Definable Quotients in Type Theory

Thorsten Altenkirch[1], Thomas Anberrée[2], and Nuo Li[2]

[1] School of Computer Science, University of Nottingham, Jubilee Campus, Wollaton Road, Nottingham, NG8 1BB, UK

[2] School of Computer Science, University of Nottingham, Ningbo Campus, 199 Taikang East Road, Ningbo, 315100, China

Abstract. In Type Theory, a quotient set is a set representing a setoid. Categorically, this corresponds to the concept of an exact coequalizer. In the present paper we consider the case of a *definable quotients*, where the quotient set arises as the codomain of a normalization function — this corresponds to the notion of a split coequalizer. We give a number of examples of definable quotients and notice that it is preferable to use the setoid structure when reasoning about the quotient set. We also show that there are examples where setoids cannot be represented in ordinary Type Theory such as the real numbers or the partiality monad under the assumption that local continuity is admissible in Type Theory.

1 Introduction

In Intensional Type Theory [11], quotient types are unavailable and we use setoids [4] instead. Setoids are just sets together with an equivalence relation. However, the disadvantage of using setoids is that we have now to lift any operation on sets to an operation on setoids. E.g. we need lists as an operation on setoids and not just on sets. Moreover, setoids are not safe in the sense that any consumer of a setoid may access the underlying representation. One way out is to use a Type Theory which supports genuine quotients such as the forthcoming Epigram 2 system (based on [3]). However, in many cases this is not necessary because the quotient is actually definable. This is the subject of the current paper.

An example is the case of integers. We can define integers as a setoid, namely as the setoid given by pairs of natural numbers $\mathbb{Z}_0 = \mathbb{N} \times \mathbb{N}$, where the equivalence relation identifies pairs representing the same difference, that is $(a, b) \sim (c, d)$ iff $a + d = c + b$. However, as it is well known, using a setoid here is unnecessary; we can use a set, namely $\mathbb{N} + \mathbb{N}$ where the first injection represents the positive numbers including 0 whereas the second injection represents the proper negative numbers. We can now define operations like addition and multiplication and show algebraic properties, such as verifying that the structure is a ring. However, this is quite complicated and uses many unnecessary case distinctions. E.g. try to prove distributivity within this setting! It is easier to define the operations on the setoid and the required algebraic properties are direct consequences of the semiring structure of the natural numbers.

Hence we propose to use both the setoid and the associated set, but to use the setoid structure to define operations on the quotient set and to reason about it. In the present paper we introduce the formal framework to do this, i.e. we give the definitions of quotient as well as definable quotients and show the equivalence of alternative definitions of quotients. We also verify that quotients correspond precisely to the notion of coequalizers, and that an additional condition, exactness, can equivalently be expressed in Type Theory or in category theory. We present a number of examples for definable quotients which are the base of a library of definable quotients.

However, not all setoids can be represented as definable quotients. Under the (reasonable) assumption of local continuity, we show that the real numbers are not a definable quotient. Another important example is the partiality monad. These counterexamples suggest that it pays off to move to a type theory where all quotient types exist — i.e. the type theory corresponding to a Heyting pretopos [8]. In this context our work can be seen as an exploration of the use of quotients within the settings of Intensional Type Theory.

1.1 Type Theory basics

We use standard type theoretic notation, inspired by Agda [12]. We write $(x : A) \to B$ for dependent function types (Π-types) and $\Sigma x : A.B$ for dependent product types (Σ-types). We assume that strictly positive inductive and coinductive types such as natural numbers $\mathbb{N}$, booleans Bool, disjoint union $A{+}B$ and lists List A are defined. We also use the family of finite sets $\mathrm{Fin} : \mathbb{N} \to \mathbf{Set}$ with $\mathrm{Fin}\,n = \{0, 1, \ldots, n-1\}$ which can be inductively generated from $0 : \mathrm{Fin}\,(n+1)$ and $+1 : \mathrm{Fin}\,n \to \mathrm{Fin}\,(n{+}1)$. We write **Set** for the universe of small sets.We write **Prop** for the subuniverse of propositions that are sets which (extensionally) have at most one inhabitant (proof-irrelevance). We assume that **Prop** contains the equality type $a = b : \mathbf{Prop}$ for any $a, b : A : \mathbf{Set}$[3] and is closed under universal ($\forall$) and existential ($\exists$) quantification. While $\forall$ exactly corresponds to a Π-type, $\exists$ is the squashing [9] of the corresponding Σ-type. **Prop** is also closed under implication ($\Longrightarrow$), conjunction ($\wedge$) which is interpreted as a Σ-type where both components are propositional (and can be dependent) while $P \vee Q$ can be defined using $\exists$ and Bool. Subset comprehension over a predicate $P : A \to \mathbf{Prop}$ is interpreted as the corresponding Σ-type, i.e. $\{a : A \mid P\,a\} = \Sigma a : A\ .\ P$. Due to proof irrelevance, the projection $\{a : A \mid P\,a\} \to A$ is an injection and we will omit it if it is obvious from the context. We also omit implicit arguments and to improve readability, we will even omit the declaration of implicitly quantified arguments, assuming that the human reader, unlike a machine, can reconstruct those. Given elements $b : B\,a$ and $b' : B\,a'$ with a proof $p : a = a'$, we write $b \simeq_p b'$ for the *heterogeneous equality* subst $B\,p\,b = b'$. Most of our examples do not require functional extensionality, but if we do we assume that it is present in form of an uninterpreted constant $\mathrm{Ext} : (\forall (x : A) \to f\,x = g\,x) \to f = g$, which

[3] This is consistent with Voevodsky's univalence interpretation of Type Theory [14], if one identify sets with types whose h-level is 2.

is justified by Hofmann's observation that extensional Type Theory is a conservative extension of the theory considered in the present paper [7]. Alternatively, we can eliminate Ext as suggested in [1].

1.2 Related Work

Quotient types were introduced by Mendler in [9] and subsequently investigated in Hofmann's PhD dissertation [7]. An extensive investigation of setoids can be found in [4]. Maetti considers extensions of both intensional and extensional Type Theory by quotient types [8]. Courtieu considers an extension of CIC (an intensional Type Theory) by *normalized types* corresponding to our definable quotients [6]. Nogin describes a modular implementation of quotient types in NuPRL (an extensional Type Theory) [10].

1.3 Main results

We develop the notion of a definable quotient within an existing intensional Type Theory instead of an extension by a new type former. This enables us to formally verify a number of basic results in Agda (see appendix), such as the relation between exact quotients, coequalizers and definable quotients. We give a number of examples for definable quotients, some which might seem surprising such as the presentation of multisets over higher order types. Finally, we show that certain quotient types are not definable quotients in our sense.

2 Setoids

We review the notion of a setoid and give a number of examples which we are going to use subsequently.

