ПРОГРАММИРОВАНИЕ, 2023, № 4, с. 3–20

КОМПЬЮТЕРНАЯ АЛГЕБРА

УДК 004.421.6

ДОПУСТИМЫЙ ПОРЯДОК НА МОНОМАХ ВПОЛНЕ ЗАДАН. КОНСТРУКТИВНОЕ ДОКАЗАТЕЛЬСТВО

© 2023 г. С. Д. Мешвелиани[a],*

[a]Институт программных систем им. А.К. Айламазяна РАН
152021 Ярославская обл., Переславский р-н, с. Веськово, ул. Петра Первого, 4 “а”, Россия
***E-mail: mechvel@scico.botik.ru*

Поступила в редакцию 23.09.2022 г.
После доработки 12.11.2022 г.
Принята к публикации 14.11.2022 г.

Обсуждаются конструктивное доказательство завершаемости алгоритма NF построения нормальной формы для многочленов нескольких переменных и связанное с ним понятие допустимого упорядочения $<_e$ на показателях мономов. В классической математике свойство обрыва убывающей цепи (well-quasiorder) отношения $<_e$ выводится из леммы Диксона, и этого достаточно для обоснования завершаемости алгоритма NF. В доказательном программировании на основе конструктивной теории типов (`Coq`, `Agda`) требуется более сильное (в конструктивной математике) свойство: свойство вполне-заданности отношения порядка (соответствует понятию well-founded – в конструктивном определении, как подобие свойства фундированности). Предлагается конструктивное доказательство этой теоремы (Т) для $<_e$, основанное на некотором известном методе, который здесь назовем “метод образцов”. Эта теорема о вполне-заданности произвольного допустимого упорядочения важна и сама по себе, независимо от алгоритма NF. Нам не известны другие работы, в которых (конструктивно) доказана эта теорема. Оказывается, она не очень сложно следует из результатов, достигнутых другими исследователями еще в 2003-м году. Доказательство запрограммировано автором на языке `Agda` в виде библиотеки `AdmissiblePPO-wellFounded` доказательных программ вычислительной алгебры, разработанной автором. Разработка включает в себя применение этой теоремы к доказательному программированию алгоритма NF. Поэтому библиотека также содержит часть доказательных программ алгебры многочленов, которая по объему значительно больше того, что нужно для доказательства теоремы Т.

Ключевые слова: компьютерная алгебра, многочлены, допустимое упорядочение мономов, конструктивное доказательство, лемма Диксона

DOI: 10.31857/S0132347423040106, **EDN:** RDEZQB

1. ВВЕДЕНИЕ

Словоупотребление.

Мы используем следующее словоупотребление.

Воплощение – программная реализация.

Свидетельство – доказательство.

Словарный порядок – лексикографический порядок.

Слово “библиотека” обозначает нашу библиотеку `AdmissiblePPO-wellFounded` [4] доказательных программ вычислительной алгебры.

Термин “хороший предпорядок” – это перевод с английского термина well-quasiorder (WQO), но – для его конструктивной разновидности (будет объяснен ниже). Например, этот перевод применен в [1].

Термин “вполне заданное упорядочение” – это наш перевод с английского термина well-founded ordering (WellFounded – в программах) – версии свойства фундированности для конструктивной логики – для применения конструктивного вывода по трансфинитной индукции (термин будет объяснен ниже). Мы не нашли в источниках перевода на русский язык термина well-founded в конструктивном значении.

Иногда применяем сокращение WF – вполне заданное (well-founded).

Логические связи между понятиями WQO и WellFounded (хороший предпорядок и вполне-заданность) – не такие, как в классической логике для понятия фундированности ([2] Глава 2, параграф 2.2). И классические определения, содержащие выражения наподобие “в любом непустом подмножестве в X существует минимальный элемент”, представлены здесь в более слабом виде, пригодном для алгоритмов, и они имеют несколько другие логические связи.

Понятия well-quasiorder и well-founded relation, – оба в конструктивном смысле, – уже давно употребляются в работах, изданных на английском языке, а также в библиотеках доказательных программ. Например, оба они определены в [18] (параграфы 2, 3). Понятие WellFounded таким же образом определено в стандартной библиотеке языка Agda.

Понятие многочлена определяется для области коэффициентов – коммутативного кольца. Здесь это кольцо считается алгоритмическим (DecCommutativeRing): операции задаются алгоритмами (функциями в программе), и дан алгоритм разрешения равенства.

Применяем разреженное представление многочлена в виде списка *ненулевых мономов*, упорядоченного по определенному закону. Показатель каждого монома представляется в плотном виде – вектор натуральных чисел (то есть – не пропуская нули в записи).

На сложность вычисления во многих методах, опирающихся на арифметику многочленов, существенно влияет выбор упорядочения для списка мономов в многочлене. Даже само решение некоторых вычислительных задач получается просто подбором подходящего упорядочения на множестве показателей мономов.

В этой статье мы приводим описание машинно-проверяемого (формального) конструктивного доказательства теоремы о том, что всякое *допустимое упорядочение* на показателях мономов является вполне заданным (well-founded – в конструктивном определении).

Нам не известны другие работы, в которых (конструктивно) доказана эта теорема.

Оказывается, что она не очень сложно следует из результатов, достигнутых другими исследователями [5] еще в 2003-м году – мы называем их метод “метод образцов”. Надо лишь добавить к нему некоторые построения – они описаны в Разделе 6.

При этом мы здесь вынуждены сначала переформулировать построения из [5], по-своему уточнить подробности, ибо по-другому не получилось построить формальное доказательство на языке Agda. Поэтому будем говорить о нашей разновидности метода образцов.

Полные формальные конструктивные доказательства соответствующих теорем содержится в библиотеке программ [4], написанной автором на языке `Agda` [6, 7].

Также описывается, как эта теорема применяется к доказательному программированию алгоритма нормальной формы для многочленов, применяемого в методе базиса Грёбнера [8].

В статье говорится не только об алгебре и логике, но и о машинно-проверяемых программах формальных доказательств на подходящем языке программирования. Поэтому мы также вынуждены привести какие-то необходимые объяснения относительно системы программирования.

Порядок изложения таков.

Во **Введении** объясняется, что такое допустимое упорядочение $<_e$ показателей в алгебре многочленов, как оно используется в алгоритме нормальной формы; что такое вполне-заданность (WF) отношения порядка и почему доказательство WF нужно для доказательного программирования алгоритма нормальной формы для многочленов нескольких переменных. Также в Разделе 2.2 дается небольшое введение в построение формальных доказательств на языке с зависимыми типами.

В Разделе 3 описываются некоторые подходы к решению задачи.

В Разделе 4 даны определения понятиям из метода образцов: белая область, образцы, уровни, мультимножества образцов (сокращение – PM). Также определяется сведе́ние по показателю образцов, уровней и PM. Определяется упорядочение $<_{pm}$ на PM и объясняется построение доказательства WF отношения $<_{pm}$.

В Разделе 5 описываются свойства сведе́ния образцов и PM. Вводится понятие лестницы показателей, отображение “ esToPM : Лестница $\to$ PM”, вводится WF упорядочение $<_{pps}$ на лестницах. Разбирается пример построения PM по лестнице.

В Разделе 6 с помощью использования упорядочения $<_{pps}$ доказывается конструктивно теорема о вполне-заданности допустимого упорядочения. Описывается структура библиотеки программ, воплощающих это доказательство и функцию нормальной формы многочлена.

1.1. Допустимое упорядочение показателей

Всюду ниже PP обозначает множество (и коммутативный моноид по сложению) показателей.

Аксиомы допустимого упорядочения $<_e$ на показателях таковы:

- STO-e | это строгое линейное (total) упорядочение (IsStrictTotalOrder $<_e$),
- $+_e$mono | $\forall e \in$ PP, $(e +_e)$ сохраняет $<_e$,
- >0 | $\quad e \neq 0_e \Rightarrow e >_e 0_e$.

Например, словарное упорядочение на векторах над $\mathbb{N}$ установленной длины n удовлетворяет этим законам. Умножению мономов соответствует сложение $+_e$ показателей. Например, при $n = 2$ произведение

$$(x^2y^0) * (x^1y^5)$$

мономов имеет показатель [3, 5].

x $\leq_e$ y определим как x $<_e$ y или x = y.

Также для нашей цели важно учитывать *поточечное упорядочение на показателях*: $\forall$i, $e_i \leq e'_i$.

Обозначим его $\leq_p$.

Соответственно, e $<_p$ e' определим как e $\leq_p$ e' и e $\neq$ e'.

Упорядочения $<_p$ и $\leq_p$ являются частичными и не являются линейными.

Как легко следует из аксиом допустимого упорядочения, порядок $<_e$ является расширением для $<_p$:

$$\forall x, y, \quad x <_p y \Rightarrow x <_e y$$

Далее, для мономов m, m' с обратимыми коэффициентами

m делит m' тогда и только тогда, когда

exp(m) $\leq_p$ exp(m').

Будем говорить в таком случае "показатель exp(m) делит exp(m')".

