

First Year PhD Annual Report

LI Nuo

University of Nottingham

Table of Contents

1	Introduction.....	3
1.1	Quotient Set.....	3
1.2	Type Theory	4
1.3	Quotient Types	5
1.4	The relation between equality and quotient types.....	5
1.5	Literature Reviews	6
2	Aims and Objectives of the Project	7
3	Theoretical Methods	8
4	Results and Discussion	8
4.1	Definitions	8
5	Examples	13
5.1	Integers	13
5.2	Rational numbers	14
5.3	Real numbers as cauchy sequences.....	15
6	Conclusion	15

Abstract. TO DO: enhance the connection within between ideas. Split the two ideas in the abstract: why in general quotient is useful and implementing in Agda. Given a setoid, that is a set equipped with an equivalence relation, one can form its quotient set, that is the set of equivalence classes. Reinterpreting this idea in type theory, quotient type is formed by some type with its equivalence relation. However in intensional Type Theory the quotient type is still unavailable. The axiomatic quotient type enables us to define real numbers, functionals, and many other sets which are not definable in current intensional Type Theory, such as Agda which is also a theorem prover. Also we can benefit from the formalisation of some quotient types, since some base types are simpler to deal with or have better features, such as the integers represented by a pair of natural numbers. Therefore, we undertake this project to investigate how to implement quotients in Agda. There are several schemes to do this, extending intensional Type Theory to quotient types, or we can define quotients without in current settings. In [2], we present some results of defining quotients in Agda and I will give some explanations and show some instances in this report as a complement to it.

1 Introduction

1.1 Quotient Set

In mathematics, a quotient represents the result of division. Usually, the division is between numbers and the result is also a number. However the concept of division can be extended to other domains. In set theory, addition and multiplication can be extended to sets. Then the addition and multiplication can be seen as the calculation on the cardinality when operating the disjoint union and product within a special class of sets, which are called set-theoretical natural numbers.

$$\begin{aligned}\underline{0} &= \emptyset \\ \underline{1} &= \{\underline{0}\} \\ \underline{2} &= \{\underline{0}, \underline{1}\} \\ &\vdots \\ \underline{n} &= \{\underline{0}, \underline{1}, \underline{2}, \dots, \underline{n-1}\}\end{aligned}$$

Similarly, we can also extend division to sets. First we need a setoid,

Definition 1. A setoid (A, \sim) is a set A equipped with an equivalence relation $\sim : A \rightarrow A \rightarrow \mathbf{Prop}$.

Definition 2. Given a setoid, we can get an equivalent class for each $a : A$,

$$[a] = \{b \in A \mid b \sim a\}$$

Definition 3. Given a setoid (A, \sim) , a quotient set A / \sim is the set of equivalent classes of \sim ,

$$A / \sim = \{[a] \in \wp(A) \mid a \in A\}$$

Then dividing a natural number a , by another number b , can be inductively defined as dividing the set \underline{a} by the equivalence relation that the corresponding number has same quotient over b . The quotient set only contains the equivalence classes which has b elements, then the quotient number is just the cardinality of this set.

To make the difference between setoids and quotient sets clear, we can use an analogy, $8 \div 2 = 4$. We know 4 is the quotient because $4 \times 2 = 8$, and we cannot recover the dividend and the divisor from the quotient 4. Similarly, the setoids can be seen as a composition of the dividend and divisor, but the quotient sets do not contain all the information from this setoid, hence we can not recover the base sets. And a set can be the quotient type of several different setoids.

Many mathematical objects are quotients. Some are quite obvious, such as integers modulo some number and rational numbers as fractions. Some are also intrinsically quotients, such as integers, real numbers.

1.2 Type Theory

The theory of types was first introduced by Russell as an alternative to naive set theory. However, the type theory we discussed here is the one developed by Per Martin-Löf. It is based on the Curry-Howard isomorphism between propositions and the types of its proofs such that it can served as a formalisation of mathematics.

Type Theory can be used as a programming language in which the evaluation of well-typed program always terminates [14]. There are a few implementation of type theory, for example NuPRL, Coq and Agda. Agda is one of the most recent implementation and it has a bundle of good features which makes it a good theorem prover as well as a programming language. Hence we can program and make formal verification in the same language. We will use it to implement our definitions and verify theorems and propositions in this project.