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

2.1 Examples

Integers The integers can be viewed as the setoid $(\mathbb{Z}_0 = \mathbb{N} \times \mathbb{N}, \sim)$ where $(a, b) \sim (c, d)$ iff $a + d = c + b$ reflecting the idea that (a, b) represents the integer $a - b$.

Rational numbers The rational numbers can in turn be defined as $(\mathbb{Z} \times \mathbb{N}, \sim)$ where $(x, m) \sim (y, n)$ iff $x \times (n + 1) = y \times (m + 1)$, reflecting that (x, m) represents the quotient $\frac{x}{m+1}$.

The real numbers The real numbers can then be defined as $(\mathbb{R}_0, \sim)$ where $\mathbb{R}_0$ is the set of Cauchy sequences and two sequences are equivalent iff their pointwise difference converges to 0.

$$\begin{aligned} \mathbb{R}_0 &= \{s : \mathbb{N} \to \mathbb{Q} \mid \forall \varepsilon : \mathbb{Q}, \quad \varepsilon > 0 \to \exists m : \mathbb{N}, \forall i : \mathbb{N}, i > m \to |s\,i - s\,m| < \varepsilon\} \\ r \sim s &= \forall \varepsilon : \mathbb{Q}, \varepsilon > 0 \to \exists m : \mathbb{N}, \forall i : \mathbb{N}, i > m \to |r\,i - s\,i| < \varepsilon \end{aligned}$$

Unordered pairs Given a set A, the unordered pairs of elements of A is the setoid $(A \times A, \sim)$ where $\sim$ is reflexive and $(a,b) \sim (b,a)$.

Finite multisets Given a set A , the finite multisets of elements in A is the setoid $(\mathrm{List}\, A, \sim)$ where two lists are equivalent iff one is the permutation of the other.

$$\mathrm{List}\, A = \Sigma n : \mathbb{N}.\, \mathrm{Fin}\ n \to A$$
$$(m,f) \sim (n,g) = \exists \varphi : \mathrm{Fin}\ m \to \mathrm{Fin}\ n \cdot \ \mathrm{Bijection}\ \varphi \wedge g \circ \varphi = f$$

Notice that $(m,g) \sim (n,f) \implies m = n$ is provable in Type Theory, based on the definition of $\mathrm{Bijection} : (A \to B) \to \mathbf{Prop}$ which we omit here.

Finite sets Given a set A, the finite sets of elements in A is the setoid $(\mathrm{List}\, A, \sim)$ where two lists are equivalent iff they contain the same elements :

$$(m,f) \subseteq (n,g) = \exists \varphi : \mathrm{Fin}\ m \to \mathrm{Fin}\ n \cdot g \circ \varphi = f$$
$$(m,f) \sim (n,g) = (m,f) \subseteq (n,g) \wedge (n,g) \subseteq (m,f).$$

For example the lists $[1,2,1]$ and $[1,2]$ are equivalent and both represent the set $\{1,2\}$.

Partiality monad Given a set A, the set of partial computations over A is given by $(A_{\perp_0}, \sim)$ where $A_{\perp_0}$ is the set of delayed computations over A and $\sim$ is a weak bisimilarity ignoring finite delays. We define $A_{\perp_0}$ as generated by the constructors

$$\mathrm{now} : A \to A_{\perp_0}$$
$$\mathrm{later} : \infty A_{\perp_0} \to A_{\perp_0}$$

where ∞ indicates a coinductive premise — categorically this is the terminal coalgebra of $F\, X = A + X$. We inductively define the relation $- \downarrow - : A_{\perp_0} \to A \to \mathbf{Prop}$ with the idea that $d \downarrow a$ means that the computation d terminates with a, by the following rules:

$$\frac{}{\mathrm{now}\ a \downarrow a} \qquad \frac{d \downarrow a}{\mathrm{later}\ d \downarrow a}$$

We define the termination order $\sqsubseteq : A_{\perp_0} \to A_{\perp_0} \to \mathbf{Prop}$ as $d \sqsubseteq d' = \forall a : A. d \downarrow a \to d' \downarrow a$ and $d \sim d' = d \sqsubseteq d' \wedge d' \sqsubseteq d$. See [5].

3 Quotients and coequalizers

We define what an (exact) quotient over a setoid is and relate this to an alternative definition given by Hofmann and to the categorical definition. All the concepts have been formalized in Agda (see Appendix A).

Definition 2 (prequotient, quotient, exact quotient).
Given a setoid $(A, \sim)$*, a* prequotient $(Q, [\,\cdot\,], \mathrm{sound})$ *over that setoid consists in*

1. *a set* Q*,*
2. *a function* $[\,\cdot\,] : A \to Q$*,*
3. *a proof* sound *that the function* $[\,\cdot\,]$ *is compatible with the relation* $\sim$*, that is*

$$\mathrm{sound}\colon (a, b : A) \to a \sim b \to [a] = [b],$$

Such a prequotient is a quotient *if we also have*

4. *for any* $B : Q \to \mathbf{Set}$*, an eliminator*

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

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

Finally, such a quotient is exact *if additionally we have a proof*

5. $\mathrm{exact} : (\forall a, b : A) \to [a] = [b] \to a \sim b.$

There are two special cases of the eliminator qelim_B described in item 4. One is if B is not dependent,

$$\mathrm{lift}\colon (f\colon A \to B) \to (\forall a, b \cdot a \sim b \to f\,a = f\,b) \to (Q \to B)$$

and the other is if B is a predicate, i.e. $B : Q \to \mathbf{Prop}$, in which case we get an induction principle:

$$\mathrm{qind}\colon ((a : A) \to B\,[a]) \to ((q : Q) \to B\,q)$$

since the condition $((p : a \sim b) \to f\,a \simeq_{\mathrm{sound}\ p} f\,b)$ of the eliminator is trivially satisfied. These two special cases are in fact sufficient to recover the eliminator, which is reminiscent of the fact that dependent elimination for the natural numbers can be constructed from non-dependent elimination and an induction principle.

Proposition 3. *A prequotient* $(Q, [\,\cdot\,], \mathrm{sound})$ *with*

1. *a non-dependent eliminator*

$$\mathrm{lift}_B\colon (f\colon A \to B) \to (\forall a, b \cdot a \sim b \to f\,a = f\,b) \to (Q \to B)$$

   for any $B\colon \mathbf{Set}$*,*
2. *a* β*-law*

$$\mathrm{lift}\text{-}\beta : \mathrm{lift}_B\ f\,p\,[a] = f\,a,$$

3. an induction principle

$$\mathrm{qind}_P\colon ((a\colon A) \to P\,[a]) \to ((q\colon Q) \to P\,q)$$

gives rise to a quotient $(Q, [\,\cdot\,], \mathrm{sound}, \mathrm{qelim}, \mathrm{qelim}\text{-}\beta)$.

We refer to Appendix A for a formal proof of Proposition 3 and its converse. The characterization in Proposition 3 was given as a definition of quotients in [7].

