ПРОГРАММИРОВАНИЕ, 2020, № 2, с. 43–52

КОМПЬЮТЕРНАЯ АЛГЕБРА

УДК 004.421.6

О МАШИННОМ ДОКАЗАТЕЛЬСТВЕ ДЛЯ АРИФМЕТИКИ ДРОБЕЙ НАД КОЛЬЦОМ С НОД

© 2020 г. С. Д. Мешвелиани*

Институт программных систем им. А.К. Айламазяна РАН
152021 Ярославская обл., Переславский р-н, с. Веськово, ул. Петра Первого, д. 4 “а”, Россия

**E-mail: mechvel botik.ru*

Поступила в редакцию 31.08.2019 г.
После доработки 20.09.2019 г.
Принята к публикации 10.10.2019 г.

Представлена разработка доказательных программ арифметики дробей в общем случае области с операцией наибольшего общего делителя. Это есть малая часть библиотеки DoCon-A машинно-проверяемых программ для компьютерной алгебры, разработанной автором. В этой системе программы включают определения соответствующих математических понятий и доказательства главных свойств применяемых алгоритмов. Эти доказательства проверяются компилятором. Применяется чисто функциональный язык программирования Agda, который также предоставляет конструкцию зависимых типов. Описано построение формальных машинно-проверяемых доказательств для оптимизированного способа сложения дробей.

DOI: 10.31857/S0132347420020089

ВВЕДЕНИЕ

Вычисления в поле частных в общем случае кольца с НОД выполняют различные системы компьютерной алгебры. Но здесь мы рассматриваем только алгоритмы и программы, сопровождаемые машинно-проверяемыми доказательствами. Наша программа опирается на базовую библиотеку вычислительной алгебры DoCon-A [1, 2] доказательных программ, написанных автором на языке Agda [4, 5]. Этот язык основан на чистой функциональности, “ленивом” способе вычисления по умолчанию, поддержке обобщенного программирования (generic programming), аппарате зависимых типов. Приблизительно можно считать, что это язык Haskell, расширенный аппаратом зависимых типов. Зависимые типы позволяют адекватно описать алгебраическую область, зависящую от обычного значения ([1], Введение). Кроме того, этот аппарат позволяет вставлять в программу доказательства утверждений, и эти доказательства будут проверяться компилятором. Становится возможным описывать метод вычисления в программе так, как это делается в учебниках и научных статьях, вместе с доказательством правильности.

Проект DoCon-A является попыткой перенести обширную библиотеку программ вычислительной алгебры DoCon [3] с языка Haskell на математически значительно более адекватный язык.

Техника проверки доказательств компилятором (точнее – проверяльщиком типов, type checker) основана на изоморфизме Карри–Ховарда [6, 7]. Всякое утверждение S представляется подходящим типом T (зависящим от значений). Доказательство для S есть любая функция (завершающийся алгоритм), которая выдает любое значение v типа T. Проверяльщик типов проверяет отношение `v : T` посредством символьных вычислений с выражениями типов. Таким образом доказательство утверждения проверяется до начала компиляции в исполняемый код.

Недоказуемое высказывание соответствует пустому типу $\bot$ с пустым множеством значений. Отрицание высказывания соответствует функции, отображающей соответствующий высказыванию тип в пустой тип. Импликация выражена типом $S \to T$ всех функций из S в T, где S и T представляют соответствующие утверждения. Конъюнкция выражена произведением $S \times T$ типов, дизъюнкция выражена суммой $S \uplus T$ типов. Доказательство индукцией по построению данного выражается в виде рекурсивно заданной функции. Применение леммы выражается в виде вызова функции, представляющей доказательство леммы.

Пока ограничиваемся только конструктивными доказательствами [8, 9] (без использования принципа Маркова для доказательства завершаемости алгоритмов). В частности, всякий объект существует лишь как итог некоторого данного ал-

горитма, и должно быть дано доказательство завершаемости этого алгоритма. Мы часто пользуемся законом исключенного третьего – это возможно в тех случаях, когда приведен алгоритм разрешения соответствующего отношения.

В этой статье мы не можем дать более подробные объяснения по данной системе программирования. Объяснения и примеры содержатся в [4, 1] и в руководстве по библиотеке [2].

Выпуск `DoCon-A 2.02` библиотеки реализует задание иерархии общих классических структур (теорий) алгебры: `Полугруппа (Semigroup)`, `Группа (Group)`, `Кольцо (Ring)`, `Целостное кольцо (IntegralRing)`, `Евклидово кольцо (EuclideanRing)`, `Поле (Field)` и некоторых других. Из конструкторов алгебраических областей реализованы конструкторы поля частных для кольца с НОД и кольца остатков для евклидова кольца.

