < ε → (- ε < q) × (q < ε)
ℚ-triangle-inequality : Fun-Ext → (x y : ℚ) → abs (x + y) ≤ abs x + abs y
pos-abs-no-increase : Fun-Ext → (q ε : ℚ) → (0ℚ < q) × (q < ε) → abs q < ε
abs-mult : Fun-Ext → (x y : ℚ) → abs x * abs y ≡ abs (x * y)
ℚ-self-dist : Fun-Ext → (q : ℚ) → ℚ-metric q q ≡ 0ℚ
ℚ-metric-commutes : (p q : ℚ) → ℚ-metric p q ≡ ℚ-metric q p
inequality-chain-to-metric : (w y z : ℚ) → w ≤ y
                                         → y ≤ z
                                         → ℚ-metric w z ≡ ℚ-metric w y + ℚ-metric y z
inequality-chain-with-metric : (x y w z ε₁ ε₂ : ℚ) → w ≤ y
                                                   → y ≤ z
                                                   → ℚ-metric x y < ε₁
                                                   → ℚ-metric w z < ε₂
                                                   → ℚ-metric x z < (ε₁ + ε₂)
```

I had imagined that absolute values would be difficult to work with, but I found that the proofs were intuitive and this section of the project progressed in a timely fashion. One amusing proof is that absolute value functions are distributive over multiplication. There are many proofs that are easy to construct without having to refer to literature, but I could not figure out a way to prove this. By consulting text books, I found that it requires a simple case analysis of the inputs.

With the definitions and proofs above, I can prove that the rationals are a metric space with respect to the alternative axioms.

```
ℚ-m1a : m1a ℚ B-ℚ
ℚ-m1a x y f = I (ℚ≤-split fe 0ℚ (abs (x - y)) (ℚ-abs-is-positive (x - y)))
  where
    α : ℚ
    α = ℚ-metric x y
    I : (0ℚ < abs (x - y)) ∔ (0ℚ ≡ abs (x - y)) → x ≡ y
    I (inl z) = 𝟘-elim (ℚ<-not-itself α ζ)
      where
        ζ : α < α
        ζ = f α z
    I (inr z) = ii
      where
        i : (x - y) ≡ 0ℚ
        i = ℚ-abs-zero-is-zero fe (x - y) (z ⁻¹)
        ii : x ≡ y
        ii = x             ≡⟨ ℚ-zero-right-neutral fe x ⁻¹ ⟩
             x + 0ℚ        ≡⟨ ap (x +_) (ℚ-inverse-sum-to-zero fe y ⁻¹) ⟩
             x + (y - y)   ≡⟨ ap (x +_) (ℚ+-comm y (- y)) ⟩
             x + ((- y) + y) ≡⟨ ℚ+-assoc fe x (- y) y ⁻¹ ⟩
             x + (- y) + y ≡⟨ ap (_+ y) i ⟩
             0ℚ + y        ≡⟨ ℚ-zero-left-neutral fe y ⟩
             y             ∎
```

```
ℚ-m1b : m1b ℚ B-ℚ
ℚ-m1b x y l = transport (λ v → v < y) (ℚ-self-dist fe x ⁻¹) l

ℚ-m2 : m2 ℚ B-ℚ
ℚ-m2 x y ϵ l₁ B = transport (λ z → z < ϵ) (ℚ-metric-commutes x y) B

ℚ-m3 : m3 ℚ B-ℚ
ℚ-m3 x y ϵ₁ ϵ₂ l₁ l₂ l B = ℚ<-trans (ℚ-metric x y) ϵ₁ ϵ₂ B l

ℚ-m4 : m4 ℚ B-ℚ
ℚ-m4 x y z ϵ₁ ϵ₂ l₁ l₂ B₁ B₂ = conclusion α
  where
    α : abs ((x - y) + (y - z)) ≤ (abs (x - y) + abs (y - z))
    α = ℚ-triangle-inequality fe (x - y) (y - z)

    β : (abs (x - y) + abs (y - z)) < (ϵ₁ + ϵ₂)
    β = ℚ<-adding (abs (x - y)) ϵ₁ (abs(y - z)) ϵ₂ B₁ B₂

    δ : abs ((x - y) + (y + (- z))) ≡ abs (x - z)
    δ = ap abs (ℚ-add-zero fe x (- z) y ⁻¹)

    conclusion : abs ((x - y) + (y - z)) ≤ (abs (x - y) + abs (y - z)) → abs (x - z) < (ϵ₁ + ϵ₂)
    conclusion l = I (ℚ≤-split fe (abs ((x - y) + (y - z))) ((abs (x - y) + abs (y - z))) l)
      where
        I : (abs ((x - y) + (y - z)) < (abs (x - y) + abs (y - z)))
            ∔ (abs ((x - y) + (y - z)) ≡ abs (x - y) + abs (y - z))
            → abs (x - z) < (ϵ₁ + ϵ₂)
        I (inl l) = ℚ<-trans (abs (x - z)) ((abs (x - y) + abs (y - z))) (ϵ₁ + ϵ₂) γ β
          where
            γ : abs (x - z) < (abs (x - y) + abs (y - z))
            γ = transport (λ k → k < (abs (x - y) + abs (y - z))) δ l
        I (inr e) = transport (_< (ϵ₁ + ϵ₂)) (e ⁻¹ • δ) β

ℚ-metric-space : metric-space ℚ
ℚ-metric-space = B-ℚ , ℚ-m1a , ℚ-m1b , ℚ-m2 , ℚ-m3 , ℚ-m4
```

It would be nice in the future to prove that the alternative definition of the metric space is equivalent to the standard definition. This is not necessary, and stems from a desire to be similar to classical mathematics.

4.2 Dedekind Reals Metric Space

We now want to prove that the reals are a metric space. We first need a suitable distance relation for the Dedekind reals. My supervisor knew of such a distance function. We say that two reals are ϵ close if we can find a pair of rationals, one either side of each real such that the the distance between the furthest value on each side is less than ϵ.