Quotients correspond to coequalizers in category theory. Let us recall the definition.

Definition 4. *Given two morphisms* $g, h : S \to A$, *a* coequalizer *of* g *and* h *is a morphism* $[\,\cdot\,] : A \to Q$ *such that for any* $f : A \to X$ *satisfying* $f \circ g = f \circ h$, *there exists a unique* $\widehat{f}$ *such that*

$$\begin{tikzcd} S \arrow[r, shift left, "g"] \arrow[r, shift right, "h"'] & A \arrow[r, "{[\,\cdot\,]}"] \arrow[dr, "f"'] & Q \arrow[d, dashed, "\widehat{f}"] \\ & & X \end{tikzcd}$$

A coequalizer is exact *if*

$$\begin{tikzcd} S \arrow[r, "g"] \arrow[d, "h"'] \arrow[dr, phantom, "\lrcorner", very near start] & A \arrow[d, "{[\,\cdot\,]}"] \\ A \arrow[r, "{[\,\cdot\,]}"'] & Q \end{tikzcd}$$

and it is split *if the morphism* $[\,\cdot\,]$ *is a split epi, that is if it has a right inverse* $\mathrm{emb} : Q \to A$.

We observe that there is an exact correspondence between quotients and coequalizers:

Proposition 5. *In the context of Definition 4 above :*

1. *Q is the quotient on $(S, \sim)$ where $s \sim s'$ if and only if $g\,s = h\,s'$. This quotient is exact iff the coequalizer is exact.*
2. *Let R be $\Sigma a, a' : A, a \sim a'$ and $\pi_0, \pi_1 : R \to A$ the projection functions. The quotient for $(R, \sim)$ is then the coequalizer for those projections and it is exact if and only if the coequalizer is exact.*

$$\begin{tikzcd} R \arrow[r, shift left, "\pi_0"] \arrow[r, shift right, "\pi_1"'] & A \arrow[r, "{[\,\cdot\,]}"] \arrow[dr, "f"'] & Q \arrow[d, dashed, "\widehat{f}"] \\ & & X \end{tikzcd}$$

where $\widehat{f} = \mathrm{lift}\,f p$ *and* $p\colon \forall a, b \cdot a \sim b \to f\,a = f\,b$ *follows from* $f \circ \pi_0 = f \circ \pi_1$.

4 Definable quotients

We now consider a general construction which allows us to construct quotients in Type Theory.

Definition 6. *A* definable quotient *is a prequotient* $(Q, [\cdot], \text{sound})$ *on a setoid* $(A, \sim)$ *along with*

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

This is exactly the specification of $[-]$ as a normalisation function with respect to emb (see [2]).

Proposition 7. *All definable quotients are exact quotients.*

Proof. Given $(f{:}\, A \to B)$ and $p : a \sim b \to f\,a = f\,b$, define $\text{lift}\,f\,p\,q = f(\text{emb}\ q)$ from which we get $\text{lift}\,f\,(p : a \sim b)\,[a] = f(\text{emb}\,[a]) = f\,a$ because $\text{emb}\,[a] \sim a$ by completeness and f respects $\sim$ by p.

To derive qind, let $f : (a{:}\, A) \to B\,[a]$ and $q : Q$. Since $[\text{emb}\ q] = q$ by stability, hence from $f(\text{emb}\ q) : B\,[\text{emb}\ q]$ we can derive a proof of $B\,q$.

It follows from Proposition 3 that this defines a quotient.

Finally, from $[a] = [b]$ we obtain by completeness that $a \sim \text{emb}([a]) = \text{emb}([b]) \sim b$ and hence $a \sim b$. That is, the quotient is exact.

4.1 Examples

We revisit the examples of setoids which turn out to correspond to definable quotients.

The integers Define $\mathbb{Z} = \mathbb{N} + \mathbb{N}$ and

$$\begin{aligned} [(a, 0)] &= \text{inl}\ a \\ [(a+1, b+1)] &= [(a, b)] \\ [(0, b+1)] &= \text{inr}\ b \end{aligned}$$

$$\begin{aligned} \text{emb}(\text{inl}\,a) &= (a, 0) \\ \text{emb}(\text{inr}\,b) &= (0, b+1) \end{aligned}$$

The fact that this gives rise to a definable quotient has been verified in Agda [13]. One could of course just use that $\mathbb{Z} = \mathbb{N} + \mathbb{N}$ and define the operations on $\mathbb{Z}$

directly. However, seeing $\mathbb{Z}$ as a quotient is helpful in proving properties of those operations and reflects the usual mathematical definition of the integers. E.g., to define $+$, we define

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

on $\mathbb{Z}_0$ and show that it respects $\sim$. Then by lifting $+_0$, we get $+$ on $\mathbb{Z}$, thus avoiding a rather incomprehensible case analysis. This becomes even more relevant when showing other properties such as distributivity of multiplication over addition [13].

Remark : instead of defining $\mathbb{Z} = \mathbb{N} + \mathbb{N}$, we could have equivalently defined $\mathbb{Z}$ as the subset of canonical elements $\mathbb{Z}_0$, by which we mean

$$\{(a,b) \in \mathbb{Z}_0 \mid (a = 0 \wedge b > 0) \vee b = 0\}.$$

The rational numbers Define $\mathbb{Q} = \{(x,m) : \mathbb{Z} \times \mathbb{N} \mid \gcd x\,(m+1) = 1\}$ and

$$\begin{aligned}[(x,m)] &= \left(\frac{x}{d}, \frac{m+1}{d} - 1\right) \text{ where } d = \gcd x\,(m+1)\\ \mathrm{emb}\,(x,m) &= (x,m)\end{aligned}$$

Note that the greatest common divisor function (gcd) is definable in Type Theory. Completeness comes from the fact that, for any common divisor d of x and $m+1$, it is provable that $\left(\frac{x}{d}, \frac{m+1}{d} - 1\right) \sim (x,m)$ because $\frac{x}{d} \times (m+1) = x \times (\frac{m+1}{d} - 1 + 1)$. Stability holds because whenever $d = \gcd x\,(m+1) = 1$, we have $\left(\frac{x}{d}, \frac{m+1}{d} - 1\right) = (x,m)$.

Unordered pairs The construction of a definable quotient over the setoids of unordered pairs $(A \times A, \sim)$ as defined in Section 2.1 depends on the choice of A. In general we require an order $\leq : A \to A \to \mathbf{Prop}$ together with functions:

$$\min, \max : A \to A \to A$$

calculating the binary minimum and maximum for that order. This allows us to define

$$Q = \{(a,b) \mid a \leq b\}$$

and

$$[(a,b)] = (\min a\,b, \max a\,b).$$

Soundness is obviously satisfied. An embedding of Q into $A \times A$ is simply the first projection — for recall that an element in Q is of the form $((a,b),p)$ where p is a proof that $a \leq b$ (see Section 1) :

$$\begin{aligned}&\mathrm{emb} : Q \to A \times A\\ &\mathrm{emb} = \pi_0\end{aligned}$$

from which completeness and stability as stated in Definition 6 clearly ensue : $[(a, b)] \sim (a, b)$ and if $a \leq b$ then $[(a, b)] = (a, b)$. Both facts follow from the properties of min and max. Thus $(Q, [\,\cdot\,])$ gives rise to a definable quotient.

