Numbers, quotients, and definability of reals

Li Nuo

November 27, 2012

1 Numbers

Since Agda is known as a proof assistant, the library of numbers is crucial. In such kind of proof assistants which are based on Martin-Löf type theory , we need to construct the type of numbers and the usual properties of them should be verifiable rather than axiomatic.

There are different ways of defining numbers, even though they are mathematically equivalent, they are technically different, which means the proving of theorems about the numbers varies. For example, integers can be defined by exploiting the isomorphism between $\mathbb{Z}$ and $\mathbb{N} + \mathbb{N}$:

```
data ℤ : Set where
  -[1+_]  : (n : ℕ) → ℤ  - -[1+ n ] stands for - (1 + n).
  +_      : (n : ℕ) → ℤ  - + n stands for n.
```

And this is exactly the definition in Agda standard library version 0.6. For each integer there is one unique representation so extra equivalence relation is not needed. However intuitively we lose the "special position" held by 0. Of course we can define three cases definition with distinct 0 constructor but too many cases are not ideal for proving. Using this definition we can define addition as

```
- An auxilliary operation:  subtraction of natural numbers

_⊖_ : ℕ → ℕ → ℤ
m       ⊖ ℕ.zero   = + m
ℕ.zero  ⊖ ℕ.suc n = -[1+ n ]
ℕ.suc m ⊖ ℕ.suc n = m ⊖ n

_+_ : ℤ → ℤ → ℤ
-[1+ m ] + -[1+ n ] = -[1+ suc (m ℕ+ n) ]
```

```
-[1+ m ] + +    n    = n ⊖ ℕ.suc m
+    m   + -[1+ n ] = m ⊖ ℕ.suc n
+    m   + +    n    = + (m ℕ+ n)
```

Alternatively we have another isomorphism between $\mathbb{Z}$ and ${}^{\mathbb{N}\times\mathbb{N}}/_{\sim}$, namely constructing the set of integers from quotienting the set of $\mathbb{N}\times\mathbb{N}$ by the following equivalence relation :

$$(n_1, n_2) \sim (n_3, n_4) \iff n_1 + n_4 = n_3 + n_2 \tag{1}$$

From this observation we can define a setoid implementation for integers.

```
data ℤ₀ : Set where
  _,_ : ℕ → ℕ → ℤ₀

_∼_    : Rel ℤ₀ _
(x1 , x2) ∼ (y1 , y2)    = (x1 ℕ+ y2) ≡ (y1 ℕ+ x2)
```

Since this definition has only one case, we don't need to define or prove for multiple cases. For example, the common operations defined on $\mathbb{Z}_0({}^{\mathbb{N}\times\mathbb{N}}/_{\sim})$ has only one case which are simpler than the one for the previous definition,

```
_+₀_ : ℤ₀ → ℤ₀ → ℤ₀
(x1 , x2) +₀ (y1 , y2) = (x1 ℕ+ y1) , (x2 ℕ+ y2)
```

The elegant definition leads to elegant proofs of the properties of integers. For example, we can easily prove the distributivity laws for it.

```
- distʳ :  _*_ DistributesOverʳ _+_
- distʳ (a , b) (c , d) (e , f) = ℕ.dist-lemʳ a b c d e f +=
- ⟨ ℕ.dist-lemʳ b a c d e f ⟩
```

The right distributivity of multiplication over addition can be proved simply by proving something about natural numbers. This is because the definition of setoid integer is to represent integers using natural numbers, the operations is defined from the operations for natural numbers and finally the equality is an equation about natural numbers. That means all these properties are derivable. In fact, we can prove everything even simpler by using the automatic ring solver