Часть проекта `DoCon-A`, посвященная разработке доказательной программы для общей арифметики дробей, опирается на реализацию этой иерархии понятий.

Обозначения и словоупотребление:

• ниже мы называем систему `DoCon-A` “библиотека”,

• имена НОД, gcd, GCD обозначают наибольший общий делитель,

• имя `Agda` иногда пишем кириллицей и склоняем по падежам.

1.1. О синтаксисе языка `Agda`

Имена операторов и отношений отделяются пробелами. Например, в строке

```
a+b≈0 : a + b ≈ 0#
```

программы `a+b≈0` есть имя значения, двоеточие отделяет имя значения от выражения типа, символы ‘+’ и ‘≈’ в выражении типа суть соответственно имя оператора сложения и имя двуместного отношения, `0#` есть имя нулевой постоянной. Вся эта строка означает объявление: значение `a+b≈0` имеет тип `a + b ≈ 0#` (и этот тип зависит от значений `a`, `b`, `0#`).

Знак подчерка в имени отношения или операции означает, что в синтаксисе программы во входном выражении на этой позиции ставится выражение аргумента этой операции.

Знак равенства ‘=’ – это определение идентификатора (присваивание) – часть синтаксиса программы. Равенство `_≈_` – это знак отношения равенства на некоторой области, это отношение задается для каждой области программой пользователя или стандартной библиотекой.

2. ПОЛЕ ЧАСТНЫХ

Поле частных над кольцом R (в программе – `Fraction R`) ([10] параграф 13) имеет смысл для целостного кольца (в программе – `Integral-Ring`), то есть для коммутативного кольца без делителей нуля (если $x * y = 0$, то $x = 0$ или $y = 0$). Здесь также рассматривается более сильное условие кольца с НОД – `GCDRing`).

Обозначим `_≈_` отношение равенства на `R`, `_+_`, `_*_` – действия сложения и умножения в области `R`.

Элементы области частных `Q = Fraction R` (дроби) определены как формальные записи

```
n/d,  n,d ∈ R,  d ≠ 0.
```

Равенство `_=’_` на `Q` определено как

```
n/d =’ n’/d’ = n * d’ ≈ n’ * d
```

Умножение, сложение и деление на `Q` определены как

```
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’
                                  для n’ ≉ 0.
```

Нетрудно доказать ([10] параграф 13), что для целостного кольца `R` операции, заданные таким образом, удовлетворяют закону *поля* (коммутативное кольцо, в котором каждый ненулевой элемент обратим по умножению).

Цель исследования:

выразить на языке `Agda` известные оптимизированные алгоритмы сложения и умножения дробей в случае кольца с НОД, сопровождая их машинно-проверяемыми доказательствами. Также рассматривается особый способ сложения дробей, в котором сокращение на НОД применяется более интенсивно.

Вышеописанные наивные способы вычисления действий `*’`, `+’` на практике в системах вычислительной алгебры не применяются из-за того, что во многих случаях вычисления в цикле приводят к быстрому росту размера знаменателя. Как пример худшего случая предложим вычисление отрезка гармонического ряда:

$$\Sigma_{k=1}^{n} 1/k \quad \text{для} \quad n = 4, 8, 12, 16, \ldots$$

В этой статье нет теоретических оценок стоимости вычисления, наша цель обозначена выше.

3. НАИВНЫЕ И ОПТИМИЗИРОВАННЫЕ АЛГОРИТМЫ АРИФМЕТИКИ ДРОБЕЙ

Объявление (на языке `Agda`)

```
record Prefraction : Set where
       constructor preFr
       field num : C
             denom : C
             denom≉0 : denom ≉ 0#
```

определяет, что дробь над целостным кольцом (с носителем, обозначенным C) состоит из числителя num, знаменателя denom и свидетельства (доказательства, данного на языке Agda) denom≠0 того, знаменатель не равен нулю.

Равенство на C обозначено _≈_, неравенство обозначено _≉_.

Отношение _=’_ равенства на типе Prefraction выражается кодом

```
_=’_ : Rel Prefraction _
f =’ g = (num f * denom g) ≈ (num g * denom f)
```

Заметим, что в правой части определения стоит выражение типа, зависящего от значений f и g, это способ выражать утверждения в виде типов.

В данной статье мы не обсуждаем действие умножения дробей, а сосредоточимся на более сложном действии, на их сложении. Сложение в наивной арифметике дробей программируется на типе Prefraction и записывается как

```
_+’_ : 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)
```

Здесь функция nz*nz применяется к значениям d≠0, d’≠0 (свидетельствам соответствующих неравенств) и возвращает свидетельство неравенства d * d’ ≉ 0. Эта функция выражает закон целостного кольца.

