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 2.00.**

* 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` $\leq$ `5 + 2`, $\leq$ and $+$ are operators.

`S` $\rightarrow$ `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` $\rightarrow$ `T`.

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

3 Fraction field `Fraction R`

The three versions are considered:

integral ring `R`**, GCD-ring, Unique factorization ring.**

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 $\times$ 10 $n/1$ **with random** $0 < n < 10$
and compute determinant by Gauss method.

`DoCon-A 2.00` **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 Half-optimized arithmetic

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

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 Optimized arithmetic

Its correctness requires an unique factorization domain.

```
n₁/d₁ + n₂/d₂ =   (((n₁d₂' + n₂d₁')/'g₁) / (d₁'d₂'(g/'g₁))
                    where
                    g   = gcd d₁ d₂
                    d₁' = d₁ /' g
                    d₂' = d₂ /' g
                    g₁  = gcd (n₁d₂' + n₂d₁') g
```

```
_+fr_ :  Op₂ Fraction
(fr′ n₁ d₁ d₁≉0 coprime-n₁d₁) +fr (fr′ n₂ d₂ d₂≉0 coprime-n₂d₂) =

                                        fr′ s' ddg' ddg'≉0 coprime-s'-ddg'
   where
   -- (FSum)
   g  = gcd d₁ d₂;              d₁' = d₁ /' g;      d₂'  = d₂ /' g
   s  = n₁ * d₂' + n₂ * d₁';    g₁  = gcd s g;      s'   = s /' g₁
   g' = g /' g₁;                dd  = d₁' * d₂';    ddg' = dd * g'
```

The three main points to prove:

```
d₁'d₂'g' ≉ 0,

s' / (d₁'d₂'g')  =fr  (n₁d₂ + n₂d₁)/(d₁d₂)        (Corr)

Coprime  s'  (d₁'d₂'g').
```

8 Proof for $d_1'd_2'g' \not\approx 0$

$d_1 \not\approx 0$, $d_1'\ g \approx d_1$. Hence $d_1' \not\approx 0$, $g \not\approx 0$ ...

IntegralRing R, $d_1' \not\approx 0$, $d_2' \not\approx 0$, $g' \not\approx 0$. Hence $(d_1'\ d_2') * g' \not\approx 0$.

9 Proof for (Corr)

(Corr) means that +fr is the sum of fractions:

$$(s/'g_1)\ d_1d_2 \ \approx\ (n_1d_2 + n_2d_1)\ ((d_1/'g)\ (d_2/'g)\ g'),$$

that is it returns a fraction equal to the one returned by +'.

$$\text{Goal:} \quad (s/'g_1)\ d_1d_2 \ \approx\ (n_1d_2' + n_2d_1')\ g\ (d_1/'g)\ (d_2/'g)\ g'$$

The right hand side is

$s\ g\ (d_1/'g)\ (d_2/'g)\ g' \quad \approx$

$s\ d_1\ (d_2/'g)\ g' \quad \approx$

$s\ d_1\ (d_2/'(g'g_1))\ g' \quad \approx$

$s\ d_1d_2/'g_1 \quad \approx$

$(s/'g_1)\ d_1d_2$

10 Proof for coprimality

```
-- (FSum)
g  = gcd d₁ d₂,               d₁' = d₁ /' g,        d₂'  = d₂ /' g
s  = n₁ * d₂' + n₂ * d₁',     g₁  = gcd s g,        s'   = s /' g₁
g' = g /' g₁,                 dd  = d₁' * d₂',      ddg' = dd * g'

d₁ ≉ 0,  d₂ ≉ 0,  Coprime n₁ d₁,  Coprime n₂ d₂.

Goal:    Coprime  s'  ((d₁' * d₂') * g')
```

Its proof needs the structure of an Unique Factorization Ring for `R`. This proof is by the following steps.