for natural numbers. The right distributivity for the two-case integers which is the library is much more cumbersome

```
distrib^r : _*_ DistributesOver^r _+_

distrib^r (+ zero) y z
  rewrite proj₂ *-zero | y |
        | proj₂ *-zero | z |
        | proj₂ *-zero | y + z |
        = refl

distrib^r x (+ zero) z
  rewrite proj₁ +-identity z
        | proj₁ +-identity (sign z S* sign x ◁ | z | ℕ* | x |)
        = refl

distrib^r x y (+ zero)
  rewrite proj₂ +-identity y
        | proj₂ +-identity (sign y S* sign x ◁ | y | ℕ* | x |)
        = refl

distrib^r -[1+ a ] -[1+ b ] -[1+ c ] = cong +_ $
  solve 3 (λ a b c → (con 2 :+ b :+ c) :* (con 1 :+ a)
                   := (con 1 :+ b) :* (con 1 :+ a) :+
                      (con 1 :+ c) :* (con 1 :+ a))
          refl a b c

distrib^r (+ suc a) (+ suc b) (+ suc c) = cong +_ $
  solve 3 (λ a b c → (con 1 :+ b :+ (con 1 :+ c)) :* (con 1 :+ a)
                   := (con 1 :+ b) :* (con 1 :+ a) :+
                      (con 1 :+ c) :* (con 1 :+ a))
        refl a b c

distrib^r -[1+ a ] (+ suc b) (+ suc c) = cong -[1+_] $
  solve 3 (λ a b c → a :+ (b :+ (con 1 :+ c)) :* (con 1 :+ a)
                    := (con 1 :+ b) :* (con 1 :+ a) :+
                       (a :+ c :* (con 1 :+ a)))
         refl a b c

distrib^r (+ suc a) -[1+ b ] -[1+ c ] = cong -[1+_] $
  solve 3 (λ a b c → a :+ (con 1 :+ a :+ (b :+ c) :* (con 1 :+ a))
                   := (con 1 :+ b) :* (con 1 :+ a) :+
                      (a :+ c :* (con 1 :+ a)))
```

```
         refl a b c

distrib^r -[1+ a ] -[1+ b ] (+ suc c) = distrib-lemma a b c

distrib^r -[1+ a ] (+ suc b) -[1+ c ] = distrib-lemma a c b

distrib^r (+ suc a) -[1+ b ] (+ suc c)
  rewrite +-⊖-left-cancel a (c ℕ* suc a) (b ℕ* suc a)
  with b ≤? c
... | yes b≤c
  rewrite ⊖-≥ b≤c
        | +-comm (- (+ (a ℕ+ b ℕ* suc a))) (+ (a ℕ+ c ℕ* suc a))
        | ⊖-≥ (b≤c *-mono ≤-refl {x = suc a})
        | ℕ.*-distrib-∸^r (suc a) c b
        | +_◁ (c ℕ* suc a ∸ b ℕ* suc a)
        = refl
... | no b≰c
  rewrite sign-⊖-≱ b≰c
        | |⊖|-≱ b≰c
        | -_◁ ((b ∸ c) ℕ* suc a)
        | ⊖-≱ (b≰c ∘ ℕ.cancel-*-right-≤ b c a)
        | ℕ.*-distrib-∸^r (suc a) b c
        = refl

distrib^r (+ suc c) (+ suc a) -[1+ b ]
  rewrite +-⊖-left-cancel c (a ℕ* suc c) (b ℕ* suc c)
  with b ≤? a
... | yes b≤a
  rewrite ⊖-≥ b≤a
        | ⊖-≥ (b≤a *-mono ≤-refl {x = suc c})
        | +_◁ ((a ∸ b) ℕ* suc c)
        | ℕ.*-distrib-∸^r (suc c) a b
        = refl
... | no b≰a
  rewrite sign-⊖-≱ b≰a
        | |⊖|-≱ b≰a
        | ⊖-≱ (b≰a ∘ ℕ.cancel-*-right-≤ b a c)
        | -_◁ ((b ∸ a) ℕ* suc c)
        | ℕ.*-distrib-∸^r (suc c) b a
        = refl
```

Back to addition for setoid integers, the operation is only valid when it respects the equivalence relation,

```
_respects_ : {A : Set} → Op₂ A → Rel A _ → Set
_⊛_ respects _≈_ = ∀ a b c d → a ≈ b → c ≈ d → (a ⊛ c) ≈ (b ⊛ d)
```

Given two pairs equal setoid integers $(x, x_1) \sim (x_2, x_3)$ and $(x_4, x_5) \sim (x_6, x_7)$, the goal just turns into some simple expression of natural numbers

$$x\mathbb{N} + x_4\mathbb{N} + (x_3\mathbb{N} + x_7) \equiv x_2\mathbb{N} + x_6\mathbb{N} + (x_1\mathbb{N} + x_5)$$

which can be automatically solved in Agda (the detail looks cumbersome).

Given any operation respects the equivalence relation, we can easily turns it into the corresponding operation for the set definition $\mathbb{Z}$ in a general way.

First, we need a normalization function to find the corresponding $z : \mathbb{Z}$ for any given $z_0 : \mathbb{Z}_0$.

```
[_]                : ℤ₀ → ℤ
[ m , 0 ]          = + m
[ 0 , suc n ]      = -[1+ n ]
[ suc m , suc n ]  = [ m , n ]
```

And the section of it

```
⌜ _ ⌝              : ℤ → ℤ₀
⌜ + n ⌝            = n , 0
⌜ -[1+ n ] ⌝       = 0 , suc n
```

Then the general lifting function is

```
lift₂  : (_⊛_ : Op₂ ℤ₀)
       → Op₂ ℤ
lift₂ _⊛_ a b = [ ⌜ a ⌝ ⊛ ⌜ b ⌝ ]
```

Since we can prove the operation respects the equivalence relation, the lifted operation should have the same properties, which means we can also lift the proofs of the theorems on setoid integers.

In fact, this kind of relationship between setoids and sets can be generalized as quotients. Given a setoid $(A, \sim)$, some set $Q : Set$ can be seen as the corresponding quotient type over this setoid, if we have a function $[\cdot] : A \to Q$ such that it fulfils certain set of properties (details in [1]).

For the set rational numbers, we could also define it using setoids from the construction of fractions and the equivalence relation on fractions.

For real numbers, we could use cauchy sequences of rational numbers to represent them. However, we still can not find a way to define the set of reals in current setting of Intensional Type Theory.

From the obervations above, there is a pattern between different kinds of numbers, namely we can create a setoid using already defined number sets to represent a new number set. This kind of relation better exploits the relation between the number sets such that we can prove theorems much simpler.

2 quotients

The quotient types enables redefining equality on types and it is also an extensional concept[**?**]. It is unavailable in Intensional Type Theory, usually we can using setoids instead. However not all types are represented using setoids which means we lose unification for them. We have to define everything twice one for sets and one for setoids. Altenkirch's setoid model solves the problem by representing all sets using setoids. It is possible because usual sets can be seen as setoids whose equality are propositional equality for given sets.

3 definability

References

[1] Thorsten Altenkirch, Thomas Anberrée, and Nuo Li. Definable Quotients in Type Theory. 2011.