Также в этом случае exp(m) $\leq_e$ exp(m') – в силу аксиом $+_e$mono, >0.

С вычислительной точки зрения одно из выгодных представлений многочлена – это список мономов, упорядоченный по убыванию относительно $<_e$. Понятия

старший показатель lExp многочлена,

старший моном lm, старший коэффициент lc

заданы в смысле упорядочения $<_e$.

1.2. Алгоритм нормальной формы многочлена

Для исследования свойств систем алгебраических уравнений от нескольких переменных важно уметь решать задачу распознавания отношения

$$f \in \text{Ideal(gs)}.$$

Это задача поиска разложения

$$f = q_1 g_1 + \ldots + q_m g_m, \qquad \text{(I)}$$

где gs – список многочленов, $g_i \in$ gs, а q_i – искомые многочлены (задача распознавания принадлежности многочлена идеалу). Она решается с помощью алгоритма вычисления базиса Грёбнера [8]. Частью этого метода является алгоритм нормальной формы многочлена.

Алгоритм NF приведения многочлена f к нормальной форме относительно списка gs многочленов находит многочлены $h, q_1, \ldots, q_m$ такие что

$$f = q_1 g_1 + \ldots + q_m g_m \; + h, \qquad \text{(II)}$$

где h является нормальной формой относительно gs, в том смысле, что $h = 0$ или ни один из старших мономов многочленов из gs не делит старший моном h.

Формула (II) представляет собой некоторое обобщение деления многочленов с остатком: h – остаток, q_i – "частные".

Заметим, что не для всех списков gs алгоритм NF распознает принадлежность идеалу (I); она распознается, например, когда gs является базисом Грёбнера.

Здесь мы считаем, что область K коэффициентов является *полем*, то есть всякий ненулевой коэффициент обратим (например, это так для рациональных чисел). Существуют разновидности этого алгоритма для более общих случаев, но и этот случай уже весьма общий.

Алгоритм NF действует следующим образом.

Нулевой f считается нормальной формой.

Для ненулевого f в списке gs находится первый многочлен, старший показатель e которого делит старший показатель e' многочлена f.

Если такого нет, то нормальной формой является f. Если такой многочлен g найден, то старший коэффициент многочлена g не равен нулю и потому делит старший коэффициент f. Поэтому старший моном m многочлена g делит старший моном m' многочлена f.

Пусть моном m_q является частным этого деления. Тогда алгоритм NF применяется рекурсивно к аргументам f', gs, где

$$f' = f - m_q g \qquad \text{(III)}$$

В нашей библиотеке функция нормальной формы также накапливает частные q_i. Здесь же рассматриваем только вычисление остатка и вопрос завершаемости программы.

(III) – это очередной шаг сведе́ния к нормальной форме.

Список мономов в многочлене упорядочен согласно любому выбранному для всей области многочленов допустимому упорядочению $<_e$ показателей.

Старший моном f при сведе́нии к f' на очередном шаге сведе́ния сокращается, и имеет место

Лемма 1. Все мономы в f' меньше старшего монома f в смысле $<_e$.

Доказательство просто следует из такой же простой леммы:

Лемма 2. Если старшие мономы многочленов f и g равны моному m, то все мономы разности

f – g меньше m в смысле $<_e$ (при этом для нулевой разности список этих мономов пуст).

Доказательство очевидно следует из алгоритма сложения списков мономов и из упорядоченности этих списков.

Теперь для доказательства Леммы 1 надо заметить, что умножение на моном m_q сохраняет порядок на мономах g – в силу свойства $+_e$mono допустимого упорядочения. Также старшие мономы многочленов f и (m_q * g) оказываются равными, и применение Леммы 2 доказывает Лемму 1.

Алгоритм NF работает до тех пор, пока в списке gs находится сводящий многочлен g. Получается убывающая последовательность $e_1 >_e e_2 >_e \ldots$ старших показателей сводимого многочлена.

Определение. Частичный порядок $\preceq$ на множестве X называется хорошим предпорядком (well-quasi-order, WQO), если для любой бесконечной последовательности a_n в X существуют $i < j$ такие, что $a_i \preceq a_j$.

Здесь мы всюду предполагаем, что строгий частичный порядок $\prec$ определен через нестрогий частичный порядок $\preceq$ как $x \prec y = x \preceq y$ и $x \neq y$.

Определение. Назовем частичный порядок $\prec$ *обрывающимся-вниз*, если он не допускает никакой бесконечно убывающей последовательности.

Заметим, что из свойства WQO следует свойство обрывание-вниз. Ибо для любой бесконечной последовательности a_n существуют $i < j$ такие, что $a_i \preceq a_j$, а из этого неравенства в наших условиях следует отрицание отношения $a_j \prec a_i$, и это доказывает, что всякая бесконечная последовательность не является убывающей в смысле $\prec$.

Далее, для упорядочения $\leq_p$ свойство WQO доказывается как известная лемма Диксона, [9] (Section 2.2), [5, 10, 11]. Ее можно выразить так: всякая последовательность e_i показателей, в которой каждый член не делится ни на один предыдущий, обрывается.

Удивительно, что доказательство этого подсознательно очевидного высказывания не так уж и просто изобрести, тем более – когда требуется конструктивное доказательство.

Из леммы Диксона также следует, что любое допустимое упорядочение $<_e$ на показателях является обрывающимся вниз. Ибо для любой бесконечной последовательности e_i показателей по лемме Диксона находятся i < j такие, что e_i делит e_j. Так как $\leq_e$ включает в себя $\leq_p$, то $e_i \leq_e e_j$.

Следовательно, согласно *классической математике*, алгоритм NF считается завершающимся.

Но этого не достаточно для систем доказательного программирования, основанных на конструктивной теории типов: `Coq`, `Agda`, ...

2. ПРОБЛЕМА ДОКАЗАТЕЛЬСТВА ЗАВЕРШАЕМОСТИ АЛГОРИТМА NF

Мы имеем дело с доказательным программированием символьных вычислений в алгебре, основанным (по умолчанию) на конструктивных машинно-проверяемых доказательствах. Для выражения алгоритмов и доказательств применяем язык `Agda` [6, 7].

В работах [9] (Section 2.2), [10] рассматривается среди других предмет конструктивного доказательство леммы Диксона.

[11] доказывает на языке Agda лемму Хигмана для алфавита из двух букв. Вполне возможно, что из этого метода получится доказательство леммы Диксона. Для этого надо показатель монома записать в унарной системе, используя две буквы, где вторая используется как разделитель между записями чисел.

Наконец, в [5] мы увидели, что конструктивное доказательство леммы Диксона на языке `Agda` для современной версии библиотеки языка Agda можно получить, применяя метод образцов из [5] (Section 2.2). Это и было нами сделано. Этот метод выбран нами так как он, как видно из дальнейшего изложения, служит и другим важным целям данного исследования.

Но в языках конструктивных доказательств, основанных на теории зависимых типов, для доказательства завершаемости алгоритма не достаточно опираться на свойство обрывание-вниз некоторого отношения порядка.

Из-за некоторых свойств интуиционистской теории типов оказывается необходимым немного более сильное свойство упорядочения: вполне-заданность (well-founded relation). В частности это требуется в системах `Agda` и `Coq`.

Достижимость и вполне-заданность.

В стандартной библиотеке языка `Agda` написано на языке `Agda` следующее определение (здесь обозначения заменяем на более привычные).

```
WfRec : (_<_ : Rel A) → RecStruct A
WfRec _<_ P x = ∀y (y < x → P y)
“The accessibility predicate: x is accessible
if everything which is smaller than x is also
accessible (inductively).”
data Acc {A : Set} (_<_ : Rel A) (x : A) : Set
  where
  acc : (WfRec _<_ (Acc _<_) x) → Acc _<_ x
WellFounded : Rel A → Set
WellFounded _<_ = ∀x (Acc _<_ x)
```

Этот отрывок программы определяет свойство Acc *достижимости* элемента (x) относительно упорядочения _<_. В строчке acc ... в конструктор WfRec вторым аргументом в переменную P ставится значение – свойство (Acc _<_). Итого получается

$$\mathrm{acc} : (\forall y\ (y < x \to \mathrm{Acc}\ \_<\_\ y)) \to \mathrm{Acc}\ \_<\_\ x$$

При этом определение WfRec и произвольное свойство P на A исчезают из определения достижимости. Поэтому это определение равносильно следующему.

Элемент $x \in A$ является достижимым по отношению $<$ на A, если и только если все y, меньшие x по отношению $<$, являются достижимыми.

(заметим, что отсюда следует достижимость всякого минимального элемента).

Отношение $<$ порядка на множестве A называется вполне заданным (WF), если все элементы A являются достижимыми для $<$.

Для простоты мы здесь рассматриваем эти определения – относительно любого частичного порядка $<$. Хотя в системах `Agda` и `Coq` (в конструктивной теории типов) это так определяется для любого двуместного отношения.

