

1 Quotient definition of 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 we can use a pair of natural nubmers as raw type \mathbb{Z}_0 .

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

Mathematically, for two elements of $(n1, n2)$ and $(n3, n4)$, when $n1 + n4 = n3 + n2$, they represent the same result of subtraction, namely the same integer. By these definition, we can define for an equivalence relation for $\mathbb{N} \times \mathbb{N}$, when we want to use $\mathbb{N} \times \mathbb{N}$ to represent integers,

$$\begin{aligned} _ \sim _ & : \text{Rel } \mathbb{Z}_0 \text{ zero} \\ (x+, x-) \sim (y+, y-) & = (x+ \mathbb{N}+ y-) \equiv (y+ \mathbb{N}+ x-) \end{aligned}$$

With this equivalence relation, the set of all pairs of natural numbers are divided into equivalence classes. For each equivalence class we can choose a representative, namely choosing the normal form within \mathbb{Z}_0 . We call the function used to choose normal forms as normalisation. The one below is one of the canonical normalisation within the \mathbb{Z}_0 .

$$\begin{aligned} [-] & : \mathbb{Z}_0 \rightarrow \mathbb{Z}_0 \\ [m, 0] & = m, 0 \\ [0, \mathbb{N}.suc\ n] & = 0, \mathbb{N}.suc\ n \\ [\mathbb{N}.suc\ m, \mathbb{N}.suc\ n] & = [m, n] \end{aligned}$$

For example, $(3, 2)$ can be normalised to $(2, 1)$, then to $(1, 0)$.

To prove it is normalisation we should prove the result is still in the same equivalence classes. And the normal form should be unique.

$$\begin{aligned} \text{normal-ok} & : \forall a \rightarrow [a] \sim a \\ \text{normal-unique} & : \forall a\ b \rightarrow a \sim b \rightarrow [a] \equiv [b] \end{aligned}$$

As soon as we have the normalisation, we can use another more general way to define equivalence relation, namely just identify their normal form.

$$\begin{aligned} _ \sim _ & : \text{Rel } \mathbb{Z}_0 \text{ zero} \\ x \sim y & = [x] \equiv [y] \end{aligned}$$

The $[-]$ is an endomap in the set \mathbb{Z}_0 , and the resulting subset is actually isomorphic to the set of integers. However since we do not distinguish the types of the original form and the normal form, we lose the information that

it has been normalised. Therefore we can define the type of the result to be the set of integers.

$$\begin{aligned} [-] & : \mathbb{Z}_0 \rightarrow \mathbb{Z} \\ [m, 0] & = + m \\ [0, \mathbb{N}.suc\ n] & = -suc\ n \\ [\mathbb{N}.suc\ m, \mathbb{N}.suc\ n] & = [m, n] \end{aligned}$$

Then this is a retraction function for the normalisation function and we call it denormalisation function.

$$\begin{aligned} \ulcorner - \urcorner & : \mathbb{Z} \rightarrow \mathbb{Z}_0 \\ \ulcorner + n \urcorner & = n, 0 \\ \ulcorner -suc\ n \urcorner & = 0, \mathbb{N}.suc\ n \end{aligned}$$

Firstly we need to prove \sim is actually an equivalence relation.

Reflexivity

$$\begin{aligned} \text{zrefl} & : \text{Reflexive } _ \sim _ \\ \text{zrefl } \{x+, x-\} & = \text{refl} \end{aligned}$$

Symmetry

$$\begin{aligned} \text{zsym} & : \text{Symmetric } _ \sim _ \\ \text{zsym } \{x+, x-\} \{y+, y-\} & = \text{sym} \end{aligned}$$

Transitivity

$$\begin{aligned} _ > \sim < _ & : \text{Transitive } _ \sim _ \\ _ > \sim < _ \{x+, x-\} \{y+, y-\} \{z+, z-\} \ x=y\ y=z & = \\ \text{cancel-+-left } (y+ \mathbb{N}+ y-) (\mathbb{N}.\text{exchange}_1\ y+ y- \ x+ z- > \equiv < & \\ (y=z\ +=\ x=y) > \equiv < \mathbb{N}.\text{exchange}_2\ z+ y- \ y+ x- & \end{aligned}$$

\sim is *Equivalence relation*

$$\begin{aligned} _ \sim _ \text{isEquivalence} & : \text{IsEquivalence } _ \sim _ \\ _ \sim _ \text{isEquivalence} & = \mathbf{record} \\ \{ \text{refl} & = \text{zrefl} \\ \text{;sym} & = \text{zsym} \\ \text{;trans} & = _ > \sim < _ \\ \} \end{aligned}$$

Now we can prove that the \mathbb{Z}_0 and its equivalence relation \sim form a setoid.