Полуоптимизированная арифметика опирается на понятия делимости и взаимной простоты. Отношение делимости определяется в программе для произвольной полугруппы в виде объявления типа

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

Здесь квантор существования имеет конструктивный смысл. Именно, всякое значение типа x|y является парой (q, eq), где eq свидетельство для равенства x • q ≈ y.

Свойство взаимной простоты определяется для произвольного моноида объявлением

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

–“для любого c (если c делит a и c делит b, то c обратим – делит единицу ϵ)”.

Понятие дроби выражается объявлением *записи* (record)

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

Оно включает требование coprime взаимной простоты числителя и знаменателя. Назовем это представление сокращенным. Равенство _=fr_ на типе Fraction такое же, как задано выше на типе Prefraction. Чтобы получить дробь в сокращенном виде применяется функция gcd и сокращение дроби на полученное значение. Это требует наличия структуры кольца с НОД (GCDRing) для кольца R, то есть такого целостного кольца, на котором заданы функция (алгоритм)

```
gcd : (a b : C) → GCD a b,
```

и функция _|?_ разрешения отношения _|_ деления. При этом понятие НОД определяется в виде объявления записи

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

Последние три поля записи представляют условия, которым значение proper должно удовлетворять. Например, условие greatest значит “для любого d (если d делит a и делит b, то d делит proper)”. Условия делимости включают в себя значения дополняющих множителей. Например, значение $\mathtt{divides}_1$ имеет вид пары (q_1, eq_1), где eq_1 есть свидетельство равенства $\mathtt{proper} * q_1 \approx a$.

Далее, функция

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

принимает значения для числителя и знаменателя и свидетельство неравенства нулю знаменателя и выдает соответствующую дробь в сокращенном виде. Она применяет вычисление НОД и деление нацело в R. Также эта функция строит доказательство взаимной простоты полученных после сокращения числителя и знаменателя. В этой разновидности арифметики дробей сумма дробей вычисляется функцией

```
_+1_ : Op2 Fraction
(fr' n1 d1 d1≠0) +1 (fr' n2 d2 d2≠0_) =
     fraction (n1 * d2 + n2 * d1) (d1 * d2) d1d2≠0
     where
     d1d2≠0 = nz*nz d1≠0 d2≠0
```

– сложить наивным способом, потом сократить на НОД. Назовем этот способ полуоптимизированным.

Оптимизированный (как мы его условно называем) способ сложения дробей применяет сокращение на НОД на ранних стадиях.

Здесь и ниже мы используем следующие обозначения, связанные со структурой GCD. Функция gcd возвращает запись GCD, из которой извлекаются некоторые готовые значения помимо объявленных в полях записи. Например, вычисление gcd d_1 d_2 для знаменателей дает запись, для которой собственно значение НОД переименовано в g, частное d_1/g переименовано в d_1', d_2/g переименовано в d_2'.

Оптимизированный способ сложения дробей n_1/d_1 и n_2/d_2 выражается через следующие предварительные вычисления:

```
g = gcd d1 d2;
d1' = d1 /' g;                d2' = d2 /' g
s = n1 * d2' + n2 * d1'; g1 = gcd s g
s' = s /' g1
```

(в наших обозначениях и в программе знак умножения пишется явно). Здесь /' обозначает деление нацело в области R в случае, когда доказано отношение делимости для аргументов. Здесь и ниже символ /' дан как комментарий. В программе вместо этого действия готовые значения дополняющих множителей d_1', d_2', s', g' извлекаются из записи итога применения функции gcd, они накапливаются заодно в ходе вычисления НОД. Оптимизированный способ взят, с переобозначениями, из книги [11], параграф 4.5.1, первая половина страницы 355. Буквально способ вычисления в [11] в наших обозначениях выражается формулой

```
s' / (d1'*(d2/'g1))
```

В программе он для удобства доказательства изменен к формуле

```
s' / ((d1'*(d2/'g)) * (g/'g1))
```

Дроби получаются равные, а стоимость вычисления существенно не отличается. В целом, программируется способ

```
(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'
```

И в этих обозначениях функция сложения задается как

```
_+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'
```

Здесь `fr'` конструктор (тег) дроби, `s'` и `ddg'` полученные числитель и знаменатель. Значения `ddg'`$\not\simeq$`0` и `coprime-s'-ddg'` суть соответственно свидетельства неравенства нулю знаменателя и взаимной простоты пары (`s'`, `ddg'`).

Способ из [11] дан для области целых чисел. Здесь же он применяется к произвольному кольцу с НОД. Законность и способ такого обобщения доказываются ниже. В [11] пишется о начальной проверке часто встречающегося условия взаимной простоты двух знаменателей. Для него сумма дробей вычисляется легче. Но мы эту проверку пропускаем потому, что в случае обратимости g вычисления по формулам (`FSum`) все-равно сильно удешевляются.