Это не что иное, как свойство *трансфинитной индукции*. То есть для любого вполне заданного упорядочения $\prec$ на множестве A и любого свойства P на множестве A доказательство этого свойства способом трансфинитной индукции по отношению $\prec$ считается законным. Некоторые упорядочения обладают этим свойством. Наша цель – это доказать вполне-заданность отношения $<_e$, или хотя-бы как-то доказать конструктивно (на языке `Agda`), что функция NF завершается.

Ссылки на доказательство леммы Диксона для этого не достаточно.

В работе [10] пишется о доказательной программе метода базиса Грёбнера (алгоритм нормальной формы является частью этого метода). Но в части доказательства завершаемости указывается на неконструктивное доказательство леммы Диксона. Это не соответствует нашей цели.

В работе [9] описывается построение доказательной программы (в системе `Coq`) метода базиса Грёбнера. Но при этом изначально предполагают вполне-заданность упорядочения на мономах (Раздел 2. “We assume a compatible well-founded order $\prec$ on the monomials ...”). Слабым местом такого подхода является то, что пользователю программы придется для каждого применяемого допустимого упорядочения отдельно доказывать вполне-заданность (well-founded). А в зависимости от задачи применяются самые разные упорядочения мономов.

Мы же конструктивно выводим это свойство вполне-заданности в общем случае, из аксиом допустимого упорядочения.

2.1. Примеры вполне заданных упорядочений

Обычный порядок $<$ на $\mathbb{N}$. Здесь WF обращается в обычный принцип индукции по построению натурального числа.

Порядок $<$ на $\mathbb{Z}$ (целые числа) не является ни обрывающимся вниз, ни тем более – WF.

Словарный порядок по $<$ на множестве $\mathbb{N} \times \mathbb{N}$ пар – вполне задан. Предлагаем это доказать как упражнение. Заметим, что здесь существует бесконечно много пар меньших, чем, например, (1, 0).

Известно, что прямое поизведение вполне заданных порядков $\prec$ и $\prec'$ является вполне заданным порядком на множестве пар со словарным упорядочением по порядкам $\prec$ и $\prec'$. В частности словарное упорядочение на множестве n-к (векторов) над $\mathbb{N}$ при постоянном n является WF (это также доказано в нашей библиотеке).

Другой пример: словарное упорядочение на списках натуральных чисел не является WF. Например, последовательность списков 1, 01, 001, 0001, ... бесконечно убывает в словарном упорядочении на списках, чего не может быть для WF упорядочения.

Для некоторых частных случаев допустимого упорядочения $<_e$ нетрудно доказать WF. Например: для словарного порядка <lex, для порядка degLex – “сначала по полной степени, потом – словарно”. Но ясно, что здесь более всего ценно одно доказательство для общего случая.

Также известна классификация всех допустимых упорядочений [12]. Но она основана на очень сложных построениях. И если пользоваться этой классификацией, то скорее всего требуемое доказательство окажется сложнее, чем приводимое ниже, это будет проблемой.

2.2. О доказательном программировании на языке `Agda`

Наша программа опирается на базовую библиотеку вычислительной алгебры `Admissble-PPO-wellFounded` [3, 4] доказательных программ, написанных автором на языке `Agda` [6, 7]. 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 ([3], Введение), например, от целого числа. 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 дела-

ется в учебниках и научных статьях, вместе с доказательством правильности.

Техника проверки доказательств компилятором (точнее – проверяльщиком типов, type checker) основана на изоморфизме Карри–Ховарда [13, 14]. 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 [15, 16] (без использования принципа Маркова для доказательства завершаемости алгоритмов). В частности, всякий объект существует лишь как итог некоторого данного алгоритма, 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.

По умолчанию, доказательства в языке Agda – конструктивные. Конструктивное доказательство имеет то преимущество перед неконструктивным, что, например, вместе с существованием объекта предъявляется алгоритм его построения. Если же доказательство необходимо, а конструктивное доказательство трудно предъявить, то Agda допускает введение аксиомы исключенного третьего (чем в нашей библиотеке мы пока ни разу не воспользовались).

В каком смысле функция на языке Agda может быть доказательством?

Если функция задает доказательство (как, например, в нашем случае доказательства $<_e$wellFounded : WellFounded _$<_e$_ теоремы о WF отношения $<_e$), то заголовок этой функции (выражение ее типа, signature) является формулировкой теоремы, а тело (implementation) этой функции является доказательством этой теоремы. Это доказательство проверяется проверяльщиком типов в общем, символьном виде, так же, как читатель проверяет доказательство, изложенное в книге. Если и когда эта функция (f) вычисляется во время исполнения, то она выдает доказательство для какого-то частного случая. И если головная функция не требует разбора строения этого доказательства, то всякая функция – клиент для f не разбирает это строение при исполнении (здесь также существенно “ленивое” вычисление в языке Agda). За счет этого затраты на проверку необходимых доказательств для головной функции не входят в затраты на вычисление результата головной функции во время работы исполняемого кода.

Другой пример: функция polNF нормальной формы многочлена из нашей библиотеки запрограммирована с использованием доказательства $<_e$wellFounded вышеназванной теоремы (Раздел 6). Это доказательство опирается на некоторые символьные вычисления с типами. Исполняемый код для polNF вычисляет нормальную форму многочлена, но на этом этапе нет затрат на вычисления для доказательства $<_e$wellFounded.

О доказательстве завершаемости

По умолчанию Agda должна убедиться в завершаемости каждой заданной функции (алгоритма). Часто Agda сама убеждается в завершаемости путем обнаружения синтаксического уменьшения рекурсивных вызовов. Но также часто выдается сообщение “Termination checking failed”. В этом случае надо помочь проверяльщику типов распознать завершаемость. Обычно проще всего ввести в функцию дополнительный аргумент в виде счетчика – натурального числа. Но этот подход не является достаточно общим. Например, он не годится для рассматриваемых здесь функций NF, polNF.

В наболее общем случае завершаемость доказывается введением некоторого вполне заданного упорядочения $\prec$ на некотором типе T, добавлением в определение функции f аргумента типа T и доказательством некоторых свойств этого аргумента, в частности того, что этот аргумент уменьшается в отношении $\prec$ при каждом рекурсивном вызове f. Пример этому приводится в дальнейшем изложении. Но два простейших показательных примера содержатся в модуле WellFounded-help.agda библиотеки [4].

Здесь же мы доказываем среди других утверждений, что для доказательств завершаемости функции NF нормальной формы достаточно использовать теорему WF для того допустимого упорядочения $<_e$ на показателях мономов, которое выбрано параметром для арифметики многочленов.

В этой статье мы не можем дать более подробные объяснения по данной системе программирования. Некоторые объяснения и примеры содержатся в [3, 7].

Хорошим введением для начинающих в программирование алгебры на языке Agda является статья [17].

2.3. О библиотеке `AdmissiblePPO-wellFounded`

В этой статье мы говорим об алгоритмах, которые запрограммированы с опорой на арифметику коммутативного кольца многочленов (тип `Pol`) над произвольным полем K, воплощенную в виде доказательных программ. Поэтому эта библиотека включает в себя задание некоторых категорий алгебраических областей дополняющих иерархию алгебры из стандартной библиотеки: IntegralDomain (целостное кольцо с разрешимым равенством), GCDDomain, EuclideanDomain, Field (поле) и некоторые другие. Из конструкторов алгебраических областей реализованы конструктор SVector векторов над моноидом в разреженном представлении, конструктор Pol кольца многочленов нескольких переменных над коммутативным кольцом.

Это на самом деле части библиотеки `DoCon-A` доказательных программ алгебры, разработанной тем же автором; взяты те части, которые нужны для программного воплощения доказательств, обсуждаемых в этой статье.

2.4. 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.

Некоторые замечания о языке и стандартной библиотеке:

x : T означает “x принадлежит типу T” (конструкция языка Agda).

Знак ‘=’ это присваивание, конструкция языка.

Равенство _≈_ это знак абстрактного отношения равенства на некоторой области, для каждой области это отношение может задаваться программой пользователя или стандартной библиотекой.

_::_ это функция из стандартной библиотеки, x :: xs означает присоединение x к списку xs.

Прописные и строчные буквы в программе различаются проверяльщиком типов. Обычно имена типов начинаются с прописной буквы, а имена функций начинаются со строчной буквы.

@ это знак присваивания в образце (конструкция языка). Например, в программе

```
foo f@(mkPol mons cs≠0 ord-exps) g =
          f + (foo' mons) * g where …
```

функция foo принимает аргумент f, разбирая его по образцу, и в правой части используется как имя f, так и часть mons образца для f.

2.5. Использование вполне-заданности $<_e$ в программе NF

Представим себе, что мы получили доказательство $<_e$wellFounded того, что всякий допустимый порядок $<_e$ на показателях является вполне заданным.

Тогда функция NF в простейшем виде программируется примерно так (условный код, для показа замысла):

```
NF gs f = nf gs f (<e wellFounded e[f])
  where
  e[f] = lExp f
  gs0 = gs
  nf : (gs : List Pol) (h : Pol) →
                          let e[h] = lExp h hm
                          in
                          Acc _<e_ e[h] → Pol
  nf gs h (acc rc) = ...
```

То есть вызывается местная вспомогательная функция nf, которая будет ниже программироваться рекурсивно.