(\mathbb{Z}_0, \sim) is a setoid

```

 $\mathbb{Z}$ -Setoid : Setoid _ _
 $\mathbb{Z}$ -Setoid = record
  { Carrier =  $\mathbb{Z}_0$ 
  ;  $\sim$  _ _ =  $\sim$  _ _
  ; isEquivalence = _  $\sim$  _ isEquivalence
  }

```

2 Rational numbers

The quotient definition of rational number is more natural to understand and the normalisation is also commonly used in regular mathematics. We just use one integer and one natural number to represent a rational number. The reason is because it is hard to exclude the invalid denominator if we use integers, so I choose the natural numbers to represent positive natural number which are one bigger.

```

data  $\mathbb{Q}_0$  : Set where
  _/suc_ : (n :  $\mathbb{Z}$ )  $\rightarrow$  (d :  $\mathbb{N}$ )  $\rightarrow$   $\mathbb{Q}_0$ 

```

and this is the equivalence relation for it

```

_  $\sim$  _ : Rel  $\mathbb{Q}_0$  zero
n1 /suc d1  $\sim$  n2 /suc d2 = n1  $\mathbb{Z}^*\mathbb{N}$  suc d2  $\equiv$  n2  $\mathbb{Z}^*\mathbb{N}$  suc d1

```

Reflexivity

```

qrefl : Reflexive _  $\sim$  _
qrefl {n /suc d} = refl

```

symmetry

```

qsym : Symmetric _  $\sim$  _
qsym {a /suc ad} {b /suc bd} = sym

```

transitivity

```

qtrans : Transitive _  $\sim$  _
qtrans {a /suc ad} {b /suc bd} {c /suc cd} a=b b=c with  $\mathbb{Z}.0?$  b
qtrans {a /suc ad} {o (+ 0) /suc bd} {c /suc cd} a=b b=c | yes refl =
   $\mathbb{Z}.solve0'$  (+ suc bd) {a} ( $\lambda$  ()) a=b 0  $\sim$ 
   $\mathbb{Z}.solve0'$  (+ suc bd) {c} ( $\lambda$  ())  $\langle$  b=c  $\rangle$ 

```

```

qtrans {a /suc ad} {b /suc bd} {c /suc cd} a=b b=c | no ¬p =
  ℤ.l-integrity (b ℤ* (+ suc bd)) (ℤ.nz* b (+ suc bd) ¬p (λ ())) (
    ℤ.*-exchange1 b (+ suc bd) a (+ suc cd) >≡<
    (ℤ.*-cong b=c a=b) >≡<
    ℤ.*-exchange2 c (+ suc bd) b (+ suc ad))

```

\sim is *Equivalence relation*

```

isEquivalenceQ0 : IsEquivalence _~_
isEquivalenceQ0 = record
  { refl = qrefl
  ; sym = qsym
  ; trans = qtrans
  }

```

Then it is natural to form the setoid

(Q_0, \sim) is a *setoid*

```

Q0setoid : Setoid _ _
Q0setoid = record {
  Carrier = Q0
  ; _~_ = _~_
  ; isEquivalence = isEquivalenceQ0
}

```

However these definition are just setoid and to form a quotient type, we need more structure. For definable quotient types, we need a representative of each equivalence class, we may have a set which is isomorphic to the set of equivalence classes, namely the normal form of the quotient type. Moreover, If we abstract the structure, we can prove some general properites for definable quotient types.

3 The general structure of definable quotient types

I will use the interfaces written by Thomas Amberree in this part. We need to first establish the quotient signature.

```

record QuSig (S : Setoid zero zero) : Set1 where
  field
    Q : Set

```

$$\begin{aligned}
& [_] : \text{Carrier } S \rightarrow Q \\
& \text{sound} : \forall \{a \ b : \text{Carrier } S\} \rightarrow (_ \approx _ S \ a \ b) \rightarrow [a] \equiv [b]
\end{aligned}$$

In this type signature, for certain setoid we have a type represent the set of the normal form, a normalisation function, and the proof that two elements in the same equivalence class normalised to the same form. With soundness, we can say normalisation is a function if we treat S as the set of equivalence classes.

However, there is no surjective requirements for the map in this signature. It means that the set of equivalence classes are not isomorphic to the set Q .

Actually, we can use the same type for $\text{Carrier } S$ and Q . For example, for Setoid \mathbb{Z}_0, \sim , we can build a quotient signature by giving \mathbb{Z}_0 and the endomap normalisation function.