```
B-ℝ : (x y : ℝ) → (ϵ : ℚ) → 0ℚ < ϵ → 𝒰₀ ̇
B-ℝ x y ϵ l =
  ∃ (p , q , u , v) : ℚ × ℚ × ℚ × ℚ , (p < x)
                                     × (u < y)
                                     × (x < q)
                                     × (y < v)
                                     × B-ℚ (min p u) (max q v) ϵ l
```

This definition is equivalent to the standard definition of the metric. The intuition behind the definition is that if we locate the two reals to some level of precision, then the reals are at most as far away as the bounds of the precision. For example, if $a < x < b$, and $c < y < d$, then x and y are within the interval $(min\ a\ c\ ,\ max\ b\ d)$, so the difference between them is at most $|min\ a\ c - max\ b\ d|$.

To define this metric, I need the min and max operations on rational numbers. It was surprisingly difficult to formalise these operations. This was a case where working within type theory complicated the implementation of the operations. I wanted to define the operations using $<$ and $\leq$, and the fact that rationals number are trichotomous with respect to order, because I have already finished these proofs.

```
max' : (x y : ℚ) → x < y ∔ (x ≡ y) ∔ y < x → ℚ
max' x y (inl _) = y
max' x y (inr _) = x

max : ℚ → ℚ → ℚ
max p q = max' p q (ℚ-trichotomous fe p q)

min' : (x y : ℚ) → x < y ∔ (x ≡ y) ∔ y < x → ℚ
min' x y (inl _) = x
min' x y (inr _) = y

min : ℚ → ℚ → ℚ
min p q = min' p q (ℚ-trichotomous fe p q)
```

These auxiliary functions complicate proofs of properties of min and max. For example, to prove that max is reflexive for rational numbers, we first have to prove it for the auxiliary function, and then use a proof that max' is the same as max.

```
max'-to-max : (x y : ℚ) → (t : x < y ∔ (x ≡ y) ∔ y < x) → max' x y t ≡ max x y
max'-to-max x y t = equality-cases t I II
  where
    I : (l : x < y) → t ≡ inl l → max' x y t ≡ max x y
    I l e = max' x y t                          ≡⟨ ap (max' x y) e ⟩
            max' x y (inl l)                    ≡⟨ ap (max' x y) (ℚ-trich-a fe x y l ⁻¹) ⟩
            max' x y (ℚ-trichotomous fe x y)    ≡⟨ by-definition    ⟩
            max x y                             ∎
    II : (l : (x ≡ y) ∔ y < x) → t ≡ inr l → max' x y t ≡ max x y
    II r e = max' x y t       ≡⟨ ap (max' x y) e                  ⟩
             max' x y (inr r) ≡⟨ ap (max' x y) (ℚ-trich-b fe x y r ⁻¹) ⟩
             max' x y (ℚ-trichotomous fe x y) ≡⟨ by-definition ⟩
             max x y          ∎

max'-refl : (q : ℚ) → (t : q < q ∔ (q ≡ q) ∔ q < q) → max' q q t ≡ q
max'-refl q (inl l) = 𝟘-elim (ℚ<-not-itself q l)
max'-refl q (inr (inl l)) = l
max'-refl q (inr (inr l)) = 𝟘-elim (ℚ<-not-itself q l)

max-refl : (q : ℚ) → max q q ≡ q
max-refl q = I (ℚ-trichotomous fe q q)
  where
    I : q < q ∔ (q ≡ q) ∔ q < q → max q q ≡ q
    I t = max q q ≡⟨ max'-to-max q q t ⁻¹ ⟩
          max' q q t ≡⟨ max'-refl q t ⟩
          q          ∎
```

This may have been avoided by instead defining max for integers, but then it would have been necessary to prove the properties we want for max on the integers first. Regardless, it was an opportunity to learn how to deal with a problem which arises in proofs developed in type theory, which may not occur in classical mathematics.

I wrote more necessary proofs for min and max, listed now without proof.

```
max-comm : (p q : ℚ) → max p q ≡ max q p
max-to-≤ : (p q : ℚ) → p ≤ q × (max p q ≡ q) ∔ q ≤ p × (max p q ≡ p)
min-refl : (q : ℚ) → min q q ≡ q
min-comm : (p q : ℚ) → min p q ≡ min q p
min-to-≤ : (p q : ℚ) → p ≤ q × (min p q ≡ p) ∔ q ≤ p × (min p q ≡ q)
≤-to-min : (x y : ℚ) → x ≤ y → x ≡ min x y
<-to-min : (x y : ℚ) → x < y → x ≡ min x y
≤-to-max : (x y : ℚ) → x ≤ y → y ≡ max x y
<-to-max : (x y : ℚ) → x < y → y ≡ max x y
min-assoc : (x y z : ℚ) → min (min x y) z ≡ min x (min y z)
min-to-max : (x y : ℚ) → min x y ≡ x → max x y ≡ y
max-assoc : (x y z : ℚ) → max (max x y) z ≡ max x (max y z)
```

It is left to prove that the reals are a metric space.

```
ℝ-metric-space : metric-space ℝ
ℝ-metric-space = B-ℝ , ℝ-m1a , ℝ-m1b , ℝ-m2 , ℝ-m3 , ℝ-m4
```

Constructing this proof was the most time consuming portion of the project. The proofs of the properties are difficult to format for this report, and still require some cleanup for readability. The proofs for m1a and m4 are egregiously long, and difficult to follow by looking at the Agda code alone.

The m2 and m3 conditions are easily proved using order transitivity, and commutativity of min and max. Condition m1a is more difficult to prove, and requires a lot of algebraic manipulation. It required me to prove that two reals are equal. Reals are defined in terms of subsets of rational numbers, so we need to determine when subsets of types are equal. This requires the use of propositional extensionality.

Given two propositions of a type, we can prove they are equal if we can establish an "if and only if" relationship between inhabitants of the propositions. The Dedekind reals are defined in terms of two subsets of rationals; the lower cut and the upper cut. Discussion within my supervisor resulted in a proof that allows me to establish equality of a Real, given that only the lower subsets of each are equal. This is useful machinery to avoid duplicate code, since usually the proofs for the upper and lower cuts are symmetric.