Per Martin-Löf proposed both intensional Type Theory and extensional Type Theory. Usually the implementation of Type Theory can be categorised into intensional ones and extensional ones. The distinction between them is whether the definitional equality is distinguished with propositional equality. In intensional Type Theory, two objects are definitional equal if they are the same, but two objects are propositional equal if we have proof. There must be proof terms for propositional equalities. However in extensional Type Theory, they are not distinguished so that type checking is undecidable due to the halting problem, but the reinterpretation of quotients is easier. Nevertheless, intensional Type Theory is more useful since type checking terminates because it only depends on definitional equality which is decidable in intensional Type Theory [1]. Therefore we also want to implement quotients in intensional Type Theory.

1.3 Quotient Types

In Martin-Löf type theory, many notions from set theory and propositional logic can be implemented easily, such as the product of sets and disjoint union of sets. However the reinterpretation of quotients in Type Theory is still a big problem. I will discuss the different related work on this topic in the literature review.

Since types of Type Theory work similar to sets in set theory, and we can define equivalence relations on types, it is natural to define quotient types as follows,

Definition 4. *Given $A : \mathbf{Set}$, and an equivalence relation $\sim : A \rightarrow A \rightarrow \mathbf{Prop}$, a quotient type $Q : \mathbf{Set}$ represents the corresponding quotient set A / \sim .*

Quotient types can be used to implement the quotients in mathematics. Some other notions in type theory can also be formalised by quotient types. For example partiality monad divided by a weak similarity ignoring finite delays [2], propositions divided by \iff and the set of extensionally equal functions. Also set-theoretical finite sets can be implemented as the quotient of lists in Type Theory.

In fact, from any given function $f : A \rightarrow B$, we can get an equivalence relation $\sim : A \rightarrow A \rightarrow \mathbf{Prop}$ which is defined as $a \sim b \stackrel{\text{def}}{=} f a \equiv f b$. Then there could be a quotient type represents A / \sim .

Moreover all types can be seen as quotient types. The types without specified equivalence relation can be seen as the quotient of itself by the trivial equivalence relation, namely the definitional equality.

If we can formalise quotient types just by the setoids, it is the best solution. However it seems impossible to do that without extending intensional Type Theory by several axioms. There is another choice, we can construct a type from scratch and prove it is the quotient of a given setoid [2]. I will explain the quotient interface and discuss some examples later. For instances in [15], the normal form integers, reducible rational numbers are definable and proved to construct quotients with respect to the corresponding setoids, such that we can treat them as the quotient types. The advantage of this idea is, it is feasible in current setting of intensional Type Theory, but we can still achieve some convenience from constructing quotients as we will discuss later. The disadvantage this only work for the quotient types which are definable in intensional Type Theory. Another disadvantage is we have to manually construct the quotient types and the proof that it can be used to formalise required quotients, manually define functions or predicates by lifting with proofs that these functions or predicates respect the equivalence relation[9]. Moreover, if the quotient sets cannot be constructed, like the normal forms of real numbers, \mathbb{R} , even set theoretically speaking \mathbb{R} can be seen as the quotient set of the cauchy sequences of rational numbers \mathbb{Q} and the equivalence relation that two sequences converge to zero.

1.4 The relation between equality and quotient types

As we have discussed before, we distinguished definitional equality and propositional equality in intensional Type Theory. Within the setoid, the equivalence

relation is usually propositional equality which is non-trivial despite the case that it is the same with definitional equality. Hence the type-checking which depends on definitional equality does not respect the equivalence relation.

There are also two different propositions expressing the equality between two elements in Type Theory [14]. Both of them require the types of two elements are definitionally equal. One is intensional equality written as $Id(A, a, b)$ and it is inhabited only we have a proof showing a and b are definitionally equal. The other is extensional equality written as $Eq(A, a, b)$, the elements do not depend on an element of A and the largest difference is if $Eq(A, a, b)$ is inhabited, then a converts to b and vice versa. The latter one will make type-checking undecidable so we usually use the first one which is available in intensional Type Theory. For example in Agda, it is redefined as $a \equiv b$ with the type A implicitly, and it has an unique element *refl*.