We consider three examples in which A is taken to be the set $\mathbb{N}$, $\mathbb{N} \to \mathbb{N}$ and $(\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$ respectively :

$A = \mathbb{N}$

We use the standard ordering $\leq : \mathbb{N} \to \mathbb{N} \to \mathbf{Prop}$ and exploit that it is constructively total $\forall m, n \cdot m \leq n \vee n \leq m$ to define min and max.

$A = \mathbb{N} \to \mathbb{N}$

We use the lexicographic ordering $<, \leq : (\mathbb{N} \to \mathbb{N}) \to (\mathbb{N} \to \mathbb{N}) \to \mathbf{Prop}$

$$\begin{aligned} f < g &= \exists m : \mathbb{N} \cdot f\, m < g\, n \wedge \forall i < m \cdot f\, i = g\, i \\ f \leq g &= f < g \vee f = g \end{aligned}$$

While this order is not constructively total, in the sense that one cannot define a test to decide whether $f < g$, it is still possible to define min and max. For instance, the operator $\min : (\mathbb{N} \to \mathbb{N}) \to (\mathbb{N} \to \mathbb{N}) \to (\mathbb{N} \to \mathbb{N})$ can be defined as :

$$\begin{aligned} \min f\, g\, n = & \text{if } f\, n = g\, n \text{ then } f\, n \\ & \text{else} \\ & \text{let } i = \min\{j \leq n \mid f\, j \neq g\, j\} \\ & \text{in if } f\, i < g\, i \text{ then } f\, n \text{ else } g\, n \end{aligned}$$

Notice that both the definition of i and the test $f\, i < g\, i$ do not depend on n but only on f and g. Thus, in the case where f and g are different, $\min f\, g$ consistently returns the same function f or g, whichever is the smallest in lexicographical order. In the case where the two functions f and g are equal, then the second branch of the top level if. . . then. . . else. . . is never chosen.

$A = (\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$

The general idea to define the operator min is the same as in the case where $A = \mathbb{N} \to \mathbb{N}$. Let $\varphi : \mathbb{N} \to (\mathbb{N} \to \mathbb{N})$ be an enumeration of natural sequences such that any finite sequence $[x_0, \ldots, x_k]$ of natural numbers is the prefix of φ_i for some i in the sense that $\varphi_i\, 0 = x_0$, . . . , $\varphi_i\, k = x_k$ (see Appendix B for a definition of φ). We define :

$$\begin{aligned} \min f\, g\, u = & \text{if } f\, u = g\, u \text{ then } f\, u \\ & \text{else} \\ & \text{let } i = \min\{i : \mathbb{N} \mid f\, \varphi_i \neq g\, \varphi_i\} \\ & \text{in if } f\, \varphi_i < g\, \varphi_i \text{ then } f\, u \text{ else } g\, u. \end{aligned}$$

Notice as previously that both the definition of i and the test $f\, \varphi_i < g\, \varphi_i$ do not depend on u but only on f and g. Under the assumption that local continuity holds (see Definition 12 below), we know that if $f\,u \neq g\,u$ then there must exist some φ_i, sharing a long enough prefix with u, such that $f\,\varphi_i \neq g\,\varphi_i$. However, if one works in a Type Theory where type checking is decidable, local continuity needs to be derivable and not just admissible for the system to accept the above definition. As an alternative, one may postulate

$$\begin{aligned}
&\text{local\_continuity} :\\
&\forall f, g : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N}\\
&\;\to (\exists u : \mathbb{N} \to \mathbb{N}\,.\, f\,u \neq g\,u)\\
&\;\to (\exists n : \mathbb{N} \quad \forall v : \mathbb{N} \to \mathbb{N} \quad (\forall i \leq n\,.\, v_i = u_i \implies f\,v \neq g\,v))
\end{aligned}$$

Finite multisets As in the case of unordered pairs, the construction of a definable quotient over the setoid of multisets $(\mathrm{List}\,A, \sim)$ defined in Section 2.1 depends on the choice of A. We again require an order $A \to A \to \mathbf{Prop}$ to define the set of finite multisets of elements of A as

$$Q = \{(m, s) : \mathrm{List}\,A \mid \forall i, j : \mathrm{Fin}\;m \cdot i \leq j \implies s\,i \leq s\,j\}$$

and a sorting function $\mathrm{sort} : \mathrm{List}\,A \to \mathrm{List}\,A$ from which we define

$$[(m, s)] = (m, \mathrm{sort}\,s).$$

Notice that the function sort can be defined from the functions min and max : $A \to A \to A$ used in the previous example about unordered pairs. However, we use a more direct method in our exploration of the case where A is the set $\mathbb{N} \to \mathbb{N}$ of natural sequences. At first glance, it might seem counterintuitive that one can constructively sort sequences of infinite natural sequences and thus obtain a definable quotient of the setoids of multisets of natural sequences. As with unordered pairs, the first projection defines an embedding from Q to $\mathrm{List}\,A$ which clearly gives rise to a definable quotient.

$A = \mathbb{N} \to \mathbb{N}$

First we define a family of preorders $\{\leq_k\}_{k:\mathbb{N}}$ on sequences of natural numbers by requesting that $u \leq_k v$ if and only if the finite sequence $[u_0, \ldots, u_k]$ comes before the finite sequence $[v_0, \ldots, v_k]$ in the lexicographic order. Writing $u \leq_k v$ for $(\leq)\,k\,u\,v$:

$$\begin{aligned}
&- \leq_{-} - : \mathbb{N} \to (\mathbb{N} \to \mathbb{N}) \to (\mathbb{N} \to \mathbb{N}) \to \mathrm{Bool}\\
&u \leq_k v = u_i \leq v_i\\
&\qquad\qquad \text{where } i = \min\,\{i : \mathbb{N} \mid i > k \vee u_i \neq v_i\}\,.
\end{aligned}$$

Notice that if $u <_k v$ for some k then $u <_l v$ for all l greater than k.