```
ℝ-equality : (((Lx , Rx) , isCutx) ((Ly , Ry) , isCuty) : ℝ)
           → (Lx ≡ Ly)
           → (Rx ≡ Ry)
           → ((Lx , Rx) , isCutx) ≡ ((Ly , Ry) , isCuty)
```

```
ℝ-left-cut-equal-gives-right-cut-equal : (((Lx , Rx) , _) ((Ly , Ry) , _) : ℝ)
                                       → Lx ≡ Ly
                                       → Rx ≡ Ry
```

```
ℝ-equality-from-left-cut : (((Lx , Rx) , isCutx) ((Ly , Ry) , isCuty) : ℝ)
                         → Lx ≡ Ly
                         → ((Lx , Rx) , isCutx) ≡ ((Ly , Ry) , isCuty)
```

```
ℝ-equality-from-left-cut' : (((Lx , Rx) , isCutx) ((Ly , Ry) , isCuty) : ℝ)
                          → Lx ⊆ Ly
                          → Ly ⊆ Lx
                          → ((Lx , Rx) , isCutx) ≡ ((Ly , Ry) , isCuty)
```

These proofs allowed me to complete the proof of m1a.

Condition m4 was not an easy proof. I spent a long time trying to come up with an *efficient* proof, since otherwise we have an explosion of cases, due to the use of min and max in the metric

for Dedekind Reals. I managed to complete the proof, but it is still very large. I suspect that it is possible to reduce the proof to a few lines, but in the interest of moving forward with the project I resorted to considering a lot of cases and directly proving each case. This results in a proof which is extremely difficult to read and follow, but since the proof type-checks it is certainly *correct*.

It was surprisingly easy to see the proof of m1b. For any real x, and any positive ϵ, we need to find a pair of rationals a and b with $a < x < b$, and $|b - a| < \epsilon$. This is simply arithmetic located of the reals, the proof of which I had abandoned earlier.

```
ℝ-m1b : m1b ℝ B-ℝ
ℝ-m1b x ε l = ∥∥-functor I (ℝ-arithmetically-located x ε l)
  where
    I : Σ (a , b) : ℚ × ℚ , (a < x)
                          × (x < b)
                          × (0ℚ < (b - a))
                          × ((b - a) < ε)
      → Σ (p , q , u , v) : ℚ × ℚ × ℚ × ℚ , (p < x)
                                          × (u < x)
                                          × (x < q)
                                          × (x < v)
                                          × B-ℚ (min p u) (max q v) ε l
    I ((a , b) , a<x , x<b , l₁ , l₂)
      = (a , b , a , b) , a<x , a<x , x<b , x<b , iv iii
        where
          i : ℚ-metric b a < ε
          i = pos-abs-no-increase fe (b - a) ε (l₁ , l₂)
          ii : ℚ-metric b a ≡ ℚ-metric a b
          ii = ℚ-metric-commutes b a
          iii : ℚ-metric a b < ε
          iii = transport (_< ε) ii i
          iv : B-ℚ a b ε l → B-ℚ (min a a) (max b b) ε l
          iv = transport₂ (λ α β → B-ℚ α β ε l) (min-refl a ⁻¹) (max-refl b ⁻¹)
```

Working towards the goal of proving the continuous extension theorem, we run into the problem of arithmetic locatedness, so it is time to finally prove it.

4.2.1 Arithmetic Locatedness of Reals

Recall the definition of arithmetic locatedness.

```
ℝ-arithmetically-located : (z : ℝ)
                         → (p : ℚ)
                         → 0ℚ < p
                         → ∃ (x , y) : ℚ × ℚ , (x < z) × (z < y) × 0ℚ < (y - x) × (y - x) < p
```

Since x is inhabited, we begin with two rationals either side of x which can be arbitrarily far apart. The question is how do we get arbitrarily close to x, given rationals p and q, with $p < x < q$? If p or q were guaranteed to be close to x, this wouldn't be an issue since we could simply choose p and $p + \epsilon$.

A naive approach would choose one side, and move closer repeatedly using roundedness of x, but if q is greater than a distance ϵ away from x this is not fruitful. As such, we need to use an iterative process to move closer to x on each side. One such process is described by Bauer and Taylor [BT09, Proposition 11.15] uses approximate halves, along with a different, but equivalent, statement of the Archimedean principle. To avoid proving that the different variations of the Archimedean principle are equivalent, I instead chose to attempt the proof using trisections of the interval (p, q). This proof was suggested to me by de Jong, a PHD student at the University of Birmingham.

By trisecting the interval (p, q), we obtain rationals a, b with $p < a < b < q$, with each sub interval a third of the difference of the whole interval. By using locatedness of the interval, we see that $a < x$

or $x < b$. We now have a new interval, either (p, b) or (a, q), which is $\frac{2}{3}$ the size of the original interval. By repeatedly applying trisections, we will find an interval smaller than any arbitrary ϵ. This proof sketch relies on the following statement: given positive rationals a and ϵ, there exists a natural number N such that $(\frac{2}{3}^n * a) < \epsilon$ when $n > N$.

The statement clearly true, but fiendishly difficult to prove. My supervisor suggested formalising logarithms, but after some initial work I decided that it would be too difficult for me prove in this fashion. There were many problems that arise even in proof sketches, including the requirement to convert between natural number and rational logarithms, the different logarithmic bases involved, and showing that logarithms preserves order when converting between naturals and rationals. I temporarily abandoned the proof while I worked on Metric Spaces.

Upon reaching condition m1b, I returned to the problem. A mathematics lecturer conceived a solution using the binomial expansion of 2/3, avoiding the use of logarithms. This was a nice proof (as an exercise, try and prove the statement using binomial expansions!), but I still believed this proof would be too difficult to implement in Agda. I would need to formalise summations and manipulations of sums of rational number. After a lot of thought, I came up with a proof of my own, by using limits of sequences of Rational Numbers, by noticing that the proof we are trying to solve is very close the definition of a limit.

4.2.2 Limits of Rational Sequences

The limit is defined as in classical mathematics.

```
_limit-of_ : (L : ℚ) → (f : ℕ → ℚ) → 𝒰₀ ̇
L limit-of f = ∀ (ε : ℚ) → 0ℚ < ε
                         → Σ N : ℕ , ((n : ℕ) → N ≤ n → ℚ-metric (f n) L < ε)
```

If we can prove that a rational sequence defined by $f(n) := 2/3^n$ converges to 0, then we are done. We notice that the sequence $g(n) := \frac{1}{n+1}$ satisfies $f(n) < g(n)$ for $n > 0$.