Intensional equality is enough for many quotients. However if we want quotient types corresponding to the sets of functions, the intensional equalities of functions are inhabited [1]. This is because functions are equated with respect to extensional features rather than intensional constructions, i.e. normal forms. Hence the functional extensionality which is not inhabited in original intensional Type Theory will be required. The introduction of functional extensionality is another big problem of intensional Type Theory, but fortunately Altenkirch introduces a solution in [1].

Most of the topics concerning quotient types are closely related to equality. One of the main issues of quotient types is how to lift the functions for base types to the ones for quotient types. Only functions respects the equivalence relation can be lifted, even in the extensional Type Theory as we will discussed later.

1.5 Literature Reviews

In [5], Mendler et al. has considered building new types from a given type by a quotient operator $//$. Their work is based on an implementation of extensional Type Theory, NuPRL. In NuPRL, every type comes with its own equality relation, so the quotient operator can be seen as a way of redefining equality in a type. But it is not all about quotient types. They also discuss the problems arised when defining functions on the new type. We can illustrate this problem with a simple example. Assume the base set is A and the new equality relation is E , then the new type can be represented by $A//E$. If we want to define a function $f : A//E \rightarrow Bool$, Assume we have two different elements in A , $a, b : A$ such that $E a b$ but $f a \neq f b$, then it becomes inconsistent since $E a b$ implies a converts to b , then $f a = f b$ which contradicts with the assumption $f a \neq f b$. Therefore even in extensional Type Theory, the definition of functions on the quotient types are not so simple. The functions have to respect the equivalence relation, namely

$$\forall a b : A, E a b \rightarrow f a = f b$$

then f is well-defined on the new type. We call it *sound* in [2] and this project.

After the introduction of quotient types, Mendler further investigates this topics from a categorical perspective in [12]. He use the correspondence between quotient types in Martin-Löf type theory and coequalizers in a category of types to define a notion called *squash types* which is further discussed by Nogin.

Hofmann in his PhD thesis [8] proposed three models for quotient types. The first one is to constructs a setoid model for quotient types. In this model all types are attached with partial equivalence relations, namely all types are setoids rather than sets. Types without specific equivalence relation can be translated as setoids with trivial reflection equality. It is similar to NuPRL. While in [9] he gives a simple model in which we have type dependency only at the propositional level, he also shows that extensional Type Theory is conservative over intensional Type Theory extended with quotient types and a universe [10].

Nogin [13] considers a modular approach to axiomatizing the same quotient types also in NuPRL. He also discusses a few complicated problems about quotient types despite the ease of constructing new types from base types. For example, since the equality is extensional, we can not recover the witness of the equality. So he suggests to include more axioms to conceptualise quotients. He decomposes the concept of quotient type into several more primitive concepts such that the quotient types can be formalised based on these concepts and can be handled much simpler.

Homeier [11] axiomatises quotient types in Higher Order Logic (HOL), which is also a theorem prover. He creates a tool package to construct quotient types as a conservative extension of HOL such that users are able to define new types in HOL. Next he defines the normalisation functions and proves several properties of these. Finally he discussed the issues when quotienting on the aggregate types such as lists and pairs.

Courtieu [7] shows an extension of Calculus of Inductive Constructions with a different notion called *Normalised Types* which are similar to quotient types, but equivalence relations are replaced by normalisation functions. Normalised types are proper subsets of quotient types,

$$(A, Q, [\cdot] : A \rightarrow Q) \Rightarrow (A, \lambda a b \rightarrow [a] = [b])$$

However not all quotient types have normal forms. Therefore it only solves part of the problem.

Similarly, Barthe and Geunvers [3] also proposes *congruence types*, which is also a special class of quotient types, in which the base type are inductively defined and with a set of reduction rules called the term-rewriting system. [The idea behind it is the \$\beta\$ -equivalence is replaced by a set of \$\beta\$ -conversion rules.](#) The congruence types can be treated as an alternative to pattern matching introduced in [6]. Hence it aims at solving problems in term rewriting systems rather than simply implementing quotient types.

2 Aims and Objectives of the Project

The objective of this project is to investigate and explore the ways to implement quotient types in Type Theory, especially in intensional one. As we have seen

quotients are quite useful in implementing mathematical objects and programming datatypes, it will be very helpful if we can define quotients in theorem prover like Agda. Also to implement some other undefinable quotients such as Real numbers, it is an unavoidable issue to implement the idea of quotient.

