Provable programming of algebra: arithmetic of fractions.

Sergei D. Meshveliani *

Program Systems Institute of Russian Academy of sciences,
Pereslavl-Zalessky, Russia. `http://botik.ru/PSI`

1

Adequate programming of mathematics.

- **Constructive mathematics, constructive logic for CA.**
- **A functional language with dependent types.** `A g d a` **(do not confuse with** `A d a` **!).**
- **Representing an algebraic domain depending on a dynamic parameter.**
- **Mathematical definitions and formal proofs are a part of a program, understood by the compiler and automatically checked *before* run-time.**
- **Termination proof is required.**
- **Exclusive "or" is applicable — for a decidable relation.**
- **Performance is not damaged.**
- **Full formal constructive proofs required or** `'postulate'`**.**
- **DoCon-A — 0.04.1 ready, 2.00 to be released in 2017.**

* This work is supported by the FANO project of the Russian Academy of Sciences, the project registration No AAAA-A16-116021760039-0.

2 About the `Agda` language

Example: `n≤5+2` is an identifier,

and in `n ≤ 5 + 2`, ≤ and + are operators.

`S` → `T` means the type of functions from `S` to `T`.

A statement `P` is expressed as a type family (`T`).

A proof for `P` is any value in `T`.

`P => Q` is expressed as `S` → `T`.

A proof of a theorem is any function (algorithm) that returns a value in the corresponding type.

3 Fraction field `Fraction R`

The two versions considered:

integral ring `R`, GCD-ring `R`.

The elements of `Q = Fraction R` are represented as

$$n/d, \quad n, d \in R, \quad d \neq 0.$$

Equality and arithmetic:

```
n/d =' n'/d'   =    n * d' ≈ n' * d

n/d *' n'/d'   =   (n * n') / (d * d')

n/d +' n'/d'   =   (n * d' + n' * d) / (d * d')

divide  n/d  n'/d' =    n/d *' d'/n'    for n' ≉ 0
```

For an *integral ring* `R`,
this domain satisfies the properties of a *field*.

4 Cancel by gcd !

Example 1:

$$\Sigma_{k=1}^{n} 1/k,$$

Example 2:

put to a matrix 10×10 $n/1$ with random $0 < n < 10$

and compute determinant by Gauss method.

`DoCon-A 2.00-pre` **applies eager cancelling by gcd,**

and uses a fraction representation with `Coprime num denom`**.**

But there are needed

– **GCDRing R**,

– **machine-checked proofs for** `Field (Fracton R)`

for the corresponding operations.

5 Naive definition and methods

```
record Prefraction : Set where
       constructor preFr

       field  num     :  C
              denom   :  C
              denom≉0 :  denom ≉ 0#



_='_ :  Rel Prefraction _

f =' g =  (num f * denom g) ≈ (num g * denom f)



_*'_ : Op₂ Prefraction

(preFr n d d≉0) *' (preFr n' d' d'≉0) =
                                      preFr (n * n') (d * d') dd'≉0
                                      where
                                      dd'≉0 = nz*nz d≉0 d'≉0


_+'_ : Op₂ Prefraction

(preFr n d d≉0) +' (preFr n' d' d'≉0) =
                             preFr ((n * d') + (n' * d)) (d * d')
                                              (nz*nz d≉0 d'≉0)
```

6 Level II of optimization

The division relation in a semigroup:

```
_|_  : Rel C _
x | y =  ∃ \q → x • q ≈ y
```

The coprimality notion for any monoid:

```
Coprime :  Rel C _
Coprime a b =  (c : C) → c | a → c | b → c | ε
```

GCD, GCD-ring:

```
record GCD (a b : C) :  Set
  where
  constructor gcd′
  field  proper   : C                -- proper gcd value
         divides₁ : proper | a
         divides₂ : proper | b
         greatest : ∀ {d} → (d | a) → (d | b) → (d | proper)

 gcd :  (a b : C) → GCD a b
```

