Конструктивная математика и зависимые типы в программировании алгебры

С. Д. Мешвелиани

Институт программных систем им. А.К.Айламазяна РАН, г. Переславль-Залесский

1 Цель: Адекватное программирование математики

Программа должна следовать изложению в духе учебников.

Построитель алгебраических областей (DoCon) воплощён на языке Haskell.

2 Что главное для научных вычислений желательно в языке программирования

Функции высшего порядка	**(H)**
Конструкторы типов, задаваемые пользователем	**(T)**
Обобщённое программирование (объекты, классы)	**(Gen)**
Зависимые типы (dependent types)	**(DT)**
Вычисление областей	**(CompDom)**
Проверяемые определения и доказательства	**(Log)**
Автоматическая проверка (verification)	**(Ver)**
Чистая функциональность	**(F)**
Ленивое вычисление	**(Lazy)**

C	
Lisp	**H F**
Refal, Flac	**F**
Mathematica Rule language	**F H (T Gen ?)**
Spad (Axiom)	**H T Gen**
Aldor (Axiom)	**H T Gen CompDom DT**
ML	**H T Gen**
Haskell	**H T Gen F Lazy**
Coq	**H T Gen CompDom DT Log Ver**
Cayenne	**H T Gen CompDom DT Log Ver F Lazy (not implemented)**
Agda [3]	**H T Gen CompDom DT Log Ver F Lazy**

3 О чистой функциональности

Пример программы на Хаскеле:

```
reverse ::  [a] ->  [a]
reverse xs =  rev [] xs   where
                          rev ys []       =  ys
                          rev ys (x : xs) =  rev (x : ys) xs


f ::  Int ->  [a] ->  ([a], [a])
f n xs =
         ( reverse xs,   reverse (take n xs) )
```

4 О динамически порождаемых областях

Пусть натуральные числа

$$1 \ < \ m_1 \ < \ ... \ < \ m_9 \ < \ 1000$$

вводятся во время исполнения программы, и пусть известно, что все они *простые* — кроме, быть может одного.

Для данной целочисленной матрицы M размера 20×20
и для каждого i программа строит проекцию

$$M \quad \text{в матрицу} \quad M_i \quad \text{над} \quad R_i = \mathbb{Z}/(m_i)$$

и выдаёт

$$det\ M_i, \quad i = 1, \ldots, \ 9.$$

Если R_i есть *поле*, то применить приведение к ступенчатому виду ...

Программа, использующая зависимые типы :

```
eucResidue :  (E : EuclideanRing) →  (g : EuclideanRing.Carrier E) →
                                                    CommutativeRing ⊎ Field
eucResidue E g =  case  prime? b  of \
                                  { (yes  prime-b) → inj₁ field-Res-b
                                  ; (no  ¬prime-b) → inj₂ commRing-Res-b
                                  }

det :  Matrix 20 20 ℤ →  (g : ℤ) →  EucResidue ℤ g

det M g =  let M' : Matrix 20 20 (EucResidue ℤ g)
               M' = projectMatrix g M
           in
           case   eucResidue  Integer2.euclideanRing  g
           of \
           { (inj₂ field)    →  detByGauss       field     M'
           ; (inj₁ commRing) →  detByExpandByRow commRing  M' }
```

5 Пример сильного применения зависимых типов (язык Agda)

```
open DecTotalOrder dto using (_≤_) renaming (Carrier to C)

data OrderedList :  List C →  Set      -- определение упорядоченности списка
     where
     nil    :  OrderedList []

     single :  (x : C) →  OrderedList (x :: [])

     prep2  :  (x y : C) →  (xs : List C) →  x ≤ y →
                           OrderedList (y :: xs) →  OrderedList (x :: y :: xs)
```

Пример:

свидетельство упорядоченности списка `(0 :: 3 :: 7 :: [])` выражается в виде *данного*

```
prep2  0  3  (7 :: [])  0≤3  (prep2 3 7 [] 3≤7  (single 7)).
```

Понятие сортировки:

```
record SortRes  (xs : List C) :  Set  where
       field
         list    :  List C
         ordered :  OrderedList list
         sameMS  :  (toMS xs) =ms (toMS list)

sort :  (xs : List C) →  SortRes xs   -- функция сортировки
sort xs =  ...
```

Программирование с зависимыми типами: Coq, Agda [3].