Now, using the quotient signature if we can prove that any function of type $S \rightarrow B$ respects the equivalence relation, then we can lift it to be a function of type $Q \rightarrow B$. Of course we need to prove that it is lift function. With the lift function we have the first definition of quotient.

```

record Qu {S : Setoid zero zero} (QS : QuSig S) : Set1 where
  private S = Carrier S
   $\_ \sim \_ = \_ \approx \_ S$ 
   $Q \_ = Q \ QS$ 
   $[\_] = [\_] \ QS$ 
  sound :  $\forall \{a \ b : S_0\} \rightarrow (a \sim b) \rightarrow [a] \equiv [b]$ 
  sound = sound QS
field
  lift : {B : Q → Set}
    → (f : (a : S) → (B [a]))
    → ((a a' : S) → (p : a ~ a') → subst B (sound p) (f a) ≡ f a'))
    → (c : Q) → B c
  liftok :  $\forall \{B \ a \ f \ q\} \rightarrow \text{lift } \{B\} \ f \ q \ [a] \equiv f \ a$ 
  liftlrr :  $\forall \{B \ a \ f \ q \ q'\} \rightarrow \text{lift } \{B\} \ f \ q \ [a] \equiv \text{lift}$ 
     $\{B\} \ f \ q' \ [a]$ 

```

In my opinion the proof irrelevance of lift operations are unnecessary since lifeok implies it.

However, there can be more than one equivalence classes normalised to the same form. Therefore the normal form do not fully contain the information of quotient type. If we can prove the completeness **Nuo: I don't think this is completeness, it is only injective**, namely two elements normalised to the same form must be in the same equivalence class. they belong to the

same equivalence class. Hence the normalisation is injective from the set of equivalence classes to the set of normal forms.

```

record QuE {S : Setoid zero zero} {QS : QuSig S} (QU : Qu QS) : Set1 where
  private S = Carrier S
   $\overline{\_} \sim \overline{\_} = \overline{\_} \approx \overline{\_} S$ 
   $\overline{Q} \_ = \overline{Q} QS$ 
   $[-] = [-] QS$ 
  sound :  $\forall \{a\ b : S\} \rightarrow (a \sim b) \rightarrow [a] \equiv [b]$ 
  sound = sound QS
  field
    complete :  $\forall \{a\ b : S\} \rightarrow [a] \equiv [b] \rightarrow a \sim b$ 

```

Even if we prove the normalisation to be injective, we still not require it to be surjective. Then \overline{Q} may have some redundance. Therefore we need more efficient quotient type.

In **Nf** we have a embedding function used to choose a representative for each equivalence class. The proof of stability shows that **emb** is a section of normalisation function. Since all elements in \overline{Q} can be the result of the normalisation, it must be surjective. The proof **compl** shows that the representative is in the same equivalence class hence we can prove the completeness as well. In this definition of quotient, the set of all equivalence classes are in fact isomorphic to the set \overline{Q} .

```

record Nf {S : Setoid zero zero} (QS : QuSig S) : Set1 where
  private S = Carrier S
   $\overline{\_} \sim \overline{\_} = \overline{\_} \approx \overline{\_} S$ 
   $\overline{Q} \_ = \overline{Q} QS$ 
   $[-] = [-] QS$ 
  field
    emb :  $\overline{Q} \rightarrow S$ 
    compl :  $\forall a \rightarrow \text{emb } [a] \sim a$ 
    stable :  $\forall x \rightarrow [ \text{emb } x ] \equiv x$ 

```

We can easily establish the function transforming the **Nf** to **QuE**, since completeness can be derived from **compl**.

```

nf2quE : {S : Setoid zero zero} → {QS : QuSig S} → {QU : Qu QS} → (Nf QS) → (QuE QU)
nf2quE {S} {QS} {QU} nf =
  record {
    complete =  $\lambda \{a\} \{b\} [a] \equiv [b] \rightarrow$ 
       $\langle \text{compl } a \rangle \blacktriangleright \text{subst } (\lambda x \rightarrow x \sim b) (\text{emb } \star \langle [a] \equiv [b] \rangle) (\text{compl } b)$ 

```

```

}
where
private S = Carrier S
 $\overline{\_} \sim \_ = \overline{\_} \approx \_ S$ 
 $\overline{Q} \_ = \overline{Q} QS$ 
 $[\_] = [\_] QS$ 
emb = emb nf
compl = compl nf
 $\langle \_ \rangle : \text{Symmetric } \overline{\_} \sim \_$ 
 $\langle \_ \rangle = \text{symmetric } \overline{S}$ 
 $\_ \blacktriangleright \_ : \text{Transitive } \overline{\_} \sim \_$ 
 $\_ \blacktriangleright \_ = \text{transitive } \overline{S}$ 

```

We can also define non-dependent lift version of quotients. We need to prove quotient induction when we have uniqueness of proof for certain proposition dependent on Q.