The fraction notion for any GCD-ring:

```
record Fraction : Set where
       constructor fr′
       field  num     : C
              denom   : C
              denom≉0 : denom ≉ 0#
              coprime : Coprime num denom
```

Optimized arithmetic:

```
fraction :  (a b : C) → b ≉ 0# → Fraction


_*₁_ : Op₂ Fraction
(fr′ n d d≉0 _) *₁ (fr′ n’ d’ d’≉0 _) =
                                   fraction (n * n’) (d * d’) dd’≉0
                                   where
                                   dd’≉0 = nz*nz d≉0 d’≉0


_+₁_ :  Op₂ Fraction
(fr′ n d d≉0 _) +₁ (fr′ n’ d’ d’≉0 _) =
                        fraction ((n * d’) + (n’ * d)) (d * d’) dd’≉0
                        where
                        dd’≉0 = nz*nz d≉0 d’≉0
```

7 Level III of optimization

```
_*fr_ : Op₂ Fraction
(fr′ n₁ d₁ d₁≉0 co-n₁d₁) *fr (fr′ n₂ d₂ d₂≉0 co-n₂d₂) =
                              fraction (n₁' * n₂') (d₁' * d₂') d₁'d₂'≉0
     where
     struc1 = gcd n₁ d₂
     struc2 = gcd n₂ d₁
     open GCD struc1 using () renaming (quot₁ to n₁'; quot₂ to d₂';
                                         proper*quot₁≈a to gcd1*n₁'≈n₁;
                                         proper*quot₂≈b to gcd1*d₂'≈d₂  )
     ...

_+fr_ :  Op₂ Fraction
(fr′ n₁ d₁ d₁≉0 coprime-n₁d₁) +fr (fr′ n₂ d₂ d₂≉0 coprime-n₂d₂) =
                                                          h *fr f'+g'
     where
     strucN = gcd n₁ n₂
     strucD = gcd d₁ d₂
     ...
     h     = fr′ gN gD gD≉0 coprime-gNgD
     ...
     f'+g' = fraction ((n₁' * d₂') + (n₂' * d₁')) (d₁' * d₂') d₁'d₂'≉0
```

This method does not require a factorization domain.

8 Proofs

There are needed proofs for `Field (Fraction R)`**:**

congruence, associativity, commutativity for

`_*fr_` **and** `_+fr_`**,**

distributivity of `_*fr_` **respectively to** `_+fr_`**,**

the division property,

...

The approach is as follows.

(A) First these propertis are proved for `Prefraction`

for the naive methods `_+'_` **and** `_*'_`**.**

(B) Then proved are (I) and (II) below:

```
toPre : Fraction → Prefraction
toPre (fr′ n d d≉0 _) =  preFr n d d≉0
```

```
(f g : Fraction) → toPre (f *fr g) =' (toPre f) *' (toPre g)        (I)
```

```
(f g : Fraction) → toPre (f +fr g) =' (toPre f) +' (toPre g)        (II)
```