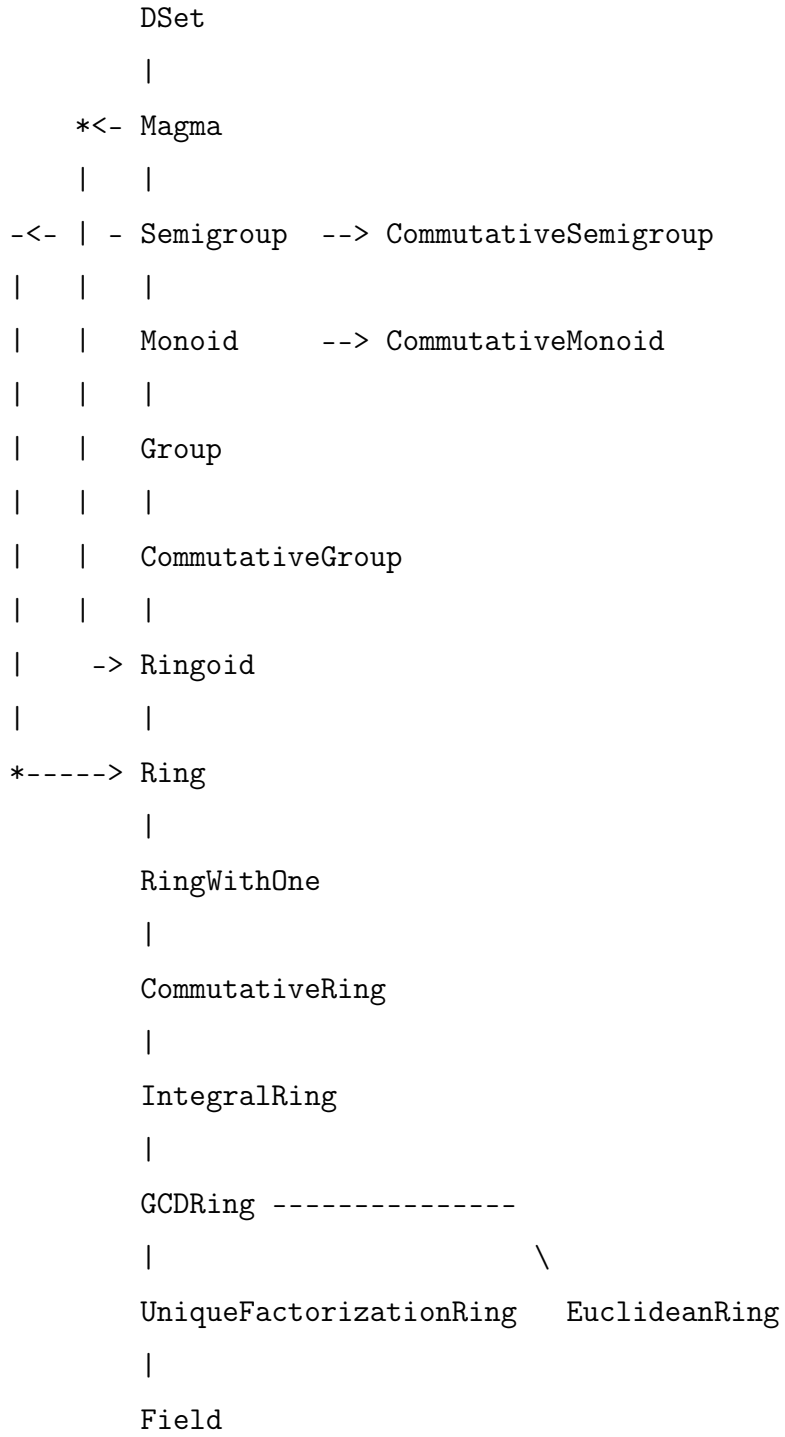
Конструктивная математика [1], конструктивная теория типов [2], доказательства.

Применение ЗТ языка Agda позволяют

- **выражать утверждение P в виде типа T (зависящего от значений),**
- **выражать доказательство для P в виде алгоритма, выдающего значение в T,**
- **выражать область, зависящую от динамически вычисляемых значений,**
- **сопровождать алгоритмы доказательствами, проверяемыми компилятором (проверка делается до начала выполнения программы).**
- **сохранять скорость вычисления при выполнении,**
- **делать автоматическую проверку конструктивныых доказательств теорем.**

DoCon-A.