```
⟨2/3⟩^_ : ℕ → ℚ
⟨2/3⟩^ 0        = toℚ (pos 1 , 0)
⟨2/3⟩^ (succ n) = rec 2/3 (_* 2/3) n


⟨1/n⟩ : ℕ → ℚ
⟨1/n⟩ n = toℚ (pos 1 , n)

⟨1/sn⟩ : ℕ → ℚ
⟨1/sn⟩ 0 = 1ℚ
⟨1/sn⟩ (succ n) = ⟨1/n⟩ n
```

If we can prove this, then by also proving the sandwich theorem by for sequences of rational numbers, we can prove that $f(n)$ converges to 0.

```
sandwich-theorem : (L : ℚ)
                 → (f g h : ℕ → ℚ)
                 → (Σ k : ℕ , ((k' : ℕ) → k ≤ k' → f k' ≤ g k' × g k' ≤ h k'))
                 → L limit-of f
                 → L limit-of h
                 → L limit-of g
```

There were several intermediate results needed. Trivially, the constant sequence which maps natural numbers to the rational number 0 converges to 0.

```
0f : ℕ → ℚ
0f _ = 0ℚ
```

```
0f-converges : 0ℚ limit-of 0f
0f-converges ϵ l = 0 , f-conv
  where
    f-conv : (n : ℕ) → 0 ≤ n → ℚ-metric (0f n) 0ℚ < ϵ
    f-conv n less = transport (_< ϵ) I l
      where
        I : ℚ-metric (0f n) 0ℚ ≡ 0ℚ
        I = ℚ-metric (0f n) 0ℚ ≡⟨ by-definition ⟩
            abs (0ℚ - 0ℚ)      ≡⟨ by-definition ⟩
            abs 0ℚ             ≡⟨ by-definition ⟩
            0ℚ ∎
```

The proof that $g(n)$ converges was very difficult, and required a lot of algebraic manipulation, and a lemma that reciprocating two fractions flips the order.

```
⟨1/sn⟩-converges : 0ℚ limit-of ⟨1/sn⟩
```

Now, it is required to prove that this sequence is indeed sandwiched by the constant 0 sequence and $\frac{1}{n+1}$.

```
⟨2/3⟩^n-squeezed : Σ N : ℕ , ((n : ℕ) → N ≤ n → (0f n ≤ (⟨2/3⟩^ n)) × (⟨2/3⟩^ n) ≤ ⟨1/sn⟩ n)
```

A problem arose during the proof of this function. For anything more than very simple inequalities of rational numbers, trying to compile the code causes the Agda compiler to hang indefinitely. With thanks to de Jong, the solution is to put any offending inequalities into an abstract sub-proof. This prevents Agda from unfolding the proof, which speeds up type-checking. While I haven't identified the exact cause, I suspect the indefinite hang is due to the size of terms involved for my representation of rationals, since they contain the proof that the fraction is in lowest terms.

This problem doesn't arise with only inequalities, addition of rationals can also result in indefinite hanging of the compiler. This fix suffices for the RationalsLimits file to compile, but didn't work for other indefinite hangs encountered. There is a command line option when compiling Agda files: –experimental-lossy-unification [Unka]. The abstraction of certain proofs, and the above command line options have solved all the cases where the compiler has struggled to compile my code. I worry that both options may fail for more complex files as the project develops in the future, and if this is the case I will have to consider refactoring some of my files to avoid huge terms.

Unfortunately, as the documentation describes, the command line option is sound, but not complete. When this option in used, there are some expressions that no longer type-check that normally would, however for my purposes it has allowed to compile problematic files, such as MetricSpaceDedekindReals. It is important to note that it doesn't allow anything to compile that shouldn't. I view the necessity of the use of this option a limitation of Agda, and with more development on the language it may be not necessary to use the option in the future.

With the proof that $\frac{2}{3}^n$ is sandwiched by sequences that converge to 0, we can conclude that it also converges to 0.

```
⟨2/3⟩^n-converges : 0ℚ limit-of ⟨2/3⟩^_
⟨2/3⟩^n-converges = sandwich-theorem 0ℚ 0f ⟨2/3⟩^_ ⟨1/sn⟩ ⟨2/3⟩^n-squeezed 0f-converges ⟨1/sn⟩-converges
```

I can now finish the proof that the Dedekind reals are arithmetically located. I can locate any real to any degree of error, bounding it below and above by rational approximations of the real, and hence conclude the proof that the Dedekind reals are a metric space. Proving this result was a highlight of the project.

4.2.3 Metric Space Completeness

A complete metric space is a metric space in which every Cauchy sequence converges. The definition is encapsulated by the following types.

```
convergent-sequence : {𝒰 : Universe} → (X : 𝒰 ̇) → metric-space X → (S : ℕ → X) → 𝒰 ̇
convergent-sequence X (B , _) S
  = ∃ x ꞉ X , (((ϵ , l) : ℚ₊) → Σ N ꞉ ℕ , ((n : ℕ) → N < n → B x (S n) ϵ l))

cauchy-sequence : {𝒰 : Universe} → (X : 𝒰 ̇) → metric-space X → (S : ℕ → X) → 𝒰₀ ̇
cauchy-sequence X (B , _) S
  = ((ϵ , l) : ℚ₊) → Σ N ꞉ ℕ , ((m n : ℕ) → N ≤ m → N ≤ n → B (S m) (S n) ϵ l)


cauchy→convergent : {𝒰 : Universe} → (X : 𝒰 ̇) → metric-space X → (S : ℕ → X) → 𝒰 ̇
cauchy→convergent X m S = cauchy-sequence X m S → convergent-sequence X m S

complete-metric-space : {𝒰 : Universe} → (X : 𝒰 ̇) → 𝒰₁ ⊔ 𝒰 ̇
complete-metric-space X = Σ m ꞉ (metric-space X) , ((S : ℕ → X) → cauchy→convergent X m S)
```

For the classical version of the CET, it is required that the target domain is a complete metric space. Assuming that I would need to have this assumption to prove the CET for rationals to Dedekind reals, I proved that the Dedekind reals are a complete metric space with respect to our chosen metric. I use the approach of the HoTT book, which first defines a Cauchy approximation sequence, and provides a partial proof that every Cauchy approximation sequence has a limit [Uni13, Theorem 11.2.12].

The left and right cuts of the limit of a Cauchy sequence f are defined in the following way.