Finally, combining (A) and (B), it is easy to obtain a proof

for the above propertes for the methods `_*fr_`, `_+fr_`**.**

9 Simple examples of a formal proof

```
='trans :  Transitive _='_
='trans {preFr n1 d1 _} {preFr n2 d2 d2≉0} {preFr n3 d3 _}
                                     n1*d2≈n2*d1 n2*d3≈n3*d2 =  goal
  where
  e0 :  d2 * (n1 * d3) ≈ d2 * (n3 * d1)
  e0 = begin
         d2 * (n1 * d3)    ≈[ ≈sym $ *assoc d2 n1 d3 ]
         (d2 * n1) * d3    ≈[ *cong₁ $ *comm d2 n1 ]
         (n1 * d2) * d3    ≈[ *cong₁ n1*d2≈n2*d1 ]
         (n2 * d1) * d3    ≈[ *cong₁ $ *comm n2 d1 ]
         (d1 * n2) * d3    ≈[ *assoc d1 n2 d3 ]
         d1 * (n2 * d3)    ≈[ *cong₂ n2*d3≈n3*d2 ]
         d1 * (n3 * d2)    ≈[ *comm d1 _ ]
         (n3 * d2) * d1    ≈[ *cong₁ $ *comm n3 d2 ]
         (d2 * n3) * d1    ≈[ *assoc d2 n3 d1 ]
         d2 * (n3 * d1)
       □

  goal : n1 * d3 ≈ n3 * d1
  goal = cancelNonzeroLFactor d2 (n1 * d3) (n3 * d1) d2≉0 e0



cong-preFr :
     ∀ {n} {n'} {d} {d'} →
     (d≉0 : d ≉ 0#) → n ≈ n' → (d≈d' : d ≈ d') →
                                let d'≉0 = ≉cong₁ d≈d' d≉0
                                in
                                (preFr n d d≉0) =' (preFr n' d' d'≉0)
```

```
cong-preFr {n} {n’} {d} {d’} _ n≈n’ d≈d’ =  *cong n≈n’ (≈sym d≈d’)


*’cong :  _*’_ Preserves₂ _=’_ _=’_ _=’_

-- ∀ (f f’ g g’} → f =’ f’ → g =’ g’ → f *’ g =’ f’ * g’

*’cong {preFr n₁ d₁ d₁≉0} {preFr n₁’ d₁’ d₁’≉0}
       {preFr n₂ d₂ d₂≉0} {preFr n₂’ d₂’ d₂’≉0}
                          n₁d₁’≈n₁’d₁ n₂d₂’≈n₂’d₂ =

       -- goal :  n₁n₂/d₁d₂ =’ n₁’n₂’/d₁’d₂’
     begin
       (n₁ * n₂) * (d₁’ * d₂’)     ≈[ xy•zu≈xz•yu ]
       (n₁ * d₁’) * (n₂ * d₂’)     ≈[ *cong n₁d₁’≈n₁’d₁ n₂d₂’≈n₂’d₂ ]
       (n₁’ * d₁) * (n₂’ * d₂)     ≈[ xy•zu≈xz•yu ]
       (n₁’ * n₂’) * (d₁ * d₂)
     □
```

References

1. S. Lang. *Algebra*. Addison-Wesley Publishing Company, 1965.
2. A. A. Markov. *On constructive mathematics*,
   In Problems of the constructive direction in mathematics. Part 2.
   Constructive mathematical analysis, Collection of articles,
   Trudy Mat. Inst. Steklov., 67, Acad. Sci.
   USSR, Moscow–Leningrad, 1962, 8–14.
3. Per Martin-Löef. *Intuitionistic Type Theory*.
   Bibliopolis. ISBN 88-7088-105-9, 1984.
4. S. D. Meshveliani.
   On dependent types and intuitionism in programming mathematics,
   (In Russian).
   In electronic journal Program systems: theory and applications,
   2014, Vol. 5, No 3(21), pp. 27–50, `http://psta.psiras.ru/read/psta2014_3_27-50.pdf`
5. S. D. Meshveliani.
   Programming basic computer algebra in a language with dependent types,
   In electronic journal Program systems: theory and applications, 2015,
   6:4(27), pp. 313–340. (In Russian),
   `http://psta.psiras.ru/read/psta2015_4_313-340.pdf`
6. S. D. Meshveliani.
   Programming computer algebra with basing on constructive mathematics. Domains with factorization. In RUSSIAN.
   In electronic journal Program systems: theory and applications,
   2017, Vol 8, No 1, 2017, 44 pages,
   `http://psta.psiras.ru/read/psta2017_1_3-46.pdf`
7. S. D. Meshveliani.
   DoCon-A — a provable algebraic domain constructor,
   the source program and manual, 2016, Pereslavl-Zalessky.
   `http://http://www.botik.ru/pub/local/Mechveliani/docon-A/`
8. U. Norell, J. Chapman.
   Dependently Typed Programming in Agda,
   `http://www.cse.chalmers.se/~ulfn/papers/afp08/tutorial.pdf`