($<_e$wellFounded e[f]) это *свидетельство* (доказательство) того, что показатель e[f] достижим по $<_e$. В предложениях для nf имеем следующие обозначения.

h это текущий остаток, который сводится по списку gs и в конце оказывается нормальной формой.

(acc rc) это свидетельство того, что показатель e[h] старшего монома из h достижим (для случая, когда h не нулевой).

А всякое свидетельство достижимости имеет вид (acc rc), где acc это стандартный конструктор, rc функция из определения достижимости во вполне заданном порядке.

В случае h ≠ 0 многочлен h имеет старший моном. Обозначим его m, а его показатель – e[h]. Когда в gs встретился многочлен g, старший моном которого делит m, работает такое предложение из функции nf :

nf gs h (acc rc) = nf gs_0 h’ (rc h’ e[h’]<e[h])

В правой части: gs_0 это запомненное первоначальное значение для gs,

h' = h – q*g это итог шага сведéния, когда старший моном сокращается.

e[h'] = lExp h' (если h' не нулевой).

e[h']<e[h] это свидетельство того, что e[h'] $<_e$ e[h]. Это доказательство получается в теле функции nf применением функции, воплощающей Лемму 1.

Данное (rc h' e[h']<e[h]) это свидетельство того, что показатель e[h'] достижим.

Таким образом от h и свидетельства достижимости e[h] функция nf рекурсивно переходит к h' и достижимости e[h']. Проверяльщик типов (type checker) системы Agda распознает завершаемость этой рекурсии, так как видит доказательство $<_e$wellFounded, доказательство достижимости e[h'] и доказательство e[h']<e[h].

Это самый простой способ запрограммировать метод NF в системе доказательного программирования на основе зависимых типов.

Уточнение программы NF

Когда очередной g не сводит h, и список gs не пуст, метод вызывает nf на аргументах gs' = хвост gs, h, acc' – это шаг "пробовать следующий элемент из gs". При этом многочлен h остается неизменным. Это нарушит доказательство завершаемости, так как третий аргумент должен содержать доказательство уменьшения соответствующего аргумента относительно выбранного WF упорядочения.

Поэтому в аргументы nf добавлены данные cnt : ℕ, eq : cnt ≡ length gs, где cnt – счетчик длины текущего списка gs. Так что при каждой рекурсии уменьшается либо старший показатель h (в смысле $<_e$), либо значение cnt. Чтобы обеспечить требуемое уменьшение, программа сначала определяет словарный порядок на множестве PP × ℕ пар. Пары (e , i) и (e' , j) сравниваются сначала по первой составляющей через $<_e$, и в случае равенства, сравниваются вторые составляющие по упорядочению _<_ на ℕ. Это выражено кодом

$_<_e<_$: Rel (PP × ℕ) _

$_<_e<_$ = ×-Lex _≡_ $_<_e_$ _<_,

где ×-Lex функция из стандартной библиотеки, задающая словарное произведение двух упорядочений. Потом программа доказывает что порядок $<_e<$ вполне задан:

$<_e$<-wellFounded : WellFounded $_<_e<_$

$<_e$<-wellFounded =

×-wellFounded $<_e$wellFounded <-wellFounded

Это делается вызовом общей функции ×-wellFounded, которая принимает свидетельства WF для каждого из двух порядков и возвращает свидетельство WF для прямого произведения этих порядков (простое доказательство из стандартной библиотеки). Наконец, вместо аргумента ($<_e$wellFounded e[f]) в начальном вызове nf ставится

($<_e$<-wellFounded (e[f] , length gs_0))

И при рекурсии в соответствующий аргумент ставится доказательство убывания по упорядочению $<_e<$ пары при переходе

(e[h] , длина gs) -> (e[h'] , длина gs')

В подробностях эту функцию (polNF) можно увидеть в модуле

source/GroebnerBasis/Field/PolNF-sample.agda

нашей библиотеки [4]. Это частный случай метода NF, служащий для ясного показа метода.

Настоящая функция polNF нормальной формы относительно списка многочленов содержится в модуле PolNF.agda того же раздела. Она еще накапливает список частных и в итоге выдает запись (record), содержащую нормальную форму h, список частных, свидетельство равенства (II) (Раздел 1.2), свидетельство нормальности h по gs.

Таким образом главный успех предприятия зависит от того, сумеем ли мы конструктивно доказать теорему $<_e$wellFounded вполне-заданности отношения $<_e$.

Эта теорема важна и сама по себе. Так как понятие допустимого упорядочения на показателях мономов часто используется в вычислительной алгебре.

Изложим решение этой задачи.

3. ПОДХОД К ДОКАЗАТЕЛЬСТВУ ВПОЛНЕ-ЗАДАННОСТИ УПОРЯДОЧЕНИЯ $<_e$

Известно, что в классической логике свойства WQO (или свойство обрывание-убывающей) и вполне-заданность равносильны. Но в конструктивной логике вполне-заданность не выводится из свойства обрывание-убывающей.

В статье [18] в Разделе 3.1 описано, каким образом в довольно общем случае для отношения со свойством WQO можно конструктивно доказать вполне-заданность – при некотором дополнительном условии наличия некоторого вполне заданного дерева – well-founded tree (WFT).

Но мы не видим, как строить такое дерево для отношения $<_e$.

Есть еще такой выход из положения. Для доказательства завершаемости функции NF не обязательно использовать вполне-заданность $<_e$. Достаточно связать (правильным образом) с ходом вычисления NF некое данное, которое с каждой рекурсией будет уменьшаться в смысле некоторого подобранного упорядочения $\prec$ на множестве

таких данных. При этом должна быть доказана вполне-заданность $\prec$.

И оказалось, что метод мультимножеств образцов (multiset of patterns) из Раздела 2.2 статьи [5] как раз применим в таком виде к конструктивному доказательству завершаемости метода NF.

Авторы [5] доказывали только лемму Диксона, причем использовали язык `ACL2*`, с которым мы не знакомы. Нам достаточно было обозреть общий замысел метода образцов в двух страницах раздела 2.2, чтобы домыслить подробности и запрограммировать на языке Agda этот метод. Наш алгоритм reducePM сведéния множества образцов (reduction of pattern set) отличается от того алгоритма, который имеется в виду в [5] (и подробности которого нам не ясны). Но это отличие не существенно, так как для нашего алгоритма мы доказали свойства, которых достаточно для конструктивного доказательства вполне-заданности допустимого упорядочения на показателях а также леммы Диксона.

При применении этого метода reducePM получается данное pm в виде мультимножеств образцов, порядок <pm на этих данных и доказательство свойства WF для <pm. Если это данное pm и некоторые ему сопутствующие данные добавить к аргументам функции polNF, то получится доказательно завершающаяся функция.

Но эта программа выглядела бы гораздо сложнее, чем целевая программа NF приведенная выше: много дополнительных аргументов в функции и доказательств связи между ними.

К счастью оказалось, что через свойство WF отношения <pm и свойства сведéния reducePM данных pm нетрудно доказать вполне-заданность отношения $<_e$ – что и приводит нас к простой программе NF, показанной выше.

Ниже описываем этот подход.

4. ОБЛАСТИ, ОБРАЗЦЫ, УРОВНИ

Пусть es список показателей. Затененной областью Sh(es) назовем (следуя [5]) множество показателей e таких, что e делится на какой-либо показатель из es. Это объединение нескольких ортантов.

Назовем белой областью W(es) дополнение `PP\Sh(es)`.

Назовем *лестницей* такой список es показателей, в котором каждый показатель не делится ни на один последующий в списке.

Для лестницы es белая область оказывается той частью пространства PP, которая находится “под лестницей”, заданной показателями es как ступеньками.

Например, для $n = 2$ лестница списка $[e_0, e_1] = [(3,2), (1,5)]$ изображена на рис. 1.

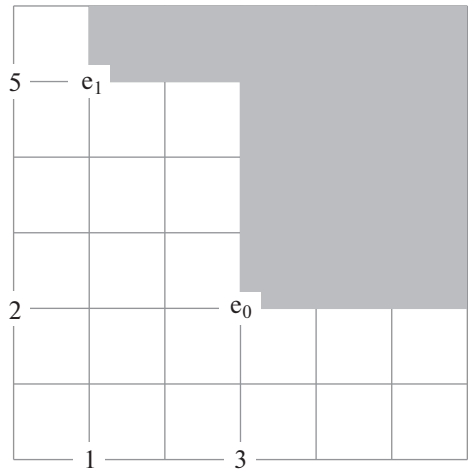

Рис. 1.

(Оба примера представления белых областей в этой статье взяты из [5]).

На рис. 1 изображена часть квадранта $0 \le x, y \le 6$. Здесь белая область состоит из двух горизонтальных и одного вертикального лучей и еще нескольких точек.

Добавление к лестнице спереди любого показателя e_2, принадлежащего $W[e_0, e_1]$, дает белую область меньшую, чем $W[e_0, e_1]$, в смысле включения $\subset$ множеств. И так далее.

Пусть <st частичное упорядочение на множестве Stc лестниц, при котором es <st es’ означает $W(es) \subset W(es')$.