The current aim is to implement some definable quotients, trying to build a model of them and study the benefits. We also need to do research on the different definitions of quotients.

Next we need to investigate on the undefinable quotients such as real numbers and partiality monads and prove why they are undefinable. The key different characters between definable and undefinable quotients will be studied.

3 Theoretical Methods

The work will be implemented in Agda, which is a dependent typed programming language and mainly used as a theorem prover. It has dependent type so that we can use Curry-Howard correspondence between types and propositions. Since propositions can be represented as types, its type checker can verify the proof.

In this project the work will be proved in Agda and also verified in Agda since it is a good choice of intensional Type Theory.

4 Results and Discussion

4.1 Definitions

Currently, we have done some work on the framework of quotient. We have submitted a paper [2] for APLAS 2011. It is about the definable quotients and some undefinable quotients. Here we only talk about the quotient set, but it is universal polymorphic.

To associate a setoid (A, \sim) with a set Q , we have several definitions as in [2], I will not present it again but explain some ideas behind them.

Given a setoid (A, \sim) , we denote the set of equivalence classes as A / \sim and the normalisation function is $[\cdot]_{\sim} : A \rightarrow A / \sim$, assigning each elements to the set it is belonging to. Hence we have

Proposition 5. $\forall a, b : A, a \sim b \iff [a]_{\sim} = [b]_{\sim}$

And the normalisation function is surjective, hence we assume classically,

Proposition 6. $\forall e : A / \sim, \exists a : A, [a]_{\sim} = e$

Namely, the normalisation function is split,

Proposition 7. $\exists s : A / \sim \rightarrow A, [\cdot]_{\sim} \circ s = 1_{A / \sim}$

Since then

$$[\cdot]_{\sim} \circ (s \circ [\cdot]_{\sim}) = [\cdot]_{\sim} \circ 1_{A / \sim}$$

And with Proposition 5, we can prove that

Proposition 8. $\forall a : A, (s \circ [\cdot]_{\sim}) a \sim a$

Some of them are only classically true. However, we worked in intensional Type Theory which is constructive. What we do is to associate a given set Q to the setoid or the quotient set. Given a function $[\cdot] : A \rightarrow Q$,

$$\text{sound} : (a, b : A) \rightarrow a \sim b \rightarrow [a] = [b]$$

is a property which means that from the images of the elements from the same equivalence class are identical, namely $[\cdot]$ respects the equivalence relation. It is also equivalent to say that there is a naming function, $na : A / \sim \rightarrow Q$, such that the following diagram commutes,

$$\begin{array}{ccc} A & \xrightarrow{[\cdot]_{\sim}} & A / \sim \\ & \searrow [\cdot] & \downarrow na \\ & & Q \end{array}$$

And na can be constructed as $[\cdot] \circ s$. We can prove this diagram is commute as,

$$na \circ [\cdot]_{\sim} = [\cdot] \circ s \circ [\cdot]_{\sim}$$

Apply Proposition 8 and Proposition 5, we know,

$$\forall a : A, (na \circ [\cdot]_{\sim}) a = ([\cdot] \circ s \circ [\cdot]_{\sim}) a = [\cdot] \circ ((s \circ [\cdot]_{\sim}) a) = [\cdot] a$$

Extensionally, we proved that the diagram commute.

However, with this property, we cannot confirm Q is the required quotient set, we only construct a prequotient. To complete a quotient, we require one eliminator for every $B : Q \rightarrow \mathbf{Set}$,

$$\begin{aligned} \text{qelim}_B : & (f : (a : A) \rightarrow B [a]) \\ & \rightarrow ((p : a \sim b) \rightarrow f a \simeq_{\text{sound } p} f b) \\ & \rightarrow ((q : Q) \rightarrow B q) \end{aligned}$$

such that $\text{qelim-}\beta : \text{qelim}_B f p [a] \equiv f a$.