- **Introduce the notions of primality, factorization to primes, factorization uniqueness.**
- **Prove the** `Prime|split` **lemma.**
- **Introduce the relation** `CoprimeByPrimes`**, and prove that for an UFt ring** `Coprime <==> CoprimeByPrimes`**.**
- **Prove the multiplicative property for** `CoprimeByPrimes`**, and thus reduce the goal to proving** `CoprimeByPrimes s' t` **for** `t =` d_1', d_2', g'.
- **Derive** `CoprimeByPrimes s'` d_1' **from** `Prime|split`**, and provide similar proofs for** d_2', g'.

Describe some details in this plan.

```
CoprimeByPrimes :  Rel C _
CoprimeByPrimes a b =  ∀ p → IsPrime p → p | a → p ∤ b
```

From `Prime|split` **it is easily derived the multiplicative property:**

```
coprimeByPrimesWithProduct :
 Prime|split → (∀ {a b c} → CoprimeByPrimes a b → CoprimeByPrimes a c →
                                            CoprimeByPrimes a (b * c))
```

`CoprimeByPrimes s' d₁'` **is proved by the function** `coprimeP-s'd₁'`.
It takes

$$\texttt{p : C,} \quad \texttt{prime-p : IsPrime p,} \quad \texttt{p|s' : p | s'}$$

and returns `p∤d₁' :  p ∤ d₁'`,

The latter negation is expressed as a function that maps any value `p|d₁' :  p | d₁'` **to the empty type** $\perp$.

So: there are given the values `p`, `prime-p`, `p|s'`, `p|d₁'`, **and the goal is to build** $\perp$ **out of this.**

This is done by using the equality

$$n_1 * d_2' \approx s - (n_2 * d_1') \qquad (1)$$

As `p` **divides** `s'`, **it also divides** $s = g_1 * s'$.

It is given `p|d₁'`. **Hence, by (1),** $p \mid (n_1 * d_2')$.

By `Prime|split`**, it holds** $p \mid n_1$ **or** $p \mid d_2'$.

In the first case, it hold $p \mid n_1$ **and** $p \mid d_1'$.
Then `p` **divides** $d_1 = g * d_1'$.

By $(\texttt{Coprime}\ n_1\ d_1)$, **it is derived that** `p` **is invertible.**
This contradicts to primality of `p`**, and produces the value** $\perp$.

In a similar style, it is proved all the rest for `Goal`.

11 Referring to an expensive algorithm is not expensive

We need to avoid factoring when performing fraction arithmetic.

The program for optimized fraction sum uses that

any common non-invertible factor for `a` **and** `b`
contains a prime which divides both `a` **and** `b`**.**

And its proof refers to the `factor` **algorithm.**

But the reference to '`factor`**' is only in the proof part of the client function.**

The program is so that factoring is not performed in the executable code for this client function.

12 Proofs for the *field* laws

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 proof scheme is as follows.

- First these propertis are proved for `(Prefraction, +', *')`.
- Then these propertis are proved for `(Fraction. +fr, *fr)`
  by using the above equality proofs for `+fr≗+'`, `*fr≗*'`.

For example, consider the proof for associativity of `+'`:

$$(n_1/d_1 \;+'\; n_2/d_2) \;+'\; n_3/d_3 \;\;=' \;\; n_1/d_1 \;+'\; (n_2/d_2 \;+'\; n_3/d_3) \qquad \text{<==>}$$

$$(n_1d_2 + n_2d_1)/(d_1d_2) \;+'\; n_3/d_3 \;\;=' \;\; n_1/d_1 \;+'\; (n_2d_3 + n_3d_2)/(d_2d_3) \qquad \text{<==>}$$

$$((n_1d_2 + n_2d_1)*d_3 + n_3d_1d_2)/(d_1d_2d_3) \;\;=' \;\; (n_1d_2d_3 + (n_2d_3 + n_3d_2)*d_1)/(d_1d_2d_3)$$

$$\text{<==>}$$

$$((n_1d_2 + n_2d_1)*d_3 + n_2(d_1d_2))*(d_1(d_2d_3)) \;\approx\; (n_1d_2d_3 + (n_2d_3 + n_3d_2)*d_1)*((d_1d_2)d_3)$$

...

13 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`