Министерство науки и высшего образования Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

ВЕРИФИКАЦИЯ АЛГОРИТМА ПОСТРОЕНИЯ БАЗИСА ГРЁБНЕРА И ЕГО ПРИМЕНЕНИЙ В СИСТЕМАХ КОМПЬЮТЕРНОЙ АЛГЕБРЫ НА ЯЗЫКЕ ИНТЕРАКТИВНОГО ДОКАЗАТЕЛЬСТВА ТЕОРЕМ LEAN

Автор: Федоров Глеб Владимирович _____

Направление подготовки: 01.03.02 Прикладная математика и информатика

Квалификация: Бакалавр

Руководитель ВКР: Трифанов А.И., канд. физ.-мат. наук _____

Санкт-Петербург, 2022 г.

Обучающийся Федоров Глеб Владимирович
Группа М34351 Факультет ИТиП

Направленность (профиль), специализация
Математические модели и алгоритмы в разработке программного обеспечения

Консультанты:

а) Гилев П.А., без звания _____

ВКР принята «_____» _____ 20___ г.

Оригинальность ВКР _____%

ВКР выполнена с оценкой _____

Дата защиты «» июня 2022 г.

Секретарь ГЭК Павлова О.Н. _____

Листов хранения _____

Демонстрационных материалов/Чертежей хранения _____

Министерство науки и высшего образования Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

УТВЕРЖДАЮ

Руководитель ОП

проф., д.т.н. Парфенов В.Г. _____

«____» _____ 20___ г.

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ

Обучающийся Федоров Глеб Владимирович

Группа M34351 **Факультет** ИТиП

Квалификация: Бакалавр

Направление подготовки: 01.03.02 Прикладная математика и информатика

Направленность (профиль) образовательной программы: Математические модели и алгоритмы в разработке программного обеспечения

Тема ВКР: Верификация алгоритма построения базиса Грёбнера и его применений в системах компьютерной алгебры на языке интерактивного доказательства теорем lean

Руководитель Трифанов А.И., канд. физ.-мат. наук, ординарный доцент Университета ИТМО

2 Срок сдачи студентом законченной работы до: «31» мая 2022 г.

3 Техническое задание и исходные данные к работе

4 Содержание выпускной квалификационной работы (перечень подлежащих разработке вопросов)

5 Перечень графического материала (с указанием обязательного материала)

Графические материалы и чертежи работой не предусмотрены

6 Исходные материалы и пособия

а) -

7 Дата выдачи задания «22» октября 2022 г.

Руководитель ВКР _____

Задание принял к исполнению _____ «22» октября 2022 г.

Министерство науки и высшего образования Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

АННОТАЦИЯ
ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЫ

Обучающийся: Федоров Глеб Владимирович

Наименование темы ВКР: Верификация алгоритма построения базиса Грёбнера и его применений в системах компьютерной алгебры на языке интерактивного доказательства теорем lean

Наименование организации, в которой выполнена ВКР: Университет ИТМО

ХАРАКТЕРИСТИКА ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЫ

1 Цель исследования: Разработать программное обеспечение на языке lean4, вычисляющее базис Грёбнера. Код данного программного обеспечения должен быть верифицирован на том же языке.

2 Задачи, решаемые в ВКР:

а) Реализация упорядочения lex и grlex для мономов. Доказательство, что реализованные доказательства являются линейными упорядочениями на множестве мономов;

б) Реализация алгоритма деления. Доказательство корректности алгоритма;

в) Реализация алгоритма построения базиса Грёбнера(алгоритм Бухбергера). Доказательство корректности алгоритма;

г) Реализация возможности пользовательского взаимодействия с кодом.