Очередной целью является подобрать подходящее конечное символьное представление для белой области и доказать, что отношение <st вполне задано. Оказывается, что из этого можно вывести все остальные цели статьи.

Обратимся к методу образцов из работы [5].

Обозначим

$$\mathbb{N}_* = \mathbb{N} \cup \{*\}$$

множество натуральных чисел, расширенное элементом *, называемым *свободой*.

Образцом называется вектор в $\mathbb{N}_*^n$.

Количество свобод в образце называется *размерностью* образца.

Размерность каждого образца не меньше нуля и не больше n – числа переменных в алгебре многочленов.

Образец без свободы называем основным (в программе – Ground), он имеет размерность 0.

Образец p представляет множество показателей e таких, что вектор p совпадает с вектором e в каждой позиции, кроме тех, что содержат *.

Как в статье [5], обозначим это множество S(p). Для списка или множества ps образцов обозначим S(ps) объединение $\cup \{S(p) \mid p \in ps\}$ (сокращение: Subset in PP defined by Patterns ps).

Таким образом всякая белая область получает конечное символьное представление в виде конечного множества образцов (таких представлений может быть много).

Имеет смысл собирать в один *уровень* образцы одной размерности и представлять белую область в виде списка уровней, убывающего по размерности, пропуская пустые уровни.

Такой список уровней называем *мультимножеством образцов* (PMultiset, коротко – PM) (в [5] близкое этому понятие называют multiset).

Заметим, что длина списка уровней в PM не превосходит 1 + n, так как эти уровни упорядочены по убыванию размерности, а размерность уровня не превосходит n.

Уровень и упорядоченный список уровней введены автором, этих построений нет в [5]. В таком виде мы понимаем их отношение к построению мультимножества по образцам, к сведéнию мультимножества и наконец, к доказательству цели, и сумели запрограммировать доказательство на языке Agda. Хотя здесь мы все это описываем на неформальном математическом языке.

Область S(pm), задаваемая образцами в нашей разновидности метода, не обязательно совпадает с W(es), но она включает в себя W(es). Вместе с вполне-заданностью упорядочения <pm (определяемого ниже в этом разделе) это дает основу для доказательства цели.

Из [5] мы взяли только главное – замысел представления области в виде образцов и построения из этого мультимножества, которое можно представить в виде вектора натуральных чисел.

Уровень – это пара из значения размерности и соответствующего списка образцов.

Например, для n = 2, образец [* , *] (generalPattern) представляет все PP – белую область для пустого списка.

Другой пример: белая область $W[e_0, e_1]$ на рис. 1 может быть представлена, например, такими уровнями размерности 1 и 0:

```
{(1, [0*, *0, *1]),
   (0, [10, 11, 12, 13, 14, 20, 21,
22, 33, 24])}
```

В этом и последующих примерах распечатки образцов мы часто пишем ij как краткое обозначение для пары (i, j), когда i, j < 10, и i* вместо (i, *).

Кратностью (lMulty) уровня назовем длину списка этого уровня.

Заметим, что длина списка не обязательно равна порядку множества элементов списка. Но для наших построений эта разница не существенна.

О применяемых упорядочениях:

чтобы облегчить чтение статьи, перечислим здесь все используемые нами упорядочения.

$<, \leq$ – упорядочения на натуральных числах.

$\leq_p$ – покоординатное упорядочение по $\leq$ на векторах над $\mathbb{N}$ (частичное).

$<_e$ – произвольное допустимое упорядочение на PP (векторах над $\mathbb{N}$).

$\leq^*$ – отношение между элементами $\mathbb{N}$ и элементами $\mathbb{N}_*$ (определенное ниже).

≤ep – отношение между показателями (типа PP) и образцами (типа Pattern), заданное (ниже) покоординатно через $\leq^*$ (частичное).

<nn – упорядочение на парах натуральных чисел, заданное словарно через <.

<lev – упорядочение на уровнях, заданное (ниже) через упорядочение <nn пар (размерность уровня, длина списка образцов уровня).

<ls – упорядочение списков уровней, заданное (ниже) словарно относительно упорядочения <lev.

<pm – упорядочение на мультимножествах образцов (PM), заданное (ниже) через <ls (почти то же, что <ls).

<pps – упорядочение на списках показателей, определенное (ниже) через <pm и отображение esToPM! : List PP → PMultiset.

Некоторые из этих упорядочений, – в общем случае скажем $\prec$ на X, – определяются через другое упорядочение, скажем $\prec'$ на Y, путем перенесения по подобранному отображению f : X → Y. При этом x $\prec$ x' определено как f x $\prec'$ f x'.

В стандартной библиотеке эта конструкция осуществляется оператором _on_ :

$$\_\prec\_ = \_\prec'\_ \text{ on } f$$

Если $\prec'$ вполне задано на Y, то $\prec$ вполне задано на X. Эта простая лемма доказана в стандартной библиотеке.

Назовем *размером* уровня lev пару натуральных чисел lSize lev = (d , |pats|)

– размерность уровня lev и его кратность.

Пусть <nn это словарный порядок на множестве $\mathbb{N} \times \mathbb{N}$. Через него определим порядок на уровнях:

<lev : Rel DimLevel _

<lev = _<nn_ on lSize

– то есть соответствующие значения lSize сравниваются по <nn.

Порядок <ls на списках уровней задаем как словарный порядок над порядком <lev :

_<ls_ : Rel DimLevels _

_<ls_ = Lex-< _ _<lev_

Порядок <pm на мультимножествах образцов задаем так:

_<pm_ : Rel PMultiset _

_<pm_ = _<ls_ on dimLevels

– сравнить списки уровней в словарном порядке над порядком <lev.

Видно, что <pm – это порядок на PMultiset изоморфный порядку lex(<nn) на мультимножествах над $\mathbb{N}$, представленных в виде упорядоченных по убыванию размерности списков пар (размерность, кратность), где 0 ≤ размерность ≤n.

То есть <pm по сути дела задает порядок на списках вида

$$(d_1, m_1), \ldots, (d_k, m_k), \qquad \text{(IV)}$$

где d_i, $m_i \in \mathbb{N}$, $n \geq d_1 > \ldots > d_k \geq 0$, и все кратности m_i ненулевые.

Этот порядок вкладывается в словарный порядок на множестве векторов над $\mathbb{N}$ длины 1 + n. При этом вложении d_i суть номера мест в векторе, а составляющие в остальных местах заполняются нулями.

Известно, что такой порядок является вполне заданным.

В стандартной библиотеке `lib-1.7` языка Agda нет доказательства этому. Это доказательство запрограммировано в нашей библиотеке для разреженного представления вектора в виде (IV).

Так что программа теперь ссылается на функцию <pm-wellFounded конструктивного доказательства вполне-заданности упорядочения <pm на мультимножествах образцов.

Далее, задуманное нами доказательство вполне-заданности допустимого порядка $<_e$ имеет следующий вид. Пусть e : PP – любой показатель. Надо доказать его достижимость Acc _$<_e$_ e. То есть доказать, что всякий показатель e' $<_e$ e является достижимым.

e не делит e', в силу законов отношения допустимого упорядочения для $<_e$ (Раздел 0.1).

Поэтому получаем лестницу из e' :: e :: [] из двух показателей (смотри начало Раздела и рисунок 1).

Далее, надо доказать Acc _$<_e$_ e" для каждого e" $<_e$ e'. Отношение $<_e$ транзитивно. Поэтому выполнено e" $<_e$ e, и потому список e" :: e' :: e :: [] является лестницей. Рассмотрим все последовательности лестниц вида

[],[e], (e' :: e ::[]), (e" :: e' :: e :: []), ...

– очередная лестница получается из предыдущей добавлением головного показателя, который меньше по $<_e$ всех остальных показателей лестницы.

Каждой лестнице st из такой последовательности сопоставляем ее белую область W(st). Очевидно, что эти множества убывают по включению:

$$W([]) \supset \ldots \supset W(st_i) \supset W(st_{1+i}) \supset \ldots$$

Ниже мы по закону сведéния каждой такой области (лестнице) сопоставляем мультимножество pm(st) образцов так, что pm(st(1+i)) получается из pm(st(i)) некоторым сведéнием образцов (pattern reduction) по очередному показателю e_i и перемещением полученных образцов между уровнями.

Ниже определяем это сведéние и доказываем, что каждое сведéние pm(st(i)) → pm(st(1+i)) уменьшает мультимножество в смысле порядка <pm.

В нашей разновидности метода образцов область S(pm), задаваемая образцами, не обязательно совпадает с W(es). Но она включает в себя W(es). В методе, описываемом ниже, последовательность pm_0 >pm pm_1 >pm ... убывает в подобранном нами вполне заданном порядке <pm, и это дает основу для доказательства цели.

4.1. Сведéние образца, уровня, мультимножества образцов

Отношение ≤* между элементами $\mathbb{N}$ и $\mathbb{N}_*$ определим так:

i ≤* * для любого натурального i,

и i ≤* j значит, что i ≤ j для натурального j.