With this eliminator we can lift a function which takes in $a : A$ but the result is dependent on the $[a] : Q$ and identifies more than $[\cdot]$, namely for the elements in the same equivalence class by $[\cdot]$, the result produced by f is the same. Combining with this function,

However this is not a exact quotient, since it is unnecessary for na function to assign one *name* ($q : Q$) to each equivalence class ($e : A / \sim$). It is very inefficient to define a too general quotient. Therefore we need a property to make a quotient exact,

$$\text{exact} : \forall a b : A, [a] = [b] \rightarrow a \sim b$$

equivalently, we have the property that na is injective,

$$\begin{aligned} \forall e f : A / \sim, na e = na f &\Rightarrow ([\cdot] \circ s) e = ([\cdot] \circ s) f \\ &\Rightarrow [s e] = [s f] \Rightarrow s e \sim s f \Rightarrow [s e]_{\sim} = [s f]_{\sim} \Rightarrow e = f \end{aligned}$$

The alternative definition of quotient with non-dependent eliminator introduced in [8], and consists of,

$$\begin{aligned} \text{lift}_B : (f : A \rightarrow B) &\rightarrow (\forall a, b \cdot a \sim b \rightarrow f a \equiv f b) \rightarrow (Q \rightarrow B) \\ \text{lift-}\beta : \text{lift}_B f p [a] &\equiv f a \end{aligned}$$

for any $B : \mathbf{Set}$, which can lift a function f which respects the equivalence relation and the following diagram commute with respect to $\text{lift-}\beta$,

$$\begin{array}{ccc} A & \xrightarrow{[\cdot]} & Q \\ & \searrow f & \downarrow \text{lift}_B f p \\ & & B \end{array}$$

In this definition we also need an introduction principle if B in the dependent eliminator is a predicate on Q ,

$$\text{qind}_P : ((a : A) \rightarrow P [a]) \rightarrow ((q : Q) \rightarrow P q)$$

The quotient with dependent eliminator and the one with non-dependent eliminator are actually equivalent. We prove this by formalise one by another in Agda. It is quite trivial to generate the non-dependent version from dependent version since lift_B and qind_P are both special cases of the dependent eliminator. However to recover dependent eliminator, it is a little complicated. We need a function *indep* to transform the dependent $a : A \rightarrow B [a]$ into the non-dependent $A \rightarrow \Sigma Q B$ which is defined as $\text{indep } f a \mapsto [a], f a$. Then we can use non-dependent eliminator to lift $\text{indep } f$ and the projection of the second component is the same as dependent function. You can check the detailed Agda proof in the Appendix.

When the quotient type is definable and we want the target type Q is just the quotient type, which means

$$Q \cong A / \sim$$

Therefore, to constructively define isomorphism in intensional Type Theory, we not only need na , but also the inverse function of it. So the definable quotients in [2] is the prequotient with

$$\begin{aligned}
& \text{emb} : Q \rightarrow A \\
& \text{complete} : (a : A) \rightarrow \text{emb}[a] \sim a \\
& \text{stable} : (q : Q) \rightarrow [\text{emb } q] \equiv q
\end{aligned}$$

emb is the embedding function which choose one representative element for each equivalence class. Hence the following diagram needs to commute,

$$\begin{array}{ccc}
& A & \\
\text{emb} \nearrow & & \nwarrow s \\
Q & \xrightleftharpoons[\text{na}^{-1}]{\text{na}} & A / \sim
\end{array}$$

Such that $\text{na}^{-1} = [\cdot]_{\sim} \circ \text{emb}$ is the inverse function of na .

$$\begin{aligned}
\text{na}^{-1} \circ \text{na} &= 1_{A/\sim} \Rightarrow [\cdot]_{\sim} \circ \text{emb} \circ [\cdot] \circ s = 1_{A/\sim} \\
&\Rightarrow \forall a : A, a \sim (s \circ [\cdot]_{\sim}) a \sim (s \circ 1_{A/\sim} \circ [\cdot]_{\sim}) a \\
&\sim (s \circ ([\cdot]_{\sim} \circ \text{emb} \circ [\cdot] \circ s) \circ [\cdot]_{\sim}) a \sim \text{emb}[a]
\end{aligned}$$

So we need this property called completeness which ensures the correctness of emb. Also,

$$\text{na} \circ \text{na}^{-1} = 1_Q \Rightarrow [\cdot] \circ s \circ [\cdot]_{\sim} \circ \text{emb} = [\cdot] \circ \text{emb} = 1_Q \Rightarrow \forall q : Q, [\text{emb } q] = q$$

is needed called the stable property which ensures $[\cdot]$ is surjective, Hence it is *normalisation* function and the Q is the quotient type without redundance.