3 Число источников, использованных при составлении обзора: 0

4 Полное число источников, использованных в работе: 0

5 В том числе источников по годам:

Отечественных			**Иностранных**		
Последние 5 лет	От 5 до 10 лет	Более 10 лет	Последние 5 лет	От 5 до 10 лет	Более 10 лет
0	0	0	0	0	0

6 Использование информационных ресурсов Internet: нет

7 Использование современных пакетов компьютерных программ и технологий:

Пакеты компьютерных программ и технологий	**Раздел работы**
Пакет `tabularx` для чуть более продвинутых таблиц	**??**, Приложения А, **??**
Пакет `biblatex` и программное средство `biber`	Список использованных источников

8 Краткая характеристика полученных результатов

9 Гранты, полученные при выполнении работы

10 Наличие публикаций и выступлений на конференциях по теме выпускной работы

-

Обучающийся Федоров Г.В. _____

Руководитель ВКР Трифанов А.И. _____

«____» _____ 20___ г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В данном разделе размещается введение.

ГЛАВА 1. ИНСТРУМЕНТ ИНТЕРАКТИВНОГО ДОКАЗАТЕЛЬСТВА ТЕОРЕМ LEAN4

В данной главе будет приведён обзор интерактивного помощника доказательства теорем lean4.

1.1. Введение

Язык lean – это функциональный язык программирования, разработанный Microsoft Research. До четвёртой версии данный языке данный язык можно было использовать, в основном, только как интерактивный помощник доказательства теорем, но сейчас он может быть применим как язык общего назначения. Доказательством этому служит тот факт, что текущая версия компилятора lean написана на lean.

Данный язык, помимо Microsoft, поддерживает довольно большое сообщество разработчиков и математиков. Так, например, на lean3 была разработана библиотека mathlib – проект по формализации математики, который позволяет переиспользовать различные математические факты в своей программе.

На момент написания данной работы lean4 находится в финальной стадии разработки, и уже пригоден для использования. Библиотека mathlib так же находится в финальной стадии переписывания на четвёртую версию.

1.2. Основания

Базисом для системы типов языка lean является исчисление конструкций – теории типов на основе λ-исчисления высшего порядка с зависимыми типами...

1.3. Доказательство в lean4

В математической логике доказательством корректности некоторого утверждения Ψ является последовательность из контекста, аксиом и утверженний, полученных при помощи заранее зафиксированных правил вывода, которая оканчивается утверждением Ψ. На этом принципе основываются все системы интрактивного доказательства теорем.

Доказательство в lean начинается с ключевого слова by, которое переводит язык в режим тактик.

В данном режиме в среде разработки появляется окно, в котором описано текущее состояние доказательства, разделёное на две части – контекст и

текущую цель. В листинге 1 приведены две фукнции f, принимающую два натуральных числа и доказательство того, что их сумма больше пяти, и g, вызывающую функцию f с аргументами 6, x и доказательством того, что $6 + x > 5$.

Листинг 1 – Пример доказательства

```
def f (x y: Nat) (h: x + y > 5) := x
def g (x: Nat): Nat := f 6 x (
                                  by
                                    induction x
                                    simp
                                    rename_i n h
                                    simp at h
                                    simp [Nat.add_succ]
                                    rw [Nat.succ_eq_add_one]
                                    have six_add_n_lt_y : 6 + n < 6 +
                                      n + 1
                                    simp
                                    exact Nat.lt_trans h
                                      six_add_n_lt_y
                                )
------------
Tactic state
x: ℕ
⊢ 6 + x > 5
```

Для того, чтобы переиспользовать доказательства, используется ключевое слово theorem. Пример такого переиспользования доказательсва – теорема $Nat.add\_succ$ из из стандартной библиотеки lean, определение которой можно увидеть в листинге 2.

Листинг 2 – Примеры теорем из стандартной библиотеки lean

```
theorem succ_add : ∀ (n m : Nat), (succ n) + m = succ (n + m)
  | _, 0   => rfl
  | n, m+1 => congrArg succ (succ_add n m)

theorem add_succ (n m : Nat) : n + succ m = succ (n + m) :=
  rfl
```

Правилами вывода служат тактики. Опишем некоторые из них на примере доказательства из листинга 1.