Иерархия общих алгебраических структур:



6 Алгоритмические методы

Расширенный алгоритм НОД для евклидова кольца.

Арифметика кольца остатков евклидова кольца.

Упорядочение списка, леммы о много-множествах.

Доказательство изоморфизма двоичного представления натуральных чисел.

...

7 Введение в программирование на языке Agda

Конструкторы типов:

`data`, `record`, `_→_` — часть языка

`_×_`, `_⊎_`, `List`, `...` — определяемы программистом.

Пример:

```
data ℕ : Set where zero : ℕ                    -- натуральное число в унарном коде
                  suc  : (n : ℕ) → ℕ

-- 3 <--> suc (suc (suc zero))

-- Функция задаётся алгоритмом -- с доказательством завершаемости.

_+_ : ℕ → ℕ → ℕ                                -- сумма
zero    + n = n
(suc m) + n = suc (m + n)
                                          Вычисление 2 + 1 --> 3 :
{- (suc (suc zero)) + (suc zero)  -->
   suc ((suc zero) + suc zero)    -->
   suc (suc (zero + (suc zero)))  -->
   suc (suc (suc zero))
-}
-- Где доказательство завершаемости:  ...

data _≤_ : ℕ → ℕ → Set                         -- понятие "меньше или равно"
  where
  z≤n : ∀ {n}                → zero ≤ n
  s≤s : ∀ {m n} (m≤n : m ≤ n) → suc m ≤ suc n
```

Об именах: `2+3`, `2≤n` против `2 + 3`, `2 ≤ n`.

```
-- Пример доказательства  (как данного языка):

2≤4 :  suc (suc zero)  ≤  suc (suc (suc (suc zero)))

2≤4 =  s≤s (s≤s z≤n)        -- свидетельство для  2 ≤ 4

-- Доказательство представляется в виде функции (алгоритма), который
-- выдаёт свидетельство для каждого набора значений аргументов.

data _≡_  {A : Set _} (x : A) :  A → Set _    -- пропозициональное равенство
     where
     refl : x ≡ x

-- Пример.  Доказательство прямым вычислением:

2+1≡3 :  2 + 1 ≡ 3
2+1≡3 =  refl

{-  (suc (suc zero)) + (suc zero)  ≡  suc (suc (suc zero)
    suc ((suc zero) + (suc zero))  ≡  suc (suc (suc zero)
    suc (suc (zero + (suc zero)))  ≡  suc (suc (suc zero)
    suc (suc (suc zero))           ≡  suc (suc (suc zero)
    -- a case of  X ≡ X
-}

-- Примеры доказательства по индукции:

≤refl : ∀ n → n ≤ n
≤refl zero    =  z≤n
≤refl (suc n) =  s≤s (≤refl n)  -- шаг индукции представлен рекурсивным вызовом
```

```
≤step : ∀ m n → m ≤ n → m ≤ suc n
≤step zero    _       _         =  z≤n
≤step (suc _) zero    ()
≤step (suc m) (suc n) (s≤s m≤n) =  s≤s (≤step m n m≤n)

                      -- goal :  suc m ≤ suc (suc n)
                      --
                      -- m≤n                :  m ≤ n
                      -- p = ≤step m n m≤n  :  m ≤ suc n
                      -- s≤s p              :  suc m ≤ suc (suc n)

-- Пример использования функции  ≤step :

sub : (m : ℕ) → (n : ℕ) → n ≤ m → ℕ

sub zero    _       _         =  zero
sub m       zero    _         =  m
sub (suc m) (suc n) (s≤s m≤n) =  sub m n m≤n

f : ℕ → ℕ
f n =  -- sub (suc n) n                          -- ошибочно
       -- sub (suc n) n (≤refl n)                -- ошибочно
       -- sub (suc n) n _                        -- неопределённо
          sub (suc n) n (≤step n n (≤refl n))
```

8 Средства построения доказательств

– **Сведéние (normalization, прямое вычисление).**

– **Композиция функций — применение *леммы*.**

– **Рекурсивный вызов — индукция.**

Конъюнкция представлена `_×_` `-- (p1 , p2) : A × B.`

Дизъюнкция представлена `_⊎_` :