Now, given a finite sequence of natural sequences $\varphi : \mathrm{Fin}\,m \to (\mathbb{N} \to \mathbb{N})$, we can order it using any algorithm

$$\mathrm{sort}_{m,k} : (\mathrm{Fin}\,m \to (\mathbb{N} \to \mathbb{N})) \to (\mathrm{Fin}\,m \to (\mathbb{N} \to \mathbb{N}))$$

which sorts m sequences according to the preorder $\leq_k$. We are then able to define :

$$\begin{aligned}[(m,\varphi)] &= (m,\psi)\\ &\quad \text{where } \psi\, i\, j = (\mathrm{sort}_{m,j}\ \varphi)\, i\, j,\end{aligned}$$

so that the finite sequence $[\psi\, 0,\ldots,\psi\,(m-1)]$ thus defined is the finite sequence $[\varphi\, 0,\ldots,\varphi\,(m-1)]$ ordered in lexicographic order. The key point justifying that claim is that

$$(\mathrm{sort}_{m,j}\ \varphi)\, i\, k = (\mathrm{sort}^*_m\ \varphi)\, i\, k \tag{1}$$

for all $i : \mathrm{Fin}\, m$ and all $k \leq j$ where $\mathrm{sort}^*_m\ \varphi$ is the finite sequence whose elements are the functions $\varphi\, i : \mathbb{N} \to \mathbb{N}$ ordered in full lexicographical order — we do not assume sort^*_m to be definable a priori although it is as a consequence of the definability of $\mathrm{sort}_{m,j}$. We omit further details of the proof, the intuition drawn from the case of unordered pairs above being more interesting.

$A = (\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$

This example will be the subject of a separate paper as its formalization is quite involved.

Finite sets

For types A over which equality is decidable, the set of finite subsets of A can easily be defined as a quotient of the setoid of finite sets $(\mathrm{List}\, A, \sim)$ considered in Section 2.1:

$$\begin{aligned}Q &= \{(m,s) : \mathrm{List}\, A \mid \forall i,j : \mathrm{Fin}\ m \cdot i \leq j \implies s\, i < s\, j\}\\ [(m,as)] &= \mathrm{nub}(\mathrm{sort}\ as)\end{aligned}$$

where $\mathrm{nub} : \mathrm{List}\ A \to \mathrm{List}\ A$ takes advantage of the decidability of equality on A to remove duplicates in a list.

Notice that there is no hope to define $[\,\cdot\,] : \mathrm{List}\ A \to Q$ when equality on A is not decidable, as $a = b$ is equivalent to $\mathrm{length}\,[(a,b)] = 1$, which is decidable. Still, in the case of $A = \mathbb{N} \to \mathbb{N}$ we can use a technique similar to the case for multisets to define a quotient — but, as for multisets with $A = (\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$, a convincing explanation will require a separate publication. However, it is not clear how to do this for higher order cases even when assuming continuity.

5 Undefinable quotients

However there are interesting setoid specifications for which it is impossible to construct a definable quotient in Type Theory. Examples include the real numbers and the partiality monad described in Section 2.1. To prove that these are indeed undefinable quotients, we first establish some properties of Type Theory in a classical metatheory. We write $\vdash a : A$ if $a : A$ is derivable in the Type Theory under consideration. In case that $\vdash P : \mathbf{Prop}$, we simply write $\vdash P$ to indicate that there is a proof p of P which is derivable, that is $\vdash p : P$.

Definition 8 (separable elements, discrete sets).

1. *Two elements a and b of a definable set are* separable*, written $a \nparallel b$, if there exists a definable test $P \colon A \to \mathrm{Bool}$ such that $\vdash P\,a \neq P\,b$.*
2. *A definable set A is* discrete *whenever $\vdash a, b : A$ and $\vdash a \neq b$ entails that a and b are separable.*

Proposition 9. *The set $\mathbb{N} \to \mathbb{N}$ is discrete.*

Proof. Assume $\vdash f, g \colon \mathbb{N} \to \mathbb{N}$ and $\vdash f \neq g$. By soundness, f and g must denote different functions and hence there is a natural number i such that $\vdash f\,i \neq g\,i$. Hence we can define $P\,h = h\,i \overset{?}{=} f\,i$ where $\overset{?}{=} \colon \mathbb{N} \to \mathbb{N} \to \mathrm{Bool}$ is a decision procedure for equality on $\mathbb{N}$.

Note that we have used classical reasoning in the proof of Proposition 9. However, we do not think it is necessary because it should be possible to extract the witness i from the proof that $f \neq g$.

Proposition 10. *Assume $e \colon A \to B$ is a definable split epi. If A is discrete then B is discrete.*

Proof. Let $\vdash s \colon B \to A$ such that $\vdash e \circ s = \mathrm{id}_B$ and let $\vdash b \neq b' \colon B$. Then $\vdash s\,b \neq s\,b'$ because s is a right inverse of e :

$$\begin{aligned}
&\vdash s\,b = s\,b' \to (e \circ s)\,b = (e \circ s)\,b' && \text{by congruence}\\
&\vdash s\,b = s\,b' \to \mathrm{id}_B\ b = \mathrm{id}_B\ b' && e \circ s = \mathrm{id}_B\\
&\vdash s\,b = s\,b' \to b = b' && \text{by definition of } \mathrm{id}_B\\
&\vdash s\,b = s\,b' \to \bot && \text{by modus ponens with } b = b' \to \bot.
\end{aligned}$$

Hence there exists $\vdash P \colon A \to \mathrm{Bool}$ such that $\vdash P\,(s\,b) \neq P\,(s\,b')$, because A is discrete, and $\vdash P' \colon B \to \mathrm{Bool}$ defined by $P' = P \circ s$ provably separates b and b'. Therefore B is discrete.

Proposition 11. *$\mathbb{R}_0$ is discrete.*

Proof. Left to the reader as it is essentially the same as the proof for Proposition 9.

To show that any set $\mathbb{R}$ which is a definable quotient of the setoid $(\mathbb{R}_0, \sim)$ given earlier in 2.1 is not discrete, we need

Definition 12 (local continuity). Local continuity *at type* $(\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$ *is the property that*

> *for all definable functions* $\varphi : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$,
> *for all definable sequences* $f : \mathbb{N} \to \mathbb{N}$,
> *there exists* $n : \mathbb{N}$ *such that*
> *for all definable sequences* $g : \mathbb{N} \to \mathbb{N}$ *satisfying* $(\forall i \le n, \vdash f\, i = g\, i)$,
> *we have that* $\vdash \varphi\, f = \varphi\, g$.

Local continuity expresses the fact that, to compute $\varphi\, f$, the reduction relation defining the operational semantics of Type Theory only inspects finitely many terms of the input sequence f. We have stated local continuity in its perhaps simplest form, at a particular type. However, we conjecture that it can be expressed and proved at all types. Whatever the case, it is easily shown that local continuity at type $(\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$ entails local continuity at some other types, in particular at type $(\mathbb{N} \to \mathbb{Q}) \to \mathrm{Bool}$, which we next use to show that no set is a definable quotient of the setoid $(\mathbb{R}_0, \sim)$ described in Section 2.1.