Отношение ≤ep между показателями и образцами (типа Pattern) определим как ≤* примененное поточечно. Это частичный порядок.

p : Pattern называется *сводимым* по e : PP (e Reduces p), если e ≤ep p.

Это определение равносильно определению отношения “p is reducible by e” из ([5], 2.2).

Для образца p, сводимого по показателю e, функция сведéния

reducePattern : PP → Pattern → Patterns

выдает список образцов, называемых нами редексами. В целевом доказательстве она применяется только когда e сводит p.

Если p содержит * в некотором месте вектора, и e содержит j в этом месте, то список редексов для этого места – это все образцы p'(0), ... p'(j-1), где p'(k) получен из p заменой * в этом месте на число k.

Исключение делается для j = 0, в каком случае * на этом месте заменяется на 0.

Здесь имеется отличие от сведéния в [5], где пишется о множестве всех чисел меньших j, тогда как при j = 0 это множество оказывается пустым. Сначала мы пытались следовать этому построению, но при этом получаемое множество образ-

цов иногда оказывалось недостаточным. Поэтому мы применили описанное выше исключение для j = 0. По крайней мере, доказательство цели – проходит.

Это действие для * производится по очереди для всех мест свободы в p. Полученные списки редексов соединяются вставкой применением функции insertPatterns так, что возвращаемый список редексов не содержит повторов.

Наш алгоритм сведе́ния устроен так, что для основного образца, сводимого по e, выдается пустой список.

Доказываем в программе, что для образца размерности d сводимого по e все остатки в (reducePattern e p) имеют размерность d – 1.

Функция

reducePatternToLevel : (e : PP) (p : Pattern) →
e Reduces p → DimLevel

требует, чтобы p был сводим по e. Она применяет (reducePattern e p) и выдает уровень, состоящий из этих остатков, и все они имеют размерность (dim p) – 1.

Сводимый основной образец дает при таком сведе́нии пустой уровень.

Уровень lev называем сводимым по показателю e (e ReducesLevel lev), если e сводит некоторый образец в lev.

Функция

```
reduceLevel : (e : PP) (lev : DimLevel) →
                                    DimLevels
```

сводит уровень, она выдает список не более, чем из двух уровней следующим образом.

Если lev не сводим по e, то выдается [lev].

Иначе, если lev содержит один образец p, то выдается [reducePatternToLevel e p rd].

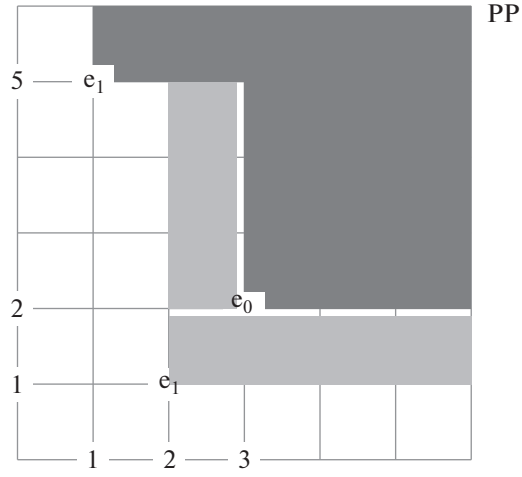

Рис. 2.

Иначе в lev находится первый образец p сводимый по e, вычисляется уровень

rLev = reducePatternToLevel e p e-reduces-p,

и выдается список (delPattern p lev) :: rLev ::[]

из двух уровней. Первый из них получен удалением из lev первого вхождения образца p. В итоге первый уровень имеет размерность dim(lev) и кратность lMulty(lev) – 1,

второй – имеет размерность dim(lev) ∸ 1 (здесь стоит знак усеченного вычитания натуральных чисел: если уменьшаемое меньше вычитаемого, то итогом считается 0).

Вообще, сведе́ние образцов, уровней и списков уровней устроены так, что было легче доказать согласованность сведе́ния с упорядочением <pm и с отношением включения белых областей.

Функция

```
preReduceLevels : PP → DimLevels → DimLevels
preReduceLevels _ [] = []
preReduceLevels e (l :: levs)
                        with e reducesLevel? l
… | no _ = l :: (preReduceLevels e levs)
… | yes rd-l =
        insertLevels (reduceLevel e l) levs
```

принимает показатель e и список levs уровней и возвращает список уровней. Она ищет в levs первый сводимый по e уровень. Если такого нет, то возвращает levs.

Иначе она сводит найденный уровень в список levs' уровней и вставляет levs' в оставшийся хвост списка levs.

Здесь функция insertLevels вставляет уровни в список по одному, применяя функцию insertLevel.

insertLevel вставляет уровень l в список согласно убыванию размерности. Если встретится уровень l' той же размерности, то l и l' сливаются в один уровень (joinLevels^{r} l l' d≡d'), при этом спис-

ки образцов приставляются друг к другу, соответственно, кратности складываются.

Функция reduceLevels применяет preReduceLevels и удаляет из результата пустые уровни.

Напомним, что мультимножество образцов (PMultiset, PM) состоит из списка непустых уровней, упорядоченного по убыванию размерности уровня.

Отношение

```
_ReducesPM_ : REL PP PMultiset _
_ReducesPM_ e pm =
             Any (e ReducesLevel_) (dimLevels pm)
```

означает, что показатель e сводит PM, то есть сводит некоторый уровень в PM.

Сведе́ние PM выражается функцией

```
reducePM : (e : PP) → PMultiset → PMultiset
reducePM e pm@(mkPM levs allHasHead-levs ord-levs)
                              with e reducesPM? pm
  … | no _             = pm
  … | yes any-rd-levs =
         mkPM levs' allHasHead-levs' ord-levs'
  where
  levs' = reduceLevels e levs
  ord-levs' = reduceLevels-preserves-<d-DecrOrdered
                                        e ord-levs
  allHasHead-levs' = allHasHead∘reduceLevels e levs
```

Она оставляет pm : PMultiset (с уровнями levs) неизменным, если e не сводит pm.

Иначе она выдает pm' : PMultiset, в котором уровни получены вызовом (reduceLevels e levs).

Здесь allHasHead∘reduceLevels это функция, которая строит доказательство того, что каждый уровень, выдаваемый функцией reduceLevels, не пуст.

5. СВОЙСТВА СВЕДЕ́НИЯ МУЛЬТИМНОЖЕСТВА ОБРАЗЦОВ

В нашей программе доказана на языке Agda теорема

```
reducePM<pm : {e : PP} {pm : PMultiset} →
                        e ReducesPM pm →
                        reducePM e pm <pm pm
reducePM<pm {e} {pm} e-rd-pm with e reducesPM? pm
… | no ¬rd = contradiction e-rd-pm ¬rd
… | yes rd = reduceLevels<ls (decrOrdered pm) rd
```

– если e сводит мультимножество pm образцов, то (reducePM e pm) выдает pm' : PMultiset, которое меньше pm в смысле <pm.

Доказательство основано на следующих соображениях.

Если сводится уровень нулевой размерности, то это выражается в удалении одного образца из этого уровня. При этом уменьшается кратность уровня.

Если сводится уровень ненулевой размерности, то либо он заменяется на уровень меньшей размерности, либо он заменяется на уровень той же размерности с меньшей кратностью и еще на уровень меньшей размерности.

Надо иметь в виду, что в нашей задаче сведе́ние списка уровней применяется только к тем спискам, которые уже упорядочены по убыванию размерности уровня. Подробности такого рода существенны при построении доказательств.

5.1. Соответствие лестница → мультимножество образцов

Есть важное свойство согласованности между PM, лестницами (их белыми областями) и действием сведе́ния. В ходе действий над PM (сведе́ние образцов, вставка уровней, и тому подобное)

появляются разные PM. Для нашей цели необходимо убедиться в сохранении важных свойств связи появляющихся PM и белых областей.

Говорим, что показатель e принадлежит образцу p (e ∈ pat p),

если e получается подстановкой некоторых чисел вместо вхождений * в p.

Заметим, что если e ∈ pat p, то e Reduces p.

Соответственно считаем, что e принадлежит списку ps образцов (e ∈ pats ps),

если e ∈ pat p для некоторого p из ps (Any (e ∈ pat_) ps).

Считаем, что e принадлежит уровню lev, если e принадлежит списку образцов lev. Считаем, что e принадлежит pm : PMultiset (e ∈ pm pm), если e принадлежит некоторому уровню из pm.

Определение

```
LevelsCoverRegion : REL DimLevels (List PP) _
LevelsCoverRegion levs es =
           ∀(e : PP) → es ¬EsDivides e →
                       e∈els levs
```

значит, что список levs уровней покрывает белую область (“Region”), заданную списком показателей. То есть всякий показатель e, принадлежащий области W(es), также принадлежит некоторому уровню из levs.

Отношение “ pm : PMultiset покрывает белую область W(es)” (PMCoversRegion pm e) означает, что список уровней из pm покрывает W(es).