```
LeftCondition : (q : ℚ) → 𝒰₀ ̇
LeftCondition q = ∃ ((ϵ , l₁) , (θ , l₂)) ꞉ ℚ₊ × ℚ₊ , in-lower-cut (q + ϵ + θ) (f (ϵ , l₁))

RightCondition : (q : ℚ) → 𝒰₀ ̇
RightCondition q = ∃ ((ϵ , l₁) , (θ , l₂)) ꞉ ℚ₊ × ℚ₊ , in-upper-cut (q - ϵ - θ) (f (ϵ , l₁))
```

We must not only prove that this is the limit of the Cauchy approximation sequence, but prove that this y satisfies the four properties of a Dedekind cut. This was an extremely challenging proof. The partial proof in the HoTT book states that "it is clear that L_y and U_y are inhabited, rounded and disjoint". While it may have been clear the author, it wasn't so clear to me, and I couldn't find any reference to a complete version of this proof. I devised a proof on my own for roundedness, and owe thanks to Ambridge for help with the proof of inhabitedness, and de Jong for help with the proof of disjointness. Locatedness could be adapted from the HoTT book For disjointness, it was helpful to prove a lemma, proving the equivalence of the two versions of disjointness in the definition of a Dedekind real in the presence of locatedness.

```
trans→disjoint : (L R : 𝒫 ℚ) → disjoint L R → (q : ℚ) → ¬ (q ∈ L × q ∈ R)
disjoint→trans : (L R : 𝒫 ℚ) → located L R → ((q : ℚ) → ¬ (q ∈ L × q ∈ R)) → disjoint L R
```

With the proof that the defined limit is a real number, it remains to show that it is indeed the limit of the Cauchy approximation sequence. This proof is given [Uni13], although due to my alternate definition of a metric space, I struggled to reformulate the proof for my own work. By slowly working through the proof on paper, I eventually managed to adapt it for my library.

By proving that every Cauchy sequence has a modulus of convergence, and that every Cauchy sequence satisfies some Cauchy approximation sequence (with help from my supervisor), I completed the proof that the Reals are a complete metric space.

```
ℝ-complete-metric-space : complete-metric-space ℝ
ℝ-complete-metric-space = ℝ-metric-space , ℝ-cauchy-sequences-are-convergent
```

4.3 Continuity

We now need to define continuity of functions between metric spaces.

```
continuous : {M₁ : 𝒰 ̇} {M₂ : 𝒱 ̇}
           → (m₁ : metric-space M₁)
           → (m₂ : metric-space M₂)
           → (f : M₁ → M₂)
           → 𝒰 ̇
continuous {𝒰} {𝒱} {M₁} {M₂} (B₁ , _) (B₂ , _) f =
  (c : M₁) → ((ϵ , l) : ℚ₊) → Σ (δ , l₂) : ℚ₊ , ((x : M₁) → B₁ c x δ l₂ → B₂ (f c) (f x) ϵ l)
```

With all of these concepts defined, it should now be possible to prove the continuous extension theorem, or at least a constructive adaptation. To investigate the intuition behind the proof, I proved that the identity function is continuous, that the composition of continuous functions preserves continuity, and concluded that the embedding of the rationals in the reals composed with the identity function is continuous. This is a trivial example, but serves as a proof of concept as to how extensions are formulated.

```
composition-preserves-continuity : {M₁ : 𝒰 ̇} {M₂ : 𝒱 ̇} {M : 𝒲 ̇}
                                 → (m₁ : metric-space M₁)
                                 → (m₂ : metric-space M₂)
                                 → (m : metric-space M)
                                 → (f : M₁ → M₂)
                                 → (g : M₂ → M)
                                 → continuous m₁ m₂ f
                                 → continuous m₂ m g
                                 → continuous m₁ m (g ∘ f)

id-continuous : continuous ℚ-metric-space ℚ-metric-space id
id-continuous c (ϵ , 0<ϵ) = (ϵ , 0<ϵ) , λ _ B → B

ℚ-ℝ-id-continuous : continuous ℚ-metric-space ℝ-metric-space (ι ∘ id)
ℚ-ℝ-id-continuous = composition-preserves-continuity ℚ-metric-space ℚ-metric-space ℝ-metric-space id ι id-continuous ι-co
```

Following the idea of the monotonic extension theorem, I formulated the cut conditions that I believe the continuous extension function should have. Unfortunately, I did not have time to prove the theorem. It may not be possible to prove the theorem with this definition of continuity, so it remains to investigate this proof, and whether suitable alternatives may be Bishop continuity or uniform continuity if there is no meaningful progress with standard continuity.

4.4 Direct Operations

During the investigation of theorem above, I proved arithmetic locatedness of real numbers. Using this, I was able to finish the direct implementation of addition which I abandoned earlier in the project. I was also able to define negation of reals, and prove that the reals are a group with respect to addition.