Nuo: Why we need qind?

```

record QuH {S : Setoid zero zero} (QS : QuSig S) : Set1 where
private S = Carrier S
 $\overline{\_} \sim \_ = \overline{\_} \approx \_ S$ 
 $\overline{Q} \_ = \overline{Q} QS$ 
 $[\_] = [\_] QS$ 
sound :  $\forall \{a\ b : S\} \rightarrow (a \sim b) \rightarrow [a] \equiv [b]$ 
sound = sound QS
field
liftH : {B : Set}
 $\rightarrow (f : S \rightarrow B)$ 
 $\rightarrow ((a\ a' : S) \rightarrow (a \sim a') \rightarrow (f\ a) \equiv f\ a')$ 
 $\rightarrow Q \rightarrow B$ 
liftHok :  $\forall \{B\ a\ f\ q\} \rightarrow \text{liftH } \{B\} f\ q\ [a] \equiv f\ a$ 
-- quotient induction
qind : (P : Q → Set)
 $\rightarrow (\forall \{x\} \rightarrow (p\ p' : P\ x) \rightarrow p \equiv p')$ 
 $\rightarrow (\forall \{a\} \rightarrow P\ [a])$ 
 $\rightarrow (\forall \{x\} \rightarrow P\ x)$ 

```

If we have the normal form definition, we can lift the function easily.

```

nf2qu : {S : Setoid zero zero} → {QS : QuSig S} → (Nf QS) → (Qu QS)
nf2qu {S} {QS} nf =
record {

```

```

lift      = λ {B} f q a- → subst B (stable0 a-) (f (emb0 a-));
liftok    = λ {B} {a} {f} {q} →
substlrr B (stable [a]) (sound (compl a)) (f (emb [a])) ▶ q _ _ (compl a);
liftlrr = refl
}
where S      = Carrier S
       $\overline{\sim}$     =  $\overline{\approx}$  S
       $\overline{[-]}$   =  $\overline{[-]}$  QS
      sound  : ∀ {a b : S} → a ~ b → [a] ≡ [b]
      sound  = sound QS
      compl  = compl nf
      stable = stable nf
      emb    = emb nf

```

4 The properties of definable quotient types

Not only the predicate can be lifted, but also the operators can be lifted.

```

Op : ℕ → Set → Set
Op 0 = λ t → t
Op (suc n) = λ t → (t → Op n t)
record SetoidOp (St : Setoid zero zero) (n : ℕ) : Set1 where
  constructor §_§
  private
    S = Setoid.Carrier St
  field
    op : Op n S
record QuotientOp {St : Setoid zero zero}
  {Qs : QuSig St} (nf : Nf Qs) (n : ℕ) : Set1 where
  constructor §_§
  private
    Q = QuSig.Q Qs
  field
    op : Op n Q
auxf : {S Q : Set} (n : ℕ) ([_] : S → Q) (emb : Q → S) →
  Op n S → Op n Q
auxf zero [_] emb op = [op]
auxf (suc n) [_] emb op = λ x → auxf n [_] emb (op (emb x))
liftoptop : {S : Setoid zero zero} (n : ℕ) (Qs : QuSig S)

```



```

    (So : SetoidOp S n) (nf : Nf Qs) → QuotientOp nf n
  liftop n (Q, [-], sound) § op § (emb, compl, stable) = § auxf n
  [-] emb op §

```

We can lift operators of any order within the normal form definition of quotient type. According to this, lift the general properties are also possible.

5 The undefinable quotient types

All of integers, rational numbers and the congruence class of modulo have definable normal form or canonical form. However real numbers is not belonging to this group. It does not have normal forms. We can use cauchy sequences or signed digits to represent real numbers. They are obvious quotient sets isomorphic to the true real numbers, but we cannot use the interface introduced above. We can only postulate it in Agda. Moreover the equivalence relation is undecidable. For these kinds of quotient types, for which we do not have normal form, we call them axiomatised quotient types.

This could be a cauchy sequence used to represent real numbers (Simplified for readability),

```

record cauℝ : Set where
  field
    f : ℕ → ℚ
    p : (n : ℕ) → ∀ (m : ℕ) → (n < m) → |(f m) - (f n)| < (1 / 2 ↑ n)

```

It contains a function to generate the sequence of numbers and a proposition that the sequence converges by bounded rate.

For example we can embedding rational numbers easily,

```

emb : ℚ₀ → ℝ
emb q = f: (λ _ → q) p: λ n m n < m → (s ≤ s z ≤ n) resp (abscanc q)

```

But for irrational numbers we have to use different ways to generate the sequences. For square root, we can use Taylor series.

6 Conclusion

Here, we only talk about definable quotient types within Agda. The quotient is a setoid and the elements in an euivalence classes are not definitionally equal. However, if we axiomatize the type form of quotient type and let the Agda automatically normalise the carrier, then the definitional equality between different elements in same equivalence class will be present.