With these two properties, we can conclude that $[\cdot]_{\sim} \circ \text{emb}$ is the inverse function of na , hence Q is isomorphic to A / \sim . We can use it as the quotient type.

In category theory, coequalizers are the generalization of quotients. We assume

$$\begin{aligned}
R &= \Sigma a, b : A, a \sim b \text{ are the pairs of equivalent elements in } A \\
\pi_0, \pi_1 : R &\rightarrow A \text{ are the projection functions for } R \\
[\cdot] : A &\rightarrow Q \text{ satisfies that sound: } \forall a, b : A, a \sim b \rightarrow [a] = [b]
\end{aligned}$$

$$\begin{array}{ccc}
R & \xrightleftharpoons[\pi_1]{\pi_0} & A \xrightarrow{[\cdot]} Q \\
& & \searrow f \quad \downarrow \hat{f} \\
& & X
\end{array}$$

Since

1. $(Q, [\cdot])$ fulfils that $[\cdot] \circ \pi_0 = [\cdot] \circ \pi_1$, we can acquire this from applying the $\pi_0 r, \pi_1 r$ for all r to sound.
2. Given any $(X, f: A \rightarrow X)$, there exists a unique \hat{f} , such that the diagram above commutes. From the definition of quotients, we can use the eliminator to lift f , namely $\hat{f} = \text{lift } f$, and the β -law simply implies the diagram commutes. The uniqueness can be proved as follows

$$\forall g: Q \rightarrow X, g \circ [\cdot] = f \Rightarrow \forall a: A, g[a] = fa = \text{lift } f \, q[a] \Rightarrow g = \text{lift } f \, q$$

These two parts proved from quotients exactly define a coequalizer. Also we can prove $[\cdot]$ is an epimorphism

$$\begin{aligned} \forall g_1, g_2: Q \rightarrow Z, g_1 \circ [\cdot] &= g_2 \circ [\cdot] \\ \Rightarrow \forall q: Q, g_1 q &= \text{lift } (g_1 \circ [\cdot]) q = \text{lift } (g_2 \circ [\cdot]) q = g_2 q \Rightarrow g_1 = g_2 \end{aligned}$$

Also the exact quotient is equivalent to the exact coequalizer,

$$\begin{array}{ccc} R & \xrightarrow{\pi_2} & A \\ \pi_1 \downarrow & \lrcorner & \downarrow [\cdot] \\ A & \xrightarrow{[\cdot]} & Q \end{array}$$

1. This diagram commutes
 $(\forall r: R, \pi_1 r \sim \pi_2 r \Rightarrow [\pi_1 r] = [\pi_2 r]) \Rightarrow [\cdot] \circ \pi_1 = [\cdot] \circ \pi_2$
- 2.

$$\begin{aligned} \forall (Z, z_1: Z \rightarrow A, z_2: Z \rightarrow A), [\cdot] \circ z_1 &= [\cdot] \circ z_2 \\ \Rightarrow (\exists u: Z \rightarrow R, \pi_1 \circ u &= z_1 \wedge \pi_2 \circ u = z_2 \\ \wedge \forall u': Z \rightarrow R, \pi_1 \circ u' &= z_1 \wedge \pi_2 \circ u' = z_2 \\ \Rightarrow u &= u') \end{aligned}$$

We can construct the unique function as $u x \mapsto z_1 x, z_2 x$, but we need to prove $z_1 x \sim z_2 x$ from exact property of quotient,

$$[\cdot] \circ z_1 = [\cdot] \circ z_2 \Rightarrow \forall x: Z, ([z_1 x] = [z_2 x] \Rightarrow z_1 x \sim z_2 x)$$

u is the function which makes the diagram commutes,

$$\forall x: Z, (\pi_1 \circ u) x = z_1 x$$

$$\forall x: Z, (\pi_2 \circ u) x = z_2 x$$

u is unique,

$$\begin{aligned} \forall u': Z \rightarrow R, \pi_1 \circ u' &= z_1 \wedge \pi_2 \circ u' = z_2 \\ \Rightarrow \forall x: Z, u' x &= z_1 x, z_2 x = u x \Rightarrow u' = u \end{aligned}$$

5 Examples

We have already define the basic requirements to create quotients in intensional Type Theory, I will then present some concrete examples in [15] to illustrate these ideas. They are implemented in Agda.