Этот способ на примере гармонического ряда (Раздел 2) показывает большое убыстрение в сравнении с полуоптимизированным способом $+_1$. Для `n = 16000` он в 500 раз быстрее, чем $+_1$, и в 8 раз быстрее, чем встроенная арифметика рациональных чисел библиотеки Glasgow Haskell - 7.10.2. Такое быстродействие можно объяснить на интуитивном уровне тем, что при формировании числителя и конечном вычислении НОД в способе $+_1$ всякий лишний общий множитель в аргументах этих действий сильно увеличивает стоимость последних умножения и вычисления НОД. Конечно, каждый из трех вышеприведенных способов окажется быстрее двух других на некотором нарочно подобранном семействе примеров. Имеет смысл выводить оценки средней стоимости вычисления сложения дробей как функции размера числителей и знаменателей для способов $+_1$ и `+fr`. Но здесь мы рассматриваем другие вопросы.

Приведем некоторые соображения об оптимизации. В каких случаях описанный выше оптимизированный способ оправдывает свое название? В случае целых чисел сокращение на нетривиальный НОД операндов умножения приводит к укорачиванию двоичного кода операндов и к удешевлению умножения. И в среднем приводит к удешевлению последующих действий при построении числителя суммы. Не зря этот способ приводится в [11]. В случае области $K[x_1, ..., x_n]$ многочленов над полем K деление операндов на нетривиальный НОД уменьшает полную степень операндов. При прочих равных условиях уменьшение полной степени ведет к удешевлению умножения многочленов, и так далее. Еще очевиднее это в случае конечного поля K, ибо тогда стоимость действий над коэффициентами ограничена постоянной. Эти соображения показывают, что имеет смысл рассматривать оптимизированный способ сложения дробей в общем случае и строить машинно-проверяемое доказательство его правильности.

3.1. Доказательство правильности в части равенства

Во-первых, требуется построить формальное доказательство на языке `Agda` того, что оптимизированный способ действительно выдает сумму дробей. При этом наивный способ сложения (Раздел 3) есть определение сложения дробей. То есть для доказательства правильности оптимизированного способа необходимо доказать, что формальные выражения

`s'/ddg'` и $(n_1 * d_2 + n_2 * d_1) / (d_1 * d_2)$

представляют равные дроби. То есть доказать равенство

$$s' * (d_1 * d_2) \approx (n_1 * d_2 + n_2 * d_1) * (d_1' * d_2' * g') \qquad (\texttt{Corr1})$$

Но в программе наивное и оптимизированное сложения определены на разных типах (второй тип включает условие взаимной простоты). Для формализации доказательства рассматриваются два представления дроби: `Prefraction` и `Fraction`. Также рассматриваются два отношения равенства: `='` на типе `Prefraction` и `=fr` на типе `Fraction`. Для доказательства равенства (`Corr1`) рассматриваются три способа сложения дробей:

- наивный способ `+'` на типе `Prefraction`,
- полуоптимизированный способ $+_1$ на типе `Fraction`,
- оптимизированный способ `+fr` на типе `Fraction`.

Нужные доказательства переносятся с версии (`Prefraction`, `='`, `+'`) на версию (`=fr`, $+_1$), и далее, на (`=fr`, `+fr`). В этом подходе законы поля доказываются для трех версий арифметики дробей: для (`Prefraction`, `+'`, `*'`), затем для (`Fraction`, $+_1$, $*_1$), затем для (`Fraction`, `+fr`, `*fr`). Такой подход обусловлен следующими причинами.

- Первая версия работает в наибольшей общности.
- Вторая версия может оказаться более эффективной, чем третья для некоторых задач.
- Третья версия часто оказывается наиболее эффективной.
- Законы поля просто доказываются для первой версии. И остается доказать, что вторая и третья версии выдают итог равный (как дробь) итогу, выдаваемому первой версией. Таким способом законы поля формально доказываются для второй и третьей версий арифметики дробей.