```
ℝ+-comm : ∀ x y → x + y ≡ y + x
ℝ-zero-left-neutral : (x : ℝ) → 0ℝ + x ≡ x
ℝ-zero-right-neutral : (x : ℝ) → x + 0ℝ ≡ x
ℝ+-assoc : ∀ x y z → x + y + z ≡ x + (y + z)
ℝ+-inverse-exists : ∀ x → Σ x' : ℝ , x + x' ≡ 0ℝ
ℝ+-inverse-exists' : (x : ℝ) → Σ x' : ℝ , (x' + x ≡ 0ℝ) × (x + x' ≡ 0ℝ)

ℝ-additive-group : Group-structure ℝ
ℝ-additive-group = _+_ , (ℝ-is-set , ℝ+-assoc , 0ℝ , ℝ-zero-left-neutral , ℝ-zero-right-neutral , ℝ+-inverse-exists
```

This is a direct implementation, and doesn't provide a framework to implement other real operators or functions. It should be possible to define and implement multiplication the same manner, with a view to proving that the reals form a complete ordered field, however the continuous extension theorem will be the focus in future extensions of this project. I did not have time to formalise any further results in this project. The following sections discuss progress, and the different avenues for extending this project.

5 Discussion

5.1 Progress

Over the course of this project, I have formalised a large amount of work on this constructive library for Dedekind reals.

- Rationals are a field
- Min, max, absolute values of rationals and properties,
- Defined metric spaces, and Cauchy and convergent sequences,
- Rationals are a metric space,
- Limits of rational sequences, convergence of $\frac{2}{3}^n$ and $\frac{1}{n}$,
- Sandwich theorem for rational sequences,
- Extension of monotonic rational functions, extension of successor function as proof of concept,
- Equality of reals (for use in proofs about reals),
- Dedekind reals order, and properties,
- Dedekind reals are arithmetically located,
- Dedekind reals are metric space,
- Cauchy approximation sequences and their limit,
- Modulus of convergence of Cauchy sequences,
- Cauchy sequences satisfy Cauchy approximation sequence condition,
- Real Cauchy sequences converges, hence Cauchy completeness of real metric space,
- Continuity of functions between metric spaces,
- Composition of continuous functions is continuous,
- Embedding from rationals to reals is continuous,
- Dedekind reals addition, negation,
- Dedekind reals are a group with respect to addition.

Due to the complexity of the proofs towards the end of this project, I did not achieve the ambition of proving that the reals form a complete ordered Archimedean field. Regardless, I am proud of the progress I have made thus far, and will continue the work outside of the project to achieve this goal. Two proofs that I am particularly proud of are the proof that the Reals are arithmetically located, and the prove that Cauchy approximation sequences converge. Both of these proofs required a large amount of research, pen and paper attempts and resilience.

5.2 Evaluation

Due to the nature of computer assisted proof verification, I can guarantee the correctness of every proof I have produced as a part of this project, up to the correctness of the types and our interpretation of the types. In terms of formalisation, at least up the progress in the project so far, I can justifiably claim that my proofs are correct, and of course this is one of the reasons that computer verified formalisation of mathematics is important. Correctness is not the only measure of success. We can also look at both elegance and code quality.

5.2.1 Elegance

It is hard to quantify the elegance of a proof. There are usually many different ways to prove a mathematical statement, and the "correct" way to prove a statement is usually a matter of debate. I have no doubt that some of my proofs may be improved. For example, when proving arithmetic locatedness, I chose to use the argument using limits because I believed that this was the easiest approach. It may have been the case that logarithms or binomials result in more elegant proofs. In the long-term future, one can imagine that these concepts will be formalised in projects like mine, and it will be easier to try different approaches to solve problems.

I am happy with the level of elegance in this project, I think the ideas behind most of the proofs are intuitive. While the primary goal of the project is not pedagogical, I think is important that lessons can be learned from the internal code in the project.

5.2.2 Code Quality

I have made efforts throughout the project to produce high quality code. It is important for code to be understandable, particularly if the code is a mathematical proof. I haven't been fully successful in this regard. There are two contributing factors.

The first is that I was building on top of my work in my previous project, in which I was learning how to implement mathematical proofs in Agda, resulting in both inefficient proofs and general untidiness of the files.

The second is that the proofs towards the end of the project were generally very large, and usually required proving lemmas mid-proof. I found myself quickly proving lemmas, but then forgetting to clean them up after completing the main proof. I will list a number of qualities I believe the code should satisfy to improve readability.

- Names of proofs, variables, sub-proofs should be consistent, at least within each file.
- If the code is not entirely self-documenting, an accompanying description in mathematical prose should be provided, indicating the ideas behind the proof, if not the full proof.
- Duplicate code should be avoided, common algebraic manipulations should be separated outside of main proofs to reduce the amount of equational reasoning, and improve readibility.
- Where suitable, pictorial descriptions of proofs should be provided, as visual aid are important to help with understanding
- The code should have consistent formatting.

I am not satisfied with the current state of some of my proofs. For example, Cauchy completeness of the real metric space, while "finished", needs some work to improve the quality of the code. My intention was rewrite some of my files to improve the code quality once I had proved that the Reals are a complete Archimedean ordered field. Due to time constraints, I did not managed to complete either of these intentions. I struggled to justify improving existing proofs as opposed to constructing new proofs as I worked towards the final goal.

I have learned that I should be extremely diligent in this regard. I should ensure that after the completion of any proof, I check that the proof is in an acceptable state, and provide a mathematical description of the proof and the idea behind it.

5.3 Prioritisation

I have had a tendency to try and work on difficult problems for too long. I spent a few weeks early in the project trying to prove the arithmetic locatedness of the reals. This was not a fruitful endeavor and I abandoned the attempt. It would have been more productive to focus my work elsewhere sooner.

There is a critical point where I should switch to working on a different problem. Whenever I began working on a different problem throughout this project, I would make progress. I had to reconsider

arithmetic locatedness later in the project, and with more experience working with the reals I was able to find a solution.