Lemma 13. *(local continuity for tests on rational sequences)*
In the presence of local continuity as in Definition 12, the following property holds :

> *for all definable functions* $\varphi\colon (\mathbb{N} \to \mathbb{Q}) \to \mathrm{Bool}$,
> *for all definable sequences* $f\colon \mathbb{N} \to \mathbb{Q}$,
> *there exists* $n : \mathbb{N}$ *such that*
> *for all definable sequences* $g\colon \mathbb{N} \to \mathbb{Q}$ *satisfying* $(\forall i \le n, \vdash f\, i = g\, i)$,
> *we have that* $\vdash \varphi\, f = \varphi\, g$.

Proof. Let $\eta\colon \mathbb{N} \to \mathbb{Q}$ be a definable bijection from $\mathbb{N}$ to $\mathbb{Q}$ and $\iota\colon \mathrm{Bool} \to \mathbb{N}$ a definable monomorphism, e.g. $\iota(\mathrm{true}) = 0$, $\iota(\mathrm{false}) = 1$. Let $\varphi\colon (\mathbb{N} \to \mathbb{Q}) \to \mathrm{Bool}$ and $f\colon \mathbb{N} \to \mathbb{Q}$ be as in the statement of Lemma 13. Define $\varphi'\colon (\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$ by $\varphi'\, f = \iota(\varphi(\eta\, f))$. By local continuity at type $(\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$, there exists $n : \mathbb{N}$ such that for all definable sequences $g'\colon \mathbb{N} \to \mathbb{N}$ satisfying $(\forall i \le n, \vdash (\eta^{-1} \circ f)\, i = g'\, i)$, we have that $\vdash \varphi'\, (\eta^{-1} \circ f) = \varphi'\, g'$. Now suppose that some definable function $g\colon (\mathbb{N} \to \mathbb{Q}) \to \mathrm{Bool}$ is such that $(\forall i \le n, \vdash f\, i = g\, i)$. Then we also have that $(\forall i \le n, \vdash (\eta^{-1} \circ f)\, i = (\eta^{-1} \circ g)\, i)$ and hence that $\vdash \varphi'\, (\eta^{-1} \circ f) = \varphi'\, (\eta^{-1} \circ g)$, that is $\vdash (\iota \circ \varphi)\, f = (\iota \circ \varphi)\, g$, by definition of φ'. Since ι is mono, we then have $\vdash \varphi\, f = \varphi\, g$, as expected.

Proposition 14. *In the presence of local continuity, no set R is a definable quotient of the setoid* $(\mathbb{R}_0, \sim)$.

Proof. Suppose for the sake of contradiction that $(R, [\,\cdot\,], \mathrm{sound})$ is a definable quotient of the setoid $(\mathbb{R}_0, \sim)$. The function $[\,\cdot\,] : \mathbb{R}_0 \to R$ is a split epi, as it has a right inverse emb, and hence by propositions 11 and 10, the set R is discrete. By exactness of the quotient, we have that $[\vec{0}] \ne [\vec{1}]$ where $\vec{0}$ and $\vec{1}$ are elements of $\mathbb{R}_0$

representing the Cauchy sequences $\lambda x.0$ and $\lambda x.1$, respectively. By discreteness of R, there exists a definable function $P : R \to \mathrm{Bool}$ such that $\vdash P[\vec{0}] \neq P[\vec{1}]$. It follows that the function $P' : \mathbb{R}_0 \to \mathrm{Bool}$ defined by $P'\,s = P[s]$ has the property that $\vdash P'\,\vec{0} \neq P'\,\vec{1}$ and that P' is closed under $\sim$. By local continuity at type $(\mathbb{N} \to \mathbb{Q}) \to \mathrm{Bool}$ (Lemma 13) and by proof irrelevance in the second component of the pairs in $\mathbb{R}_0$, there is a number n_P such that, for all definable sequences $f{:}\,\mathbb{N} \to \mathbb{Q}$,

$$(\forall i \leq n_P, \vdash f\,i = 0_{\mathbb{Q}}) \text{ entails } P'\,f = P'\,(\pi_0\,\vec{0}).$$

Define $g\,i$ = if $i \leq n$ then $0_{\mathbb{Q}}$ else $1_{\mathbb{Q}}$, such that $P'g = P'\vec{0}$ by local continuity. However $g \sim \vec{1}$ and hence $P'\,g = P'\,\vec{1}$, which contradicts $P'\,\vec{1} \neq P'\,\vec{0}$.

Using very similar reasoning it can be shown that $\mathbb{N}_\perp$ is not definable either.

It seems that all sets definable in ordinary Type Theory (using only the set formers Π, Σ, $=$, finite sets, W, see e.g. [11]) are discrete. This observation shows that the reals are not definable as an exact quotient in ordinary Type Theory while Proposition 14 shows that reals are not a definable quotient in any extension of ordinary Type Theory, as long as local continuity is admissible.

6 Conclusions

The main result of the present work is that the notion of a definable quotient in Intensional Type Theory is useful and doesn't require any extension of the theory. We hope that our formalisation of the notion and the examples help to popularize this notion among people using Type Theory. Some of the examples are maybe surprising, i.e. the possibility to define unordered pairs and multisets for 1st order function types, even though the order (and equality) of the elements are undecidable. Assuming an internal proof of local continuity, this can be even extended beyond 1st order. We also show that under the assumption of local continuity the set of real numbers cannot be defined by normalisation. This also extends to other examples such as the partiality monad. These natural examples strongly suggest that while the notion of a definable quotient is useful, we would also like to be able to use quotient sets which do not fall in this category. In the present work we have only covered the notion of a quotient by a propositional family. It seems interesting, especially in the context of higher dimensional Type Theory inspired by Voevodsky's proposal [14], to consider non-propositional quotients, e.g. the quotient of a set by a groupoid. An example for a definable quotient of this kind would be the quotient of a non-canonical notion of finite sets by isomorphism.

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 and James Chapman. Big-step normalisation. *Journal of Functional Programming*, 19(3-4):311–333, 2009.

3. Thorsten Altenkirch, Conor McBride, and Wouter Swierstra. Observational equality, now! In *PLPV '07: Proceedings of the 2007 workshop on Programming languages meets program verification*, pages 57–68, New York, NY, USA, 2007. ACM.
4. G. Barthe, V. Capretta, and O. Pons. Setoids in type theory. *Journal of Functional Programming*, 13(02):261–293, 2003.
5. Venanzio Capretta. General recursion via coinductive types. *Logical Methods in Computer Science*, 1(2):1–18, 2005.
6. Pierre Courtieu. Normalized types. In *Proceedings of CSL2001*, volume 2142 of *Lecture Notes in Computer Science*, 2001.
7. Martin Hofmann. *Extensional concepts in intensional type theory*. PhD thesis, School of Informatics., 1995.
8. M. Maietti. About effective quotients in constructive type theory. *Types for Proofs and Programs*, pages 166–178, 1999.
9. N.P. Mendler. Quotient types via coequalizers in Martin-Löf type theory. In *Proceedings of the Logical Frameworks Workshop*, pages 349–361, 1990.
10. Aleksey Nogin. Quotient types: A modular approach. In *ITU-T Recommendation H.324*, pages 263–280. Springer-Verlag, 2002.
11. B. Nordström, K. Petersson, and J.M. Smith. *Programming in Martin-Löf's type theory*, volume 85. Citeseer, 1990.
12. Ulf Norell. *Towards a practical programming language based on dependent type theory*. PhD thesis, Department of Computer Science and Engineering, Chalmers University of Technology, 2007.
13. Li Nuo. Representing numbers in Agda. Technical report, School of Computer Science, University of Nottingham, 2010. Summer internship report.
14. Vladimir Voevodsky. Univalent foundations of mathematics, 2011. webpage.