`(inj₁ p1) : A ⊎ B;` `(inj₂ p2) : A ⊎ B`

Импликация представлена `_→_` `(f : A → B).`

Отрицание:

```
data ⊥ : Set  where     -- пустой тип

infix 3 ¬_

¬_ :  Set _ → Set _     -- отрицание
¬ P =  P → ⊥

-- Пример:

1≰0 :  ¬ (suc zero ≤ zero)    -- (suc zero ≤ zero) → ⊥
1≰0 ()

2≰1 :  ¬ (suc (suc zero)) ≤ suc zero
2≰1 (s≤s 1≤0) =
                1≰0 1≤0
```

Разрешимые отношения:

```
data Dec (P : Set _) : Set _  where
                              yes :  ( p :   P) → Dec P
                              no  :  (¬p : ¬ P) → Dec P
```

9 О доказательстве от противного, о законе исключённого третьего

Они применимы — для разрешимых отношений.

9.1 Пример, когда закон исключённого третьего не применим

G — свободная группа, порождённая $\{a_1, \ldots, a_9\}$,

$g_1, \ldots, g_7 \in G$, $H = subgroup(g_1, \ldots, g_7)$,

Рассмотрим свойство $w \in H$.

Имеем полуразрешающую процедуру для $w \in H$: ...

Пусть доказывается некое утверждение P для всех $w \in G$,

и доказательство имеет вид

"Если $w \in H$ то ... иначе ..."

...

10 Пропуск доказательства — 'postulate'

11 Стоимость формального конструктивизма

Множитель трудоёмкости 30.

Множитель размера доказательства 15.

Расходы проверяльщика типов (type checker).

12 Начало общей алгебраической библиотеки

```
record Semigroup (A : Setoid) : Set                    -- примерно так
  where
  open Setoid A using (_≈_) renaming (Carrier to C)
  field
    _•_   :  C → C → C
    cong• :  (x y x' y' : C) →  x ≈ x' →  y ≈ y' →  (x • y) ≈ (x' • y')
    assoc :  (x y z : C)  → (x • y) • z ≈ x • (y • z)

 nat+semigroup : Semigroup Nat.setoid        -- полугруппа ℕ по сложению
 nat+semigroup =
                record{ _•_ = _+_;  cong• = cong+;  assoc = assoc+ }
    where
    _+_ :  ℕ → ℕ → ℕ
    0       + n = n
    (suc m) + n = suc (m + n)

    _=n_ =  Setoid._≈_ Nat.setoid          -- равенство на ℕ

    assoc+ :  (x y z : ℕ) → (x + y) + z  =n  x + (y + z)
    assoc+ 0       y z =  refl
    assoc+ (suc x) y z =
                begin
                  ((suc x) + y) + z    =n[ refl ]
                  suc ((x + y) + z)    =n[ PE.cong suc (assoc+ x y z) ]
                  suc (x + (y + z))    =n[ refl ]
                  (suc x) + (y + z)
                □
    cong+ :  (x y x' y' : ℕ) →  x =n x' →  y =n y' →  x + y ≈ x' + y'
    cong+ x y x' y' =  < skipped >
```

Список литературы

1. Марков, А.А.: О конструктивной математике.

   Проблемы конструктивного направления в математике. 2. Конструктивный математический анализ, Труды МИАН СССР, 1962, том 67, страницы 8 – 14, http://mi.mathnet.ru/tm1756

2. Per Martin-Löef: Intuitionistic Type Theory.

   Bibliopolis. ISBN 88-7088-105-9 (1984).

3. Norell, U., Chapman, J.: Dependently Typed Programming in Agda.

   http://www.cse.chalmers.se/~ulfn/papers/afp08/tutorial.pdf

4. С. Д. Мешвелиани:

   О зависимых типах и интуиционизме в программировании математики.

   Программные системы: теория и приложения: электрон. научн. журн. 2014, Т. 5, No 3. http://psta.psiras.ru/read/psta2014_3_27-50.pdf

5. Meshveliani, S. D.: Dependent Types for an Adequate Programming of Algebra. CICM-2013. July 2013, Bath, UK. CEUR Workshop Proceedings, Vol. 1010, ISSN 1613-0073, 2013. http://ceur-ws.org/Vol-1010/paper-05.pdf