5.1 Integers

All the result of subtraction between natural numbers are integers. Therefore it is naturally to define a pair of natural numbers to represent integers. Hence the base type of the quotient is

$$\mathbb{Z}_0 = \mathbb{N} \times \mathbb{N}$$

Mathematically, for any two pairs of natural numbers (n_1, n_2) and (n_3, n_4) ,

$$n_1 - n_2 = n_3 - n_4 \iff n_1 + n_4 = n_3 + n_2$$

since the pair of integers represent the same result of subtraction, they define the same integer. Hence we can define an equivalence relation for $\mathbb{N} \times \mathbb{N}$ as

$$(n_1, n_2) \sim (n_3, n_4) = n_1 + n_4 \equiv n_3 + n_2$$

Here \equiv is the propositional equality. so that the \mathbb{Z}_0 / \sim is the quotient integer. Integer is also definable in intensional Type Theory as $\mathbb{N} + \mathbb{N}$ where we define two constructors

$$(n : \mathbb{N}) \Rightarrow +n : \mathbb{Z}$$

$$(n : \mathbb{N}) \Rightarrow -\text{suc } n : \mathbb{Z}$$

Firstly to construct the prequotient based on the setoid (\mathbb{Z}_0, \sim) , we need to define the $[\cdot] : \mathbb{Z}_0 \rightarrow \mathbb{Z}$ as

$$[(a, 0)] = +a$$

$$[(0, \text{suc } b)] = -\text{suc } b$$

$$[(\text{suc } a, \text{suc } b)] = [(a, b)]$$

and prove *sound*. Then we define *emb* function and prove all the required properties for definable quotients

$$\text{emb}(+a) = (a, 0)$$

$$\text{emb}(-\text{suc } b) = (0, b + 1)$$

We have done these in Agda [15]. The quotients here are not just something relate the setoid with the quotient type, we use lift functions to define functions

trivially and use properties to transform the proof term for the setoid to the quotient type. For instance, the addition of the setoid is defined as

$$(a, b) +_0 (a', b') = (a + a', b + b')$$

We can then define use the eliminator to lift the operator, or just define a lift function for binary operators,

$$\text{lift } * z_1 z_2 = [\text{emb } ztext_1 * \text{emb } z_2]$$

or a more general lift function for n-ary operators,

$$\begin{aligned} \text{lift}' 0 \text{ op} &= [\text{op}] \\ \text{lift}' (\text{suc } n) \text{ op} &= \lambda x \rightarrow \text{lift}' n (\text{op } (\text{emb } x)) \end{aligned}$$

Then we don't need to define the addition of integers by several cases.

$$+ = \text{lift } +_0$$

If we lift the operators in this way, we have to prove it respects the equivalence relation later. The main benefits from the quotients arise in proving properties. Because for normal form integers, we have two cases for each argument. The number of cases will expand exponentially if we can not combine cases. The proof of distributivity of multiplication over addition is so cumbersome that it is hard to write and read. However, we could lift the proof for the setoid integers so that we could prove it in one case. This convenience is due to the simplicity of the proof for the setoid (\mathbb{Z}_0, \sim) .

5.2 Rational numbers

The quotients of rational numbers is more natural to understand and the normalisation function is also commonly used in regular mathematics. Generally we can use a pair of integers to represent rational numbers. However, it is complicated to exclude 0 in the denominator. For simplicity, we just use one integer for *numerator* and one natural number for *denominator-1* to represent a rational number to avoid the invalid cases from construction.

$$\mathbb{Q}_0 = \mathbb{Z} \times \mathbb{N}$$

The equivalence relation is

$$(n_1, d_1) \sim (n_2, d_2) = n_1 \times (d_2 + 1) \equiv n_2 \times (d_1 + 1)$$

The normal form of rational numbers can just be defined by adding a condition that the numerator and denominator are coprime.

$$\mathbb{Q} = \Sigma(n : \mathbb{Z})(d : \mathbb{N}), \text{Coprime } n (d + 1)$$

Since there are a set of *gcd* (great common divisor) functions in Agda, it is possible to define the normalisation functions (See Appendix). *emb* function can be trivially defined by forgetting coprime proof.