This is a difficult balance. Working on proofs tends to be a sequential process, so working on something else usually means assuming that what you are currently working on is correct. It is entirely feasible to postulate a proof (by leaving it as a hole in Agda), and then find that it is impossible to prove the postulate. As a result, I have been reluctant to jump ahead of difficult proofs. I have learned that I should more swiftly identify when I should stop working on a difficult problem, and work productively on a different problem. I should be more willing to make assumptions, which may have drawbacks but should be more useful in the long term.

5.4 Identifying Proofs

I have found that constructive analysis has been particularly challenging. There has been a wealth of work on constructive analysis (see [TD88] or [BB87]), but this work tends to be from a purely mathematical perspective, without references to Type Theory. As such, most of the proofs in my work have been constructed "in my own words", sometimes using *ideas* from proofs found in literature. An egregious example I mentioned earlier was in a proof in the HoTT book, which stated that "it is clear that L_y and U_y are inhabited, rounded and disjoint". Unfortunately, it was not so clear to me, and it took a few weeks for me to work through the proof, with help from Ambridge, and de Jong who suggested a strategy for proving the disjointedness of the limit of Cauchy sequences. It can be frustrating to not understand a proof, whilst knowing that it can be solved. This exemplifies why it is extremely important for proofs to be clean and comprehensive, because otherwise they are pedagogically useless.

On the other hand, it is extremely satisfying to write a proof which I have constructed on my own. I am very proud of my proof that there exists some n such that $(\frac{2}{3})^n < p$ for any positive p. As I mentioned previously, I had multiple suggestions, but eventually came up with my own strategy using the sandwich theorem. It is satisfying regardless to construct a proof based on ideas found in literature, but even more so when the proof is developed completely on my own, and prove that my idea for the proof is correct.

5.5 Minimal Hypotheses

Mathematically speaking, a theorem is stronger when it requires fewer assumptions, because it is then applicable to more situations. As such, a "perfect" development would prioritise assuming the minimal hypothesis for each and every proof.

Consider now a proof which I updated over the course of this new project. Originally, I formulated and proved it without the use of function extensionality. I reformulated this proof during the course of this project, when I redefined order of integers and updated my proofs of order of rationals.

It is appreciable not only how the readability increases, but how it is much easier to understand how the proof works, and the improved conciseness. The drawback is that in order to write the proof in the above way, I am required to assume function extensionality. In the first variation, there is a large amount of algebraic manipulation of integers, without a clear reason why this is necessary. In the updated proof it is clear exactly what the chosen point is, and how it is proved that it is between the two arbitrary rationals.

Previously, I believed that the development should be "pure", and that every proof should only assume the absolute minimum necessary to complete the proof, but after discovering situations such as the above, I find myself leaning towards the side of elegance.

For the purposes of this project, it is not so important that my specific implementation of certain proofs involving rationals assumes function extensionality. Firstly, the rationals were built with the intention of being used for the Dedekind reals, which requires liberal use of function extensionality, and is globally assumed in files which use the Reals. Secondly, is it unlikely that these rationals will be used outside the context of the Dedekind reals. If I needed another version which explicitly excluded function extensionality, I could trivially formalise them.

6 Future Work

The initial extension is to finish the proof that the Dedekind reals are a complete Archimedean ordered field. This will serve as a basis for a large number of applications. The short term plan follows.

- Prove the continuous extension theorem for metric spaces. It would be very useful to prove this directly, as it allows us to extend any continuous rational function to reals, and it is known that computable functions are continuous [htta]. Failing this, we try and prove that Bishop continuous rational functions can be extended, which should be easier, but is less general.
- If all attempts to prove extension theorems fail, then instead we can directly implement multiplication of reals. It is known to be egregious, and may be lengthy but should be possible to prove.
- Conclude by proving the field axioms for Dedekind reals, and prove that the reals are Dedekind complete.

With this formalisation in place, we look to the applications of the library.

6.1 Exact Computation

The thesis of Booij [Boo21] describes how the reals may be equipped with locators to observe information. It is shown that "we can compute arbitrarily precise approximations to reals with locator", but stops short of doing so. Hence, with this development of the reals, by equipping them with locators we can work on providing verified exact computation with real numbers.

Due to various choices of representation, my code does not have any claim to efficiency of computation. I have chosen representations of numbers that are conducive to writing *proofs* efficiently. As an explicit example, there would be an added layer of difficulty to use a binary representation of natural numbers to formalise the reals. Using binary induction is possible, but adds complexity to proofs that are much simpler using natural induction.

For the purpose of exact computation it would be prudent to have isomorphic representations which are more suitable for the purposes of efficient computation. A nice extension of this project would be to write these efficient representations, showing that they are isomorphic to my definitions, and provide an interface for exact computation of real numbers.

6.2 HoTT Book

Having used the HoTT Book as guidance for some of the proofs in this development, it would be useful to continue the formalisation that follows in the book. In particular, I am interested in formalising the Cauchy reals, and show that they can be embedded in the Dedekind reals. I am also interested in the role of the Dedekind reals in homotopy type theory. One investigation is into the link between elements of synthetic homotopy theory developed using the Dedekind reals, and constructive analysis.

6.3 Formalisation

Formalisation is an important part of modern mathematics. Verification of proofs provides guarantees of correctness, and in particular proofs developed in a constructive setting allows us to extract useful programs. The reals as a foundations are conducive to extensions in many studies in mathematics. I am personally interested in formalising concepts in real and complex analysis, in numerical analysis, and the topology of real numbers.

7 Conclusion

The initial goal of this project was to formalise the complete ordered Archimedean field in Agda. After direct attempts at defining operators for reals, it became clear that formalisation involving reals

is laborious. As such, I searched to find an approach which avoided direct implementations of reals operations.