A Formalization in Agda

```
module Quotient where

open import Data.Product
open import Function

open import Relation.Binary.Core
open import Relation.Binary.PropositionalEquality
  hiding (isEquivalence)

open import ThomasProperties
```

Definition of setoids

```
record Setoid : Set₁ where
  infix 4 _≈_
  field
    Carrier : Set
    _≈_     : Carrier → Carrier → Set
    isEquivalence : IsEquivalence _≈_
  open IsEquivalence isEquivalence public

open Setoid renaming
  (refl to reflexive; sym to symmetric; trans to transitive)
```

Prequotients

```
record PreQu (S : Setoid) : Set₁ where
  constructor
    Q:_[ ]:_sound:_
  private
    A = Carrier S
    _~_ = _≈_ S
  field
    Q   : Set
    [_] : A → Q
    sound : ∀ {a b : A} → a ~ b → [ a ] = [ b ]
open PreQu renaming
  (Q to Q'; [_] to nf; sound to sound')
```

Quotients as prequotients with a dependent eliminator.

```
record Qu {S : Setoid} (PQ : PreQu S) : Set₁ where
  constructor
    qelim:_qelim-β:_
  private
    A     = Carrier S
    _~_   = _≈_ S
    Q     = Q' PQ
    [_]   = nf PQ
    sound : ∀ {a b : A} → (a ~ b) → [ a ] = [ b ]
    sound = sound' PQ
  field
    qelim : {B : Q → Set}
          → (f : (a : A) → B [ a ])
          → ((a b : A) → (p : a ~ b)
            → subst B (sound p) (f a) = f b)
          → (q : Q) → B q
    qelim-β : ∀ {B a f} q → qelim {B} f q [ a ] = f a
open Qu
```

Proof irrelevance of qelim

```
qelimIrr : {S : Setoid} {PQ : PreQu S} (x : Qu PQ)
  → ∀ {B a f q q'}
  → qelim x {B} f q (nf PQ a)
    = qelim x {B} f q' (nf PQ a)
qelimIrr x {B} {a} {f} {q} {q'} = (qelim-β x {B} {a} {f} q)
  ▶ ⟨ qelim-β x {B} {a} {f} q' ⟩
```

Exact quotients

```
record QuE {S : Setoid} {PQ : PreQu S} (QU : Qu PQ) : Set₁ where
  constructor
    exact:_
  private
    A  =  Carrier S
    _~_  =  _≈_ S
    [_]  =  nf PQ
  field
    exact : ∀ {a b : A} → [ a ] = [ b ] → a ~ b
open QuE
```

Quotients as prequotients with a non-dependent eliminator (lift). (As in Hofmann's PhD dissertation.)

```
record QuH {S : Setoid} (PQ : PreQu S) : Set₁ where
  constructor
    lift:_lift-β:_qind:_
  private
    A      =  Carrier S
    _~_    =  _≈_ S
    Q      =  Q' PQ
    [_]    =  nf PQ
  field
    lift     : {B : Set}
             → (f : A → B)
             → ((a b : A) → (a ~ b) → f a = f b)
             → Q → B
    lift-β : ∀ {B a f q} → lift {B} f q [ a ] = f a
    qind : (P : Q → Set)
             → (∀ x → (p p' : P x) → p = p')
             → (∀ a → P [ a ])
             → (∀ x → P x)
open QuH renaming (lift to lift'; lift-β to lift-β')
```

Definable quotients

```
record QuD {S : Setoid} (PQ : PreQu S) : Set₁ where
  constructor
    emb:_complete:_stable:_
  private
    A     =  Carrier S
    _~_   =  _≈_ S
    Q     =  Q' PQ
    [_]   =  nf PQ
  field
    emb    : Q → A
    complete : ∀ a → emb [ a ] ~ a
    stable : ∀ q → [ emb q ] ≡ q
open QuD
```