Далее, функция

esToPM : (es : List PP) →

∃(\(pm : PMultiset) → PMCoversRegion pm es)

сопоставляет списку es показателей некоторое PM, которое покрывает W(es). Это задает отображение \es → pm и еще доказательство соответствующего утверждения об этом отображении. Эта функция устроена следующим образом. Начальное PM (generalPM) состоит из свобод, покрывает все пространство PP и соответствует пустому списку показателей. По рекурсии, если текущий список es отображен в pm, которое покрывает W(es), то тогда (reducePM e pm) покрывает область W(e :: es). Этот шаг индукции доказывается в теле функции esToPM.

Функция

esToPM! : List PP → PMultiset

esToPM! es = proj_1 (esToPM es)

отличается от esToPM тем, что возвращает только PM.

Пример. Зададим три показателя:

$e_0 = [3, 2]$, $e_1 = [1, 5]$, $e_2 = [2, 1]$.

Применяя функцию esToPM! (опирающуюся на сведéние PM), построим три PM, соответствующие лестницам $[e_0]$, $[e_1, e_0]$, $[e_2, e_1, e_0]$:

$pm[e_0]$ = esToPM! [e_0]

$pm\text{-}e_1e_0$ = esToPM! (e_1 :: e_0 :: [])

$pm\text{-}e_2e_1e_0$ = esToPM! (e_2 :: e_1 :: e_0 :: [])

Белая область для лестницы $[e_2, e_1, e_0]$ показана на рис. 2.

Демонстрационная функция (модуль Pol. Dickson.Test нашей программы) этого примера показывает такие значения для этих PM:

```
pm[e0] = PM [ Lev [*1, *0, 2*, 1*, 0*] ]
pm-e1e0 = PM [ Lev [*1, *0, 1*, 0*],
           Lev [24, 23, 22, 21, 20] ]
pm-e2e1e0 = PM [ Lev [*0, 1*, 0*],
     Lev [11, 01, 24, 23, 22, 21, 20]]
```

Согласно алгоритму esToPM! эти три PM получены последовательным применением функции reducePM :

[*, *] –reducePM e_0–> $pm[e_0]$ –reducePM e_1–>

$pm\text{-}e_1e_0$ –reducePM e_2–> $pm\text{-}e_2e_1e_0$

При этом выполнены неравенства

$pm[e_0]$ >pm $pm\text{-}e_1e_0$ >pm $pm\text{-}e_2e_1e_0$.

Белая область $W[e_0]$ представима в виде двух горизонтальных и трех вертикальных лучей. Это в точности выражает значение $pm[e_0]$.

$W[e_1e_0]$ представима двумя горизонтальными и одним вертикальным лучами, и еще несколькими точками. $pm\text{-}e_1e_0$ содержит образец 1*, из-за чего $pm\text{-}e_1e_0$ задает область, строго включающую $W[e_1e_0]$. Это допустимо – условие правильности алгоритма названо ниже.

$W[e_2e_1e_0]$ представима одним горизонтальным и одним вертикальным лучами, и еще четырьмя точками. $pm\text{-}e_2e_1e_0$ задает область, строго включающую $W[e_2e_1e_0]$. Кроме того, в этом PM имеются наложения. Например, образец 1* включает образец 11. На правильность применения метода это не влияет.

Достаточные условия правильности этого метода образцов таковы.

• Очередное PM покрывает соответствующую белую область.

• Очередное PM меньше предыдущего PM в смысле <pm.

Первое из этих свойств следует из леммы о принадлежности показателя сведéнию (доказанной в нашей библиотеке):

```
e∤ e' ∧ e' ∈ pat ∧ e-reduces-pat ⇒ e'∈ reduce-e-pat :
                {e e' : PP} {p : Pattern} →
                e ⋠ S' e' → e' ∈ pat p → e ≤ ep p →
                Any (e' ∈ pat_) (reducePattern e p)
```

– “Если e не делит e’, e’ принадлежит образцу p и e сводит p, то e’ принадлежит некоторому из образцов reducePattern e p”.

6. ТЕОРЕМА О ВПОЛНЕ-ЗАДАННОСТИ ДОПУСТИМОГО УПОРЯДОЧЕНИЯ

Зададим порядок на списках показателей:

```
_<pps_ : Rel (List PP) _
_<pps_ = _<pm_ on esToPM!
```

То есть для списков es и es’ сравниваются два PM, полученные применением esToPM к спискам es и es’ соответственно.

Так как порядок <pm вполне задан, то и <pps вполне задан. Стандартная библиотека весьма просто доказывает это в общем случае, как свойство отображения _on_ на упорядочениях.

Функция

```
e:es<pps-es : ∀e es → All (e <e_) es →
                                (e :: es) <pps es
```

доказывает Лемму:

если показатель e меньше всех показателей из es по допустимому упорядочению $<_e$, то (e :: es) <pps es.

В частности, при добавлении такого показателя спереди к лестнице получается меньшая лестница в смысле <pps.

Доказательство легко следует из свойств функции reducePM, доказанных раньше.

Теорема WF-admissible.

Всякое допустимое упорядочение $<_e$ показателей вполне задано.

Следующая программа на языке Agda формулирует и доказывает эту теорему.

```
<e wellFounded : WellFounded _<e_
<e wellFounded e =
  aux e [] [] (<pps-wellFounded [])
  where
  aux : (e : PP) (es : List PP) →
        All (e <e_) es → Acc _<pps_ es →
        Acc _<e_ e
  aux e es e<es (acc rec-es) = acc rc
    where
    rc : ∀e' → e' <e e → Acc _<e_ e'
    rc e' e'<e =
      aux e' (e :: es) e'<ees (rec-es _ ees<pps-es)
        where
        ees<pps-es = e:es<pps-es e es e<es
        e'<ees =
          e'<e ::
            (AllM.map (\e<x → <e trans e'<e e<x) e<es)
```

Это короткое доказательство выражает формально следующее неформальное рассуждение.

Введем сокращения и обозначения:

ees = e :: es – для показателя e и списка es,

AccE = Acc _$<_e$_ – достижимость показателя по $<_e$,

AccPPs = Acc _<pps_ – достижимость списка показателей по <pps,

e <all es = All (e $<_e$_) es – отношение “ e меньше всех показателей из списка es”.

(определение достижимости (WellFounded, Acc) дано в Разделе 2).

Пусть e – любой показатель. Надо доказать его достижимость AccE e. То есть доказать, что всякий показатель e’ $<_e$ e является достижимым (по определению, это единственный способ доказать достижимость).

Лемма Aux. Для любых показателя e и списка es показателей, если e <all es и если AccPPs es, то AccE e.

Докажем лемму Aux трансфинитной индукцией по отношению <pps для списка es. Индуктивное предположение здесь означает:

лемма Aux выполнена для всех списков es’, для которых es’ <pps es.

Формально это выражается как

(Ind)

```
∀es' → es' <pps es →
        (∀u → u <all es' → AccPPs es' → AccE u)
```

Надо из этих условий вывести AccE e. По определению достижимости, для этого достаточно вывести то, что для любого e’ $<_e$ e выполнено AccE e’.

По лемме e:es<pps-es, из условия e<es : e <all es выводим ees<pps-es : ees <pps es

(слева от двоеточия пишем имя утверждения, справа – выражение его смысла).

Из условий e’ $<_e$ e, e <all es по транзитивности следует e’<ees : e’ <all ees.

В утверждение (Ind) подставляем es’ = ees, во второй аргумент подставляем доказательство ees<pps-es, подставляем u = e’, доказательство e’<ees подставляем в условие u <all es’. Раньше доказано, что отношение <pps является вполне заданным. Поэтому, раз AccPPs es и ees <pps es, то выполнено AccPPs ees.

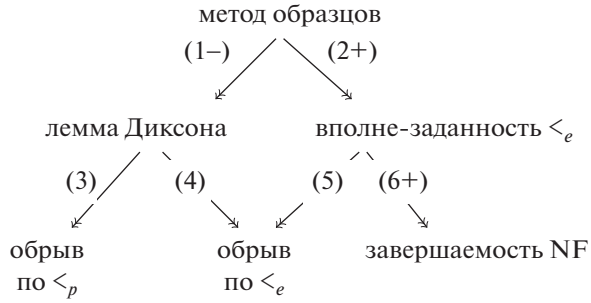

Рис. 3.

По предположению индукции, из этого следует AccE e'.

Лемма доказана.

Доказательство теоремы WF-admissible теперь получается применением леммы Aux к аргументам

e, [], [], (< pps-wellFounded []).

То есть подставляется es = []. Следующее выражение [] предсталяет доказательство того, что e меньше всех элементов пустого списка, а (<pps-wellFounded []) это доказательство достижимости пустого списка относительно упорядочения <pps.

Теперь предлагаем читателю сравнить это рассуждение с текстом функции $<_e$wellFounded.

Главными частями этого доказательства теоремы являются лемма <pps-wellFounded о вполне-заданности отношения <pps на списках показателей (которое довольно сложно устроено) и лемма e:es<pps-es об уменьшении списка es показателей в отношении <pps при добавлении показателя из *белой области* для es (определение дано в Разделе 4). Видимая простота доказательства этой теоремы достигнута тем, что от рассмотрения достижимости показателя e делается переход к рассмотрению подходящих *лестниц* es, и применяются леммы <pps-wellFounded и e:es<pps-e относительно упорядочения на этих лестницах.