Например, покажем, как сочетательное свойство сложения доказывается для этих трех версий арифметики дробей. Преобразование между двумя представлениями дроби выражается двумя функциями:

```
toPre : Fraction → Prefraction
toPre (fr' n d d≠0 _) = preFr n d d≠0
                                        - удалить условие взаимной простоты
fromPre : Prefraction → Fraction
fromPre (preFr n d d≠0) = fraction n d d≠0
                                                - привести к сокращенному виду
```

Функции

```
fromPre∘toPre : ∀{f} → fromPre (toPre f) =fr f
fromPre∘toPre {f} = fr-sym {f} {fromPre (toPre f)} (f=fr=fraction-f f)

toPre∘fromPre : ∀{f} → toPre (fromPre f) =' f
toPre∘fromPre {preFr n d d≠0} = <пропуск>
```

выражают доказательство того, что отображения `toPre` и `fromPre` являются взаимно обратными (и, следовательно, взаимно-однозначными). Функции

```
+₁via+' : (f g : Fraction) → (f +₁ g) =fr (fromPre (toPre f +' toPre g))
+₁via+' f g = =fr-refl {f +₁ g}

toPre+homo : (f g : Fraction) → toPre (f +₁ g) =' (toPre f +' toPre g)
toPre+homo = <тело функции пропущено>

fromPre+homo : (f g : Prefraction) →
                  fromPre (f +' g) =fr
                  (fromPre f +₁ fromPre g)
fromPre+homo = <тело функции пропущено>
```

представляют простое доказательство того, что отображения `toPre` и `fromPre` являются гомоморфизмами относительно операций $+_1$ и +'. Функция

```
+₁cong : _+₁_ Preserves₂ _=fr_ → _=fr_ → _=fr_
```

представляет доказательство того, что операция $+_1$ конгруэнтна, то есть согласована с равенством `=fr`. Это доказательство выводится из свойства конгруэнтности отображения `toPre` и из леммы $+_1$`via+'`.

Сочетательный закон для наивного сложения +' доказывается так. Для дробей n_1/d_1, n_2/d_2, n_3/d_3 свойство сочетательности выражается равенством

$$n * d' \approx n' * d, \qquad (2)$$

где n выражение для числителя дроби (n_1/d_1 +' n_2/d_2) +' n_3/d_3, и n' выражение для числителя дроби n_1/d_1 +' (n_2/d_2 +' n_3/d_3). Подобным образом составляются выражения для знаменателей d и d'. Левая и правая части равенства (2) суть многочленные выражения с целыми коэффициентами от переменных n_1, d_1, n_2, d_2, n_3, d_3. Доказательство равенства (2) состоит в приведении этих выражений к нормальному виду многочлена (раскрытие скобок, приведение подобных членов и так далее) с использованием законов коммутативного кольца. При этом каждый шаг такой нормализации записывается в виде шага доказательства на языке Agda для равенства ≈ в кольце R. Это делается автоматически по вызову функции специального доказывателя равенств (что важно, написанной на Агде – так же, как все в библиотеке Агды) из модуля `InCommutativeSemiring` библиотеки. Это сокращает большой участок кода формального доказательства до одного вызова библиотечной функции.

Следующая функция строит доказательство сочетательного закона для операции $+_1$ путем сведения его к доказательству этого закона для +'.

Поясним, как нужно читать нижеприведенное доказательство. Сначала надо прочесть присваивания, данные после ключевого слова `where`. Затем читать левую колонку выражений между `begin-fr` и `end-fr`, затем читать соответствую-

щую правую колонку выражений. Левая колонка представляет последовательность дробей, в которой каждая дробь f равна следующей в смысле отношения =fr, а правая часть против выражения f представляет применение функции, воплощающей доказательство этого равенства.

```
+₁assoc : FP=fr.Associative _+₁_
+₁assoc f g h =
begin-fr
  (f +₁ g) +₁ h                =fr[ +₁cong e0 h=from-h' ]
  from (f' +' g') +₁ (from h') =fr[ e1 ]
  from ((f' +' g') +' h')      =fr[ fromPre-cong (+'assoc f' g' h') ]
  from (f' +' g'+h')           =fr[ fromPre+homo f' (g' +' h') ]
  f'' +₁ (from (g' +' h'))     =fr[ +₁cong f''=f (fromPre+homo g' h') ]
  f +₁ (g'' +₁ h'')           =fr[ +₁cong₂ (+₁cong g''=g h''=h) ]
  f +₁ (g +₁ h)
end-fr
where
from = fromPre; f' = toPre f
g' = toPre g; h' = toPre h
f'' = from f'; g'' = from g'
h'' = from h'
f''=f = fromPre∘toPre {f}
g''=g = fromPre∘toPre {g}
h''=h = fromPre∘toPre {h}
h=from-h' = =fr-sym (fromPre∘toPre {h})

e0 : (f +₁ g) =fr from (f' +' g')
e0 =
 =fr-sym
 (begin-fr
   from (f' +' g') =fr[ fromPre+homo f' g' ]
   f'' +₁ g''      =fr[ +₁cong f''=f g''=g ]
  f +₁ g
 end-fr
 )
e1 : (from (f' +' g')) +₁ (from h') =fr  from ((f' +' g') +' h')
e1 = =fr-sym (fromPre+homo f'+g' h')
```

Доказательство каждого шага из правой колонки дано в виде конструкции =fr[...]. Это не особая конструкция языка, а вызов стандартной функции _=[_]_ трех аргументов (написанной на Агде). Также стандартными функциями являются префиксная функция begin_ и постфиксная _□ – они в программе переименованы соответственно в begin-fr_ и _end-fr. Вся конструкция begin-fr ... end-fr наглядно показывает вывод равенства с повторным применением закона транзитивности trans. Доказательство использует свидетельство +'assoc сочетательного закона для +', то, что fromPre является изоморфизмом относительно двух операций сложения дробей и что операция +' конгруэнтна по каждому аргументу (лемма +₁cong).