The continuous extension theorem was identified as a candidate alternative. Extension theorems have been widely studied, but there seems to be a lack of constructive proofs in literature. Hence, I formalised metrics spaces, using an alternative definition which avoids the use of reals, and proved that the Dedekind reals and rationals are metric spaces. This required formalising more operators of rationals and their properties, including the min, max and absolute value operations. It also required the challenging proof that the reals are arithmetically located, which I worked on over the course of the project, and finally proved using the sandwich theorem for sequences of rationals numbers.

I proved that the reals are a complete metric space, which was the second most challenging proof in this project. In particular, proving that the limit of a Cauchy sequence satisfies disjointedness took weeks of thought, and assistance from my supervisor, Ambridge and de Jong, even though it is claimed that is "is clear" in literature.

I have developed a deeper appreciation of the importance of clean code. While it may be true that the proofs in this project *correct*, it doesn't serve any pedagogical purpose if it is difficult to understand the idea behind the proof. I believe the cleanliness of code should be on the same pedestal as correctness. If the code is not self-documenting, then a description in mathematical prose should be provided. At various points in the project I became frustrated because I couldn't understand a proof in literature (or the proof wasn't given), and there is no excuse to pass on this frustration to readers of my own code.

I have appreciated experimenting with my work flow. I have tried to work on different proofs when I am finding a particular one too difficult. I realise that I need to make concerted efforts to acknowledge more precisely the best time to work on a different problem, to be more productive with my time. Unfortunately, I fell into this trap towards the end of the project, working on the continuous extension theorem for too long, where working on a different problem might have allowed for further progression.

I discussed the concept of minimal hypothesis, and how I let go the notion that my work should be "mathematically pure" in favour of elegance and readability of proofs.

I have discussed the potential avenues for extensions of this project. Exact computation is an exciting extension, following the work of Booij. Formalisation of the real number section in the HoTT book is an interest of mine, as well as general applications of real numbers in real and complex analysis, and topology.

I would like to thank Escardo, Ambridge and de Jong, for helping to me learn and understand the ideas behind many of the proofs in this project, and helping to sketch pen and paper proofs for me to implement in Agda.

References

[22] *Archimedean property*. en. Page Version ID: 1073144725. Feb. 2022. URL: https://en.wikipedia.org/w/index.php?title=Archimedean_property&oldid=1073144725 (visited on 03/06/2022).

[Att20] Mark van Atten. *Luitzen Egbertus Jan Brouwer*. Feb. 2020. URL: https://plato.stanford.edu/entries/brouwer/.

[BB87] Errett Bishop and Douglas Bridges. "Constructive Analysis". In: *Journal of Symbolic Logic* 52.4 (1987), pp. 1047–1048. DOI: 10.2307/2273838.

[Bel21] John L. Bell. *The axiom of choice*. Dec. 2021. URL: https://plato.stanford.edu/entries/axiom-choice/.

[Boo21] Auke B. Booij. "Extensional constructive real analysis via locators". In: *Mathematical Structures in Computer Science* 31.1 (2021), pp. 64–88. DOI: 10.1017/S0960129520000171.

[BT09] ANDREJ BAUER and PAUL TAYLOR. "The Dedekind reals in abstract Stone duality". In: *Mathematical Structures in Computer Science* 19.4 (2009), pp. 757–838. DOI: 10.1017/S0960129509007695.

[Coq18] Thierry Coquand. "Type Theory". In: *The Stanford Encyclopedia of Philosophy*. Ed. by Edward N. Zalta. Fall 2018. Metaphysics Research Lab, Stanford University, 2018. URL: https://plato.stanford.edu/archives/fall2018/entriesype-theory/ (visited on 03/02/2022).

[Dow97] Mark Dowson. "The Ariane 5 Software Failure". In: *SIGSOFT Softw. Eng. Notes* 22.2 (Mar. 1997), p. 84. ISSN: 0163-5948. DOI: 10.1145/251880.251992. URL: https://doi.org/10.1145/251880.251992.

[DP20] Peter Dybjer and Erik Palmgren. "Intuitionistic Type Theory". In: *The Stanford Encyclopedia of Philosophy*. Ed. by Edward N. Zalta. Summer 2020. Metaphysics Research Lab, Stanford University, 2020. URL: https://plato.stanford.edu/archives/sum2020/entriesype-theory-intuitionistic/ (visited on 03/02/2022).

[Esc19] Martín Hötzel Escardó. *Various new theorems in constructive univalent mathematics written in Agda*. https://github.com/martinescardo/TypeTopology/. Agda development. Feb. 2019.

[Gra18] Daniel Grayson. "An introduction to univalent foundations for mathematicians". en. In: *Bulletin of the American Mathematical Society* 55.4 (Oct. 2018), pp. 427–450. ISSN: 0273-0979, 1088-9485. DOI: 10.1090/bull/1616. URL: https://www.ams.org/bull/2018-55-04/S0273-0979-2018-01616-9/ (visited on 03/02/2022).

[How80] William Alvin Howard. "The Formulae-as-Types Notion of Construction". In: *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus, and Formalism*. Ed. by Haskell Curry et al. Academic Press, 1980.

[htta] Andrej Bauer (https://cs.stackexchange.com/users/1329/andrej-bauer). *Why are computable functions continuous?* Computer Science Stack Exchange. URL:https://cs.stackexchange.com/q/81018 (version: 2017-09-08). eprint: https://cs.stackexchange.com/q/81018. URL: https://cs.stackexchange.com/q/81018.

[httb] KConrad (https://mathoverflow.net/users/3272/kconrad). *Widely accepted mathematical results that were later shown to be wrong?* MathOverflow. URL:https://mathoverflow.net/q/35558 (version: 2021-02-15). eprint: https://mathoverflow.net/q/35558. URL: https://mathoverflow.net/q/35558.

[LQ15] Henri Lombardi and Claude Quitté. "Commutative Algebra: Constructive Methods". In: *Algebra and Applications* (2015). ISSN: 2192-2950. DOI: 10.1007/978-94-017-9944-7. URL: http://dx.doi.org/10.1007/978-94-017-9944-7.

[Off] U. S. Government Accountability Office. *Patriot Missile Defense: Software Problem Led to System Failure at Dhahran, Saudi Arabia*. en. URL: https://www.gao.gov/products/imtec-92-26 (visited on 03/16/2022).

[Sim83] G.F. Simmons. *Introduction to Topology and Modern Analysis*. International series in pure and applied mathematics. R.E. Krieger Publishing Company, 1983. ISBN: 9780898745511. URL: https://books.google.co.uk/books?id=tEFAAQAAIAAJ.

[Ste99] D.E. Stevenson. "A critical look at quality in large-scale simulations". In: *Computing in Science Engineering* 1.3 (1999), pp. 53–63. DOI: 10.1109/5992.764216.

[TD88] A.S. Troelstra and D. van Dalen. *Constructivism in Mathematics, Vol 1*. ISSN. Elsevier Science, 1988. ISBN: 9780080570884. URL: https://books.google.co.uk/books?id=-tc2qp0-2bsC.

[Tro73] A. S. Troelstra. *Metamathematical Investigation of Intuitionistic Arithmetic and Analysis*. New York: Springer, 1973.

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

[Unka] Unknown. *Lossy Unification — Agda 2.6.3 documentation*. URL: https://agda.readthedocs.io/en/latest/language/lossy-unification.html?highlight=experimental-lossy-unification (visited on 04/02/2022).

[Unkb] Unknown. *What is Agda? — Agda 2.6.3 documentation*. URL: https://agda.readthedocs.io/en/latest/getting-started/what-is-agda.html (visited on 03/27/2022).