Relations between types of quotients:
Below, we show the following, where the arrow → means "gives rise to" :
QuH → Qu (Proposition 3 in the paper)
Qu → QuH (Reverse of Proposition 3)
QuD → QuE (A definable quotient is always exact)
QuD → Qu
QuD → QuH (Also a consequence of QuD → Qu and Qu → QuH)

```
QuH→Qu : {S : Setoid} → {PQ : PreQu S}
  → (QuH PQ) → (Qu PQ)
QuH→Qu {S} {Q: Q [ ] : [_] sound: sound}
  (lift: lift lift-β: β qind: qind) =
  record
    {qelim = λ {B} → qelim₁ {B}
    ;qelim-β = λ {B} {a} {f} → qelim-β₁ {B} a f
    }
  where
    A     = Carrier S
    _~_  = _≈_ S
      -- the dependent function f is made independent
    indep : {B : Q → Set} → ((a : A) → B [ a ]) → A → Σ Q B
    indep f a = [ a ], f a
    indep-β : {B : Q → Set}
           → (f : (a : A) → B [ a ])
           → (∀ a b → (p : a ~ b) → subst B (sound p) (f a) ≡ f b)
           → ∀ a a' → (a ~ a') → indep {B} f a ≡ indep f a'
    indep-β {B} f q a a' p = (cong_,_ [ a ] [ a' ] (sound p) (f a))
                              ▶ ((λ b → [ a' ], b) ⋆ (q a a' p))
    lift₀ : {B : Q → Set}
      → (f : (a : A) → (B [ a ]))
      → ((a a' : A) → (p : a ~ a')
      → subst B (sound p) (f a) ≡ f a')
      → Q → Σ Q B
    lift₀ f q = lift (indep f) (indep-β f q)
    qind₁ : {B : Q → Set}
      → (f : (a : A) → B [ a ])
      → (q : ∀ a b → (p : a ~ b) → subst B (sound p) (f a) ≡ f b)
      → ∀ (c : Q) → proj₁ (lift₀ f q c) ≡ c
    qind₁ {B} f q = qind P heredity base
      where
        f' : Q → Σ Q B
        f' = lift₀ f q
```

```
            P : Q → Set
            P c = proj₁ {_} {_} {Q} {B} (lift₀ f q c) = c
            heredity : ∀ x → (p p' : P x) → p = p'
            heredity x p p' = =-prfIrr ((lift₀ f q x) ₁) x p p'
            base : ∀ a → P [ a ]
            base a = proj₁ ⋆ β

        qelim₁ : {B : Q → Set}
              → (f : (a : A) → (B [ a ]))
              → (∀ a b → (p : a ~ b) → subst B (sound p) (f a) = f b)
              → (c : Q) → B c
        qelim₁ {B} f q c = subst B (qind₁ f q c)
          (proj₂ {_} {_} {Q} {B} (lift₀ f q c))
        qelim-β₁ : ∀ {B} a f q → qelim₁ {B} f q [ a ] = f a
        qelim-β₁ {B} a f q =
          (substIrr B (qind₁ f q [ a ])
            (cong-proj₁ {Q} {B} (lift₀ f q [ a ]) (indep f a) β)
            (proj₂ {_} {_} {Q} {B} (lift₀ f q [ a ]))) ▶
          (cong-proj₂ {Q} {B} (lift₀ f q [ a ]) (indep f a) β)


Qu→QuH : {S : Setoid} → {PQ : PreQu S}
  → (Qu PQ) → (QuH PQ)
Qu→QuH {S} {Q: Q [ ] : [_] sound: sound} (qelim: qelim qelim-β: β) =
  record
  {lift = λ {B} f s → qelim {λ _ → B} f (λ a b p
       → (subFix (sound p) B (f a)) ▶ (s a b p))
  ; lift-β = λ {B} {a'} {f} {s} → β {λ _ → B} {a'} {f} (λ a b p
       → (subFix (sound p) B (f a)) ▶ (s a b p))
  ; qind = λ P irr f
     → qelim {P} f (λ a b p → irr [ b ] (subst P (sound p) (f a)) (f b))
  }
  where
    subFix : ∀ {A : Set} {c d : A} (x : c = d) (B : Set) (p : B)
       → subst (λ _ → B) x p = p
    subFix refl _ _ = refl


QuD→QuE : {S : Setoid} {PQ : PreQu S} {QU : Qu PQ}
  → (QuD PQ) → (QuE QU)
QuD→QuE {S} {Q: Q [ ] : [_] sound: _}
  (emb: emb complete: complete stable: _) =
  record {exact = λ {a} {b} [ a ] = [ b ]
     → ⟨ complete a ⟩₀
       ▶₀ subst (λ x → x ~ b) (emb ⋆ ⟨ [ a ] = [ b ] ⟩) (complete b)
     }
     where
```

```
      A        =  Carrier S
      _~_      =  _≈_ S
      ⟨_⟩₀  :  Symmetric _~_
      ⟨_⟩₀  =  symmetric S
      _ ▶₀ _  :  Transitive _~_
      _ ▶₀ _  =  transitive S

QuD→Qu : {S : Setoid} → {PQ : PreQu S}
   → (QuD PQ) → (Qu PQ)
QuD→Qu {S} {Q: Q [ ] : [_] sound: sound}
   (emb: ⌜_⌝ complete: complete stable: stable)  =
   record
   {qelim  =  λ {B} f _ a → subst B (stable a) (f ⌜ a ⌝)
   ; qelim-β  =  λ {B} {a} {f} s
      → substIrr B (stable [ a ]) (sound (complete a)) (f ⌜ [ a ] ⌝)
      ▶ s _ _ (complete a)
   }

QuD→QuH : {S : Setoid} → {PQ : PreQu S}
   → (QuD PQ) → (QuH PQ)
QuD→QuH {S} {Q: Q [ ] : [_] sound: sound}
   (emb: ⌜_⌝ complete: complete stable: stable)  =
   record
   {lift   =  λ f _ q → f ⌜ q ⌝
   ; lift-β  =  λ {B} {a} {f} {s} → s ⌜ [ a ] ⌝ a (complete a)
   ; qind  =  λ P _ f → λ x → subst P (stable x) (f ⌜ x ⌝)
   }
```

Or

```
QuD→QuH' : {S : Setoid} → {PQ : PreQu S}
   → (QuD PQ) → (QuH PQ)
QuD→QuH' {S}  =  Qu→QuH ∘ QuD→Qu
```

B Definition of the enumeration φ of natural sequences

We define a family $\{\varphi_i : \mathbb{N} \to \mathbb{N}\}_{i:\mathbb{N}}$ of natural sequences with the property that any finite sequence $[x_0, \ldots, x_k]$ of natural numbers is a prefix of some φ_i, which we need in some of our examples. Furthermore, although not strictly needed here, these sequences are pairwise distinct. One idea to define such a family is to request that the sequences φ_{2i} at even indices are those starting with 0 while the others are in turn split into those starting with 1 (the $\varphi_{2i+1+2k}$, i.e. every other sequence of odd index) and the remaining sequences, starting with at least 2, etc. For each subfamily of sequences starting with the same prefix of length

φ_0	φ_1	φ_2	φ_3	φ_4	φ_5	φ_6	φ_7	φ_8	φ_9	φ_{10}	φ_{11}	φ_{12}	φ_{13}	φ_{14}	φ_{15}	φ_{16}	φ_{17}	φ_{18}	φ_{19}	φ_{20}
0	1	0	2	0	1	0	3	0	1	0	2	0	1	0	4	0	1	0	2	0
0	0	1	0	0	1	2	0	0	0	1	1	0	2	3	0	0	0	1	0	0
0	0	0	0	1	0	0	0	0	1	1	0	2	0	0	0	0	0	0	1	1
0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	1	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

Table 1. Prefixes of φ_0 to φ_{20}

n, we define the $(n+1)^{\text{th}}$ term in the same manner. Table B shows the prefix of the first 21 sequences.

Here are complete definitions written in the programming language Haskell. The first version is perhaps the easiest to understand but is given in terms of lists. The second version is a direct translation of the former in the language of functions directly. Finally, the third version is more direct and perhaps easier to read for some people.

B.1 Version 1

```
phi                  :: Int → (Int → Int)
phi i j              = sequences !! i !! j
sequences            = [0, 0 . .] : (tail (startWithAtLeast 0))
startWithAtLeast n = interleave (startWith n)
                               (startWithAtLeast (n + 1))
startWith n          = map (n:) sequences
interleave (x : xs) ys = x : interleave ys xs
```

B.2 Version 2

```
phi                  :: Int → (Int → Int)
phi                  = startWithAtLeast 0
startWithAtLeast n = interleave (startWith n)
                      (startWithAtLeast (n + 1))

startWith n i 0 = n
startWith n i (j + 1) = phi i j
```

```
interleave fs gs 0         = fs 0
interleave fs gs (i + 1) = interleave gs (fs ∘ (+1)) i
```

B.3 Version 3

```
phi :: Int → (Int → Int)
phi i 0 | even i =         0
        | odd i =          phi (i ‘div‘ 2) 0 + 1
phi i (j + 1) | even i = phi (i ‘div‘ 2) j
        | odd i =          phi (i ‘div‘ 2) (j + 1)
```