Теперь покажем перенесение доказательств для операции +₁ на доказательства для операции +fr – с полуоптимизированного способа на оптимизированный. Функция

```
+fr≐+₁ : ∀ f g → (f +fr g) =fr (f +₁ g)
+fr≐+₁ (fr' n₁ d₁ d₁≠0 coprime-n₁d₁) (fr' n₂ d₂ d₂≠0 coprime-n₂d₂) =
                                                    s'*den2≈num2*ddg'
  where
  sum2 = n₁ * d₂ + n₂ * d₁
  G    = proper (gcd sum2 d₁d₂)
  num2 = div sum2 G
  den2 = div d₁d₂ G
  s'*den2≈num2*ddg' : s' * den2≈num2 * ddg'
  …
```

представляет доказательство того, что эти два способа сложения дают равные дроби. Здесь обозначения взяты из определения операции +fr данного в формуле (FSum). Целевым равенством является

```
s' * den2≈num2 * ddg' (Corr2)
```

Четыре значения в этом равенстве суть некоторые рациональные выражения от значений n_1, d_1, n_2, d_2, g, g', g_1, G, n_1', d_1', n_2', d_2'. Чтобы упростить цель до равенства многочленных выражений обе части (Corr2) умножаются на значение Ggg_1 = G * (g * g_1). Получается равенство (Corr2'), не содержащее знаменателей. Равенство (Corr2') доказывается приведением его частей к одному и тому же нормальному виду многочлена, а также подстановкой равенств, полученных из формул (FSum) (и извлеченных из структур GCD), например, g * d_1' ≈ d, g_1 * g' ≈ g. Наконец, целевое равенство (Corr2) выводится из (Corr2') вызовом (cancelNonzeroLFactor Ggg_1). Функция cancelNonzeroLFactor представляет доказательство леммы:

В целостном кольце для каждого ненулевого элемента a *выполнено свойство сокращения*

```
∀b c (a * b ≈ a * c ==> b ≈ c).
```

Библиотека включает простое доказательство этого, вместе с сотней или двумя лемм, сопровождающих описание классических структур алгебры.

Далее, доказательство сочетательного закона для действия +fr составляется с использованием доказательства $+_1$assoc для действия $+_1$ и доказательства равенства +fr≐$+_1$ для двух способов сложения дробей. Это делается так же, как в выше описанном перенесении доказательства сочетательного закона с версии +' на версию $+_1$. Таким же образом другие доказательства переносятся с версии +' на $+_1$, и далее на +fr.

Рассмотрим теперь построение свидетельств необходимых для составления итоговой дроби. Эти свидетельства (конструктивные доказательства) составляются для утверждений

```
coprime : Coprime s' ddg'
d₁d₂≠0# : d₁ * d₂ ≠ 0#
g≠0     : g ≠ 0#
ddg'≠0  : (d₁ * d₂) * g' ≠ 0#
g'≠0    : g' ≠ 0#
d₁'≠0   : d₁' ≠ 0#
```

и некоторых других. Первое из них обсуждается ниже. Остальные весьма просто получаются из свойства отсутствия делителей нуля в R и из леммы о делимости ∀x y → x | y → y ≠ 0# → x ≠ 0#.

3.2. Доказательство взаимной простоты для итога суммы дробей

Продолжаем использовать обозначения формул (FSum) Раздела 3. Осталась главная часть доказательства: составление свидетельства для свойства взаимной простоты значений s' и ddg':

```
Coprime s' ((d₁' * d₂') * g').
```

Из доказательства этого свойства следует, что в оптимизированном сложении нахождение последнего НОД можно пропустить. Доказательство основано на обобщенной лемме Евклида:

(LE) для любых элементов a,b,c кольца с НОД выполнено свойство

если Coprime a b и Coprime a c, то Coprime a (b * c).

Цель (Coprime s' ddg') доказывается по следующей схеме.

• Coprime d_1' n_1 (== Coprime n_1 d_1/g) следует из Coprime n_1 d_1.