Логическая зависимость по конструктивному выводу между рассматриваемыми здесь теоремами показана на рис. 3.

Здесь "обрыв по $\prec$" означает свойство обрывание-убывающей по $\prec$ цепи.

Все эти выводы могут быть выражены конструктивно. Из них знаком '+' отмечены те, которые конструктивно доказаны на языке `Agda` в нашей библиотеке.

Вклад автора состоит в следующем.

• Воплощение на языке Agda метода образцов из [5] (его видоизменения), при этом замысел метода принадлежит [5].

• Выводы (1+), (2+), (6+). (1+) получается из метода образцов путем введения упорядочения <pps на лестницах и функции доказательства с лестницей в качестве дополнительного аргумента. (2+) приведен выше в этом разделе. (6+) является простым следствием (2+) (Раздел 2.5).

• Воплощение на языке Agda различных определений, алгоритмов и лемм, которые используются в этой работе, но также входят в нашу библиотеку и в силу своей общности применимы во многих других случаях.

Отличие от подхода [9] к завершаемости алгоритма NF нормальной формы состоит в том, что [9] изначально предполагают вполне-заданность упорядочения на мономах, что ведет к неудобствам (Раздел 2, перед 2.1). Здесь же мы привели конструктивное машинно-проверяемое доказательство вполне-заданности всякого допустимого упорядочения мономов – в общем случае.

6.1. О программе доказательства в библиотеке

Точное выражение названных здесь определений, действий, функций доказательств входит в программу `AdmissiblePPO-wellFounded` [4], написанную на языке `Agda`.

Алгебра многочленов запрограммирована в модулях раздела source/Pol/ этой библиотеки, понятие показателя и допустимого упорядочения показателей выражены в модуле source/Pol/PowerProduct.agda.

Модули поддержки доказательства вполне-заданности допустимого упорядочения содержатся в разделе source/Pol/Dickson/.

Перечислим модули этого раздела библиотеки.

I.agda содержит понятия образца, уровня, вводит различные отношения на этих данных.

II.agda содержит функции сведéния образца, уровня, списка уровней, доказательства свойств сведéния для этих данных.

OfPMultiset.agda вводит понятие PMultiset (PM – мультимножество образцов), упорядочение <pm на PM, функцию сведения PM по показателю, доказательства различных свойств связи между этими сущностями, например, вполне-заданности упорядочения <pm.

ForAdmissiblePPO.agda вводит упорядочение <pps на лестницах, доказывает вполне-заданность <pps, вводит функцию esToPM построения PM из лестницы, доказывает, что каждый шаг продолжения лестницы дает меньшую лестницу в смысле <pps, доказывает вполне-заданность $<_e$wellFounded всякого допустимого упорядочения $<_e$ на показателях.

Dickson.agda содержит вывод леммы Диксона путем применения леммы о вполне-заданности упорядочения <pm.

Test.agda содержит показ вычисления последовательности PM по лестнице из трех показателей.

В модуле source/Vector/Sparse/LexOrd.agda воплощено доказательство теоремы о вполне-заданности словарного упорядочения на векторах из $\mathbb{N}^n$ для установленной размерности n для разреженного представления вектора. Главная часть всех вышеприведенных построений все-таки сводится к этой теореме.

В модуле source/Pol/NormalForm/PolNF.agda воплощена доказательная функция polNF нормальной формы относительно списка многочленов.

Библиотека проверена в системе `Agda 2.6.2.2, ghc-9.2.1, MAlonzo, Ubuntu Linux 18.04`.

Она должна работать в любой системе, где работает `Agda 2.6.2.2, MAlonzo`.

Файл install.txt содержит инструкцию по сборке программы в системе `Agda`. Команды

```
> cd source
> agda $agdaLibOpt $agdaFlags
      TypeCheckAll.agda
```

производят проверку типов всей библиотеки, в частности проверяются все доказательства.

Затем команда

```
> agda -c $agdaLibOpt $agdaFlags
Pol/Dickson/Test.agda
```

собирает исполняемый модуль для программы показа построения РМ по лестнице.

Затраты на сборку: на машине частоты 3 GHz с 8 Gb оперативной памяти каждый из этих двух вызовов ‘agda’ занимает меньше 5 минут.

Затем команда `> ./Test`

быстро вычислит три РМ по лестнице, заданной в `Test.agda`, и распечатает их (Пример из Раздела 5.1).

7. ЗАКЛЮЧЕНИЕ

Представлено формальное конструктивное машинно-проверяемое доказательство на языке Agda свойства вполне-заданности (well-founded в конструктивном определении) произвольного допустимого упорядочения $<_e$ показателей мономов многочленов нескольких переменных [8]. Оно основано на методе образцов из [5].

Нам не известны другие работы, в которых (конструктивно) доказана эта теорема.

Доказательство представлено библиотекой `AdmissiblePPO-wellFounded` [4] доказательных, разработанной автором.

На основе этой теоремы на языке Agda составлена доказательная программа алгоритма нормальной формы для многочленов [8], теорема обеспечивает простое доказательство завершаемости.

8. БЛАГОДАРНОСТИ

• Автор благодарен Антонине Н. Непейводе и Андрею П. Немытых за их замечания по содержанию статьи.

• Автор благодарен участникам семинара “Компьютерная алгебра” (факультет ВМК МГУ, руководитель С.А. Абрамов) за их обсуждение доклада по предмету этой статьи.

• Исследование частично поддержано Министерством науки и высшего образования РФ, госзадание 122012700089-0.

СПИСОК ЛИТЕРАТУРЫ

1. *Гаранина Н.О.* Общие знания в хорошо структурированных системах с абсолютной памятью // Модел. и анализ информ. Систем. 2013. Т. 20. № 6. С. 10–21.
2. *Ершов Ю.Л., Палютин Е.А.* Математическая логика. 6-е изд., испр. М.: ФИЗМАТЛИТ, 2011. 356 с. ISBN 978-5-9221-1301-4.
3. *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_27-50.pdf
4. *Мешвелиани С.Д.* AdmissiblePPO-wellFounded – программа на языке Agda формального конструктивного доказательства леммы Диксона и теоремы о вполне-заданности допустимого упорядочения на мономах многочленов. Институт программных систем РАН, Переславль-Залесский, 2022, www.botik.ru/pub/local/Mechveliani/inAgda/admissiblePPO-wellFounded.zip
5. *Martin-Mateos F.J., Alonso J.A., Hidalgo M.J., Ruiz-Reina J.L.* A Formal Proof of Dickson’s Lemma in ACL2. M.Y. Vardi and A. Voronkov (Eds.): LPAR 2003, LNAI 2850. 2003. P. 49–58.
6. Agda. A proof assistant. A dependently typed functional programming language and its system. http://wiki.portal.chalmers.se/agda/pmwiki.php.
7. *Norell U.* Dependently Typed Programming in Agda. AFP 2008: Advanced Functional Programming, Lecture Notes in Computer Science, vol. 5832, Springer, Berlin–Heidelberg, 2008. P. 230–266.
8. *Бухбергер Б.* Базисы Грёбнера. Алгоритмический метод в теории полиномиальных идеалов. В сборнике “Компьютерная алгебра”, редакторы Б. Бухбергер, Дж. Коллинз, Р. Лоос. Перевод с английского. Москва, МИР, 1986. С. 331–372.
9. *Coquand Th. and Persson H.* Gröbner Bases in Type Theory. 1998. 13 p. https://www.researchgate.net/publication/221186683_Grobner_Bases_in_Type_Theory
10. *Théry L.* A Machine-Checked Implementation of Buchberger’s Algorithm // Journal of Automated Reasoning. 2001. V. 26. P. 107–137.
11. *Romanenko S.A.* Proof of Higman’s Lemma (for two letters) Formalized in Agda. In Russian. Июль 2017. https://pat.keldysh.ru/roman/doc/talks/2017_Romanenko__Higman’s_lemma_for_2_letters_in_Agda_ru__slides.pdf Agda program for the proof:

https://github.com/sergei-romanenko/agda-Higman-lemma, in the folder Berghofer.

12. *Robbiano L.* Term orderings on the polynomial ring. Proc. EUROCAL ’85 European Conference on Computer Algebra. Linz 1985, Springer Leer. Notes Comp. Sei. 1986. V. 204. P. 513–517.
13. *Curry H.B., Feys R.* Combinatory Logic, vol I. Amsterdam, North-Holland, 1958. 417 p.
14. *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. P. 479–490.
15. *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.
16. *Per Martin-Loef.* Intuitionistic type theory. Bibliopolis, ISBN 88-7088-105-9. 1984. 91 p.
17. *Stricland Neil P.* Euclid’s theorem. An annotated proof in Agda that there are infinitely many primes. https://nextjournal.com/agda/euclid-theorem
18. *Vytiniotis D., Coquand Th., Wahlstedt D.* Stop When You Are Almost Full. Adventures in Constructive Termination // ITP 2012: Interactive Theorem Proving. 2012. P. 250–265.