5.3 Real numbers as cauchy sequences

We can represent real numbers as cauchy sequences of rational numbers [4].

$$\mathbb{R}_0 = \{s : \mathbb{N} \rightarrow \mathbb{Q} \mid \forall \varepsilon : \mathbb{Q}, \varepsilon > 0 \rightarrow \exists m : \mathbb{N}, \forall i : \mathbb{N}, i > m \rightarrow |s\ i - s\ m| < \varepsilon\}$$

And we define the equivalence relation of two sequences by the proposition that their pointwise difference converges to 0.

$$r \sim s = \forall \varepsilon : \mathbb{Q}, \varepsilon > 0 \rightarrow \exists m : \mathbb{N}, \forall i : \mathbb{N}, i > m \rightarrow |r\ i - s\ i| < \varepsilon$$

Then \mathbb{R}_0 / \sim is the quotient set of real numbers. However it is undefinable because real numbers have no normal forms. Therefore we cannot use the definable quotient interface for it. The undefinability is proved in [2]. Nevertheless, we could easily embedding rational numbers as the cauchy sequences of all the same rational numbers. But for irrational numbers, there is no such an uniform way to generate a sequences.

6 Conclusion

Currently we investigate the possible quotient definitions in intensional Type Theory and present some examples and benefits from the definable quotients. For definable quotients, it provides an alternative choice to define functions or prove propositions which reuses things and could be simpler in most cases. However, to solve the problems arose from undefinable quotients, a new type former may be unavoidable.

TO DO: future extension.

If we axiomize quotient types in intensional Type Theory, then every type can be seen as quotient type, when the default equivalence relation is just [reflection equality](#).

References

1. Thorsten Altenkirch. Extensional equality in intensional type theory. In *14th Symposium on Logic in Computer Science*, pages 412 – 420, 1999.
2. Thorsten Altenkirch, Thomas Anberrée, and Nuo Li. Definable quotients in type theory. 2011.
3. Gilles Barthe and Herman Geuvers. Congruence types. In *Proceedings of CSL’95*, pages 36–51. Springer-Verlag, 1996.
4. Errett Bishop and Douglas Bridges. *Constructive Analysis*. Springer, New York, 1985.
5. Robert L. Constable, Stuart F. Allen, S. F. Allen, H. M. Bromley, W. R. Cleaveland, J. F. Cremer, R. W. Harper, Douglas J. Howe, T. B. Knoblock, N. P. Mendler, P. Panangaden, Scott F. Smith, James T. Sasaki, and S. F. Smith. Implementing mathematics with the nuprl proof development system, 1986.
6. Thierry Coquand. Pattern matching with dependent types. In *Types for Proofs and Programs*, 1992.
7. Pierre Courtieu. **Normalized types**. In *Proceedings of CSL2001*, volume 2142 of *Lecture Notes in Computer Science*, 2001.
8. Martin Hofmann. *Extensional concepts in intensional type theory*. PhD thesis, School of Informatics., 1995.
9. Martin Hofmann. A simple model for quotient types. In *Proceedings of TLCA’95, volume 902 of Lecture Notes in Computer Science*, pages 216–234. Springer, 1995.
10. Martin Hofmann. Conservativity of equality reflection over intensional type theory. In *Selected papers from the International Workshop on Types for Proofs and Programs, TYPES ’95*, pages 153–164, London, UK, 1996. Springer-Verlag.
11. Peter V. Homeier. Quotient types. In *In TPHOLs 2001: Supplemental Proceedings*, page 0046, 2001.
12. N.P. Mendler. Quotient types via coequalizers in martin-lof type theory. In *Proceedings of the Logical Frameworks Workshop*, pages 349–361, 1990.
13. Aleksey Nogin. Quotient types: A modular approach. In *ITU-T Recommendation H.324*, pages 263–280. Springer-Verlag, 2002.
14. Bengt Nordström, Kent Petersson, and Jan M. Smith. *Programming in Martin-Löf’s type theory: an introduction*. Clarendon Press, New York, NY, USA, 1990.
15. Li Nuo. Representing numbers in agda. 2010.