• Coprime d_1' d_2' выполнено из-за того, что d_2' и d_1' суть дополняющие множители в структуре GCD d_1 d_2.

• По лемме Евклида, выполнено отношение Coprime d_1' n_1d_2'.

• Докажем теперь Coprime s d_1'. Предположим, что некоторый x делит s и делит d_1'. Из равенства s = n_1*d_2' + n_2*d_1' следует x | n_1d_2'.

Из выведенного выше отношения Coprime d_1' n_1d_2' следует, что x обратим.

Таким же способом доказывается утверждение Coprime s d_2'.

Далее, по лемме Евклида, имеем Coprime s d_1'd_2'.

Так как s' делит s, то выполнено отношение Coprime s' d_1'd_2'.

s' и g' суть дополняющие множители в GCD s g. Поэтому выполнено Coprime s' g'. Из леммы Евклида теперь следует Coprime s' ddg'.

По этой схеме в программе построено формальное доказательство в виде функции на языке Agda.

Ниже '|' обозначает отношение делимости в кольце R,

$x \sim y$ обозначает, что x и y ассоциированы, то есть делят друг друга.

3.3. Отступление: обобщенная лемма Евклида

В учебниках лемма Евклида формулируется так:

если простое число p делит a*b,

то p | a или p | b.

И доказательство использует или свойство однозначности разложения на множители, или свойство кольца главных идеалов. В нашем случае произвольного кольца с НОД такая формулировка леммы и такие условия не применимы. Поэтому в предыдущей версии программы добавлялось условие кольца с однозначным разложением на множители, а лемма Евклида формулировалась не для простого числа p, а для простого элемента этого кольца. При этом доказательство последнего свойства в оптимизированном сложении сильно усложнялось. Но позже оказалось, что известно изящное доказательство леммы Евклида в более общей формулировке, в условии кольца с НОД – как выражено выше в утверждении (**LE**). И это позволяет сильно упростить программу и повысить ее общность. Доказательство обобщенной леммы Евклида является, по-видимому, фольклором. Позже автор нашел доказательство в электронном сообщении на форуме любителей математики в компьютерной сети. Приведем его здесь, переписав его в более ясном виде.

Лемма GCD*. Для кольца с НОД R и для любых a, b, c из R выполнено отношение $\gcd(ac, bc) \sim c \cdot \gcd(a, b)$.

Доказательство.

В случае $c = 0$ отношение $\gcd(0, 0) \sim 0$ выполнено, так как из определения НОД (Раздел 3) следует, что $\gcd(0, 0) = 0$.

Рассмотрим случай $c \neq 0$. Делимость $c \cdot \gcd(a, b) \mid \gcd(ac, bc)$ сразу следует из того, что $c \cdot \gcd(a, b)$ делит ac и делит bc.

Осталось доказать $\gcd(ac, bc) \mid c \cdot \gcd(a, b)$. Так как $c \mid ac$, $c \mid bc$, то $c \mid \gcd(ac, bc)$, и $\gcd(ac, bc) = xc$ для некоторого x, и по свойству НОД выполнено $xc \mid ac$, $xc \mid bc$. То есть $xcy = ac$, $xcz = bc$ для некоторых y, z. Из последних двух равенств следуют равенства $c \cdot (xy - a) = 0 = c(xz - b)$. Раз $c \neq 0$, то по свойству целостного кольца выполнены равенства $xy - a = 0 = xz - b$. Следовательно, $x \mid a$, $x \mid b$, $x \mid \gcd(a, b)$, $xc \mid c \cdot \gcd(a, b)$. Подставляя сюда равенство $\gcd(ac, bc) = xc$, выведенное раньше, получаем требуемую делимость.

Доказательство леммы Евклида.

Даны отношения Coprime(a, b), Coprime(a, c). Выведем Coprime(a, bc).

Предположим, что x делит a и делит bc. Тогда $x \mid \gcd(ac, bc)$. По лемме GCD*, $x \mid c \cdot \gcd(a, b)$. Из условия $\gcd(a, b) \sim 1$ взаимной простоты следует $x \mid c$. Из этого и из условий $x \mid a$, Coprime(a, c) следует, что x обратим. Это доказывает лемму.

В библиотеке доказательство леммы Евклида формализовано на языке Agda в виде функции euclid в модуле OfGCD.agda.

Реализация алгоритма и доказательства: описанный метод сложения дробей воплощен в библиотеке DoCon-A 2.02 [2] (файлы source/Fraction/*.agda). Файл source/demoTest/FractionTest.agda содержит демонстрацию.

4. О ДРУГИХ РАБОТАХ ПО ДАННОМУ ПРЕДМЕТУ

Система Axiom [12] программирования научных вычислений известна с 1970-х годов. Ее язык Spad – это инструмент для обобщенного программирования, и библиотека Axiom реализует башню классических структур алгебры. В этой библиотеке конструктор Fraction поля частных принимает как аргумент целостное кольцо. И при наличии в области операции НОД применяется и арифметика дробей, основанная на сокращенном виде дроби. Но язык Spad не поддерживает ни зависимые типы, ни, вообще, средства доказательного программирования.

Система Coq [13] имеет большое множество библиотек, с включением машинно-проверяемых доказательств различных непростых теорем в математике. На странице сети https://github.com/math-comp/math-comp/blob/master/mathcomp/algebra/fraction.v дана доказательная программа для арифметики поля частных целостного кольца. Рассматривается только наивная арифметика дробей, которая проста для доказательств в теории.

Алгебраическая часть библиотеки системы Lean [14] содержит выражение конструкции локального кольца:

https://github.com/leanprover-community/mathlib/

tree/master/src/ringt_heory,

модуль localization.lean. Это обобщение понятия поля частных. Но не видно никакой оптимизации вычисления суммы дробей. Очевидно, в системах Coq и Lean можно реализовать такой доказательный алгоритм. Но это пока не сделано, так как эти системы направлены в первую очередь на теоретические доказательства в математике.

5. ЗАКЛЮЧЕНИЕ

Описанный выше опыт с доказательной программой арифметики дробей в общем случае показывает, что язык Agda и его реализация позволяют адекватно выразить эту алгебраическую конструкцию, вместе с необходимым основанием в виде библиотеки классических категорий алгебры, и провести необходимые полные формальные доказательства для оптимизированного способа сложения дробей.

Также библиотека DoCon-A имеет то значение, что она является первой общей и сравнительно обширной библиотекой вычислительной алгебры для языка Agda.

БЛАГОДАРНОСТИ

Автор благодарен неизвестному рецензенту, который указал на существование обобщенной леммы Евклида.

Исследование поддержано Министерством науки и высшего образования РФ, исследовательский проект No AAAA-A19-119020690043-9.

СПИСОК ЛИТЕРАТУРЫ

1. *Мешвелиани С.Д.* О зависимых типах и интуиционизме в программировании математики. В электронном журнале Программные системы: теория и приложения. 2014. Т. 5. Вып. 3. С. 27–50. http://psta.psiras.ru/read/psta2014 3 2 7-50.pdf
2. *Мешвелиани С.Д.* DoCon-A. Библиотека доказательных программ компьютерной алгебры. Переславль-Залесский, 2015–2018. http://www.botik.ru/pub/local/Mechveliani/docon-A/
3. *Mechveliani S.D.* Computer algebra with Haskell: applying functional-ategorial-'lazy' programming. In Proceedings of International Workshop CAAP-2001, Ed. V.P. Gerdt. Dubna, Russia. C. 203–211.
4. *Norell U.* Dependently Typed Programming in Agda. AFP 2008: Advanced Functional Programming, Lecture Notes in Computer Science, vol. 5832, Springer, Berlin–Heidelberg. 2008. C. 230–266.
5. Agda. A proof assistant. A dependently typed functional programming language and its system. http://wiki.portal.chalmers.se/agda/pmwiki.php.
6. *Curry H.B., Feys R.* Combinatory Logic, vol I. Amsterdam, North-Holland, 1958. 417 c.
7. *Howard W.A.* The formulae-as-types notion of construction. To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism. Boston, Academic Press, 1980. C. 479–490.
8. *Марков А.А.* О конструктивной математике. Проблемы конструктивного направления в математике. 2. Конструктивный математический анализ, Сборник работ. Тр. МИАН СССР, 67. Изд-во АН СССР, М.–Л., 1962, С. 8–14.
9. *Martin-Loef, Per.* Intuitionistic type theory. Bibliopolis, ISBN 88-7088-105-9 (1984), 91 c.
10. *ван дер Варден Б.Л.* Алгебра. Наука, Москва, 1979, 624 с.
11. *Кнут Д.Е.* Искусство программирования для ЭВМ. Том 2. Мир, Москва, 1977. 55 печ. л.
12. *Jenks R.D., Sutor R.S. и др.* Axiom, the Scientific Computation System. Springer-Verlag, New York–Heidelberg–Berlin, 1992.
13. *Chlipala A.* Certified Programming with Dependent Types: A Pragmatic Introduction to the Coq Proof Assistant. MIT Press, 2013. http://adam.chlipala.net/cpdt/
14. *de Moura L., Kong S., Avigad J., van Doorn F., von Raumer J.* The Lean Theorem Prover. 25th International Conference on Automated Deduction (CADE-25), Berlin, Germany, 2015. https://leanprover.github.io/papers/system.pdf.