Наиболее частым типом доказательства является доказательство того, что объекты A и B находятся в некотором отношении. Если отношение рефлексивно, и объект A возможн тривиальным образом перестроить в объект B, то тактика rfl закроет цель.

Тактика induction использует метод математической индукции по переданному ей аргументу. Результат применения тактики можно увидеть в листинге 2.

Листинг 3 – induction

```
Tactic state
case zero
⊢ 6 + Nat.zero > 5
case succ
n+: ℕ
n_ih+: 6 + n+ > 5
⊢ 6 + Nat.succ n+ > 5
```

Если в результате применения тактики появляются новые переменные, lean даст им имена автоматически. Обычно, полученные имена – это имена переменных из применённой теоремы или функции. Для того, чтобы доказательство не ломалось при каких-либо изменениях имён вне текущей теоремы, автоматически сгенерированные имена запрещено использовать, что символизирует крест после сгенерированного имени. Чтобы явно использовать переменную в доказательстве её нужно переименовать, для чего используется тактика rename_i.

Тактика simp упрощает текущую цель и закрывает результат упращения, если это возможно. Для упрощения используются все теоремы, помеченные тего @[simp]. Так, цель $6 + Nat.zero > 5$ упрощается в $6 > 5$ что, очевидно, верно. Данная тактика может принимать аргументы, которые она так же использует для упрощения.

Тактика rw переписывает текущую цель при помощи переданных ей аргументов. Аргументами могут быть, например, теоремы и функции. Так, цель $\vdash 5 < Nat.succ(6 + n)$ при помощи $rw[Nat.succ\_eq\_add\_one]$ будет переписана в $\vdash 5 < 6 + n + 1$.

Тактика have связывает тип и/или значение с именем. Так мы ввели утверждение have six_add_n_lt_y : 6 + n < 6 + n + 1, после чего доказали его тактикой simp.

Тактика exact пытается закрыть цель при помощи переданной ей теоремы. В листинге 1 мы закрыли цель по транзитивности отношения $<$ на натуральных числах – exact Nat.lt_trans h six_add_n_lt_y.

В данном параграфе были описаны далеко не все тактики, использованные в данной работе. Подробное описание всех тактик в языке lean можно найти в .....

Выводы по главе 1

Вывод: в данной главе было дано описанию языку lean, процессу доказательсва и программирования на нём.

ГЛАВА 2. ТЕОРИЯ БАЗИСОВ ГРЁБНЕРА

Данная глава будет посвящена введению основных понятий теории колец от нескольких переменных.

2.1. Основные определения

Будем называть вектором степеней конструкцию следующего вида

$$\alpha = (\alpha_1 \dots \alpha_n), \alpha_i \in \mathbb{N}. \tag{1}$$

Назовём вектором переменных следующий вектор

$$x = (x_1 \dots x_n). \tag{2}$$

Мономом от переменных $x_1 \dots x_n$ называется конструкция следующего вида

$$x^\alpha = (x_1^{\alpha_1} \dots x_n^{\alpha_n}). \tag{3}$$

Полиномом f, с коэффициентами из поля K, называется конечная линейная комбинация мономов, которая записывается следующим образом

$$f = \sum_\alpha c_\alpha * x^\alpha, c_\alpha \in K. \tag{4}$$

Множеством всех полиномов от переменных $x_1 \dots x_n$ над полем K будем обозначать как $K[x_1 \dots x_n]$. Отметим, что на данном множестве можно естественным образом ввести операции $+$ и $*$ таким образом, чтобы структура $\langle K[x_1 \dots x_n], +, * \rangle$ удовлетворяла аксиомам кольца.

Подмножество $I \subset K[x_1 \dots x_n]$ называется идеалом, если выполнены следующие условия:

а) $0 \in I$;

б) если $f, g \in I$, то $f + g \in I$;

в) если $f \in I$ и $h \in K[x_1 \dots x_n]$, то $hf \in I$.

Пусть $f_1 \dots f_s$ – набор полиномов от нескольких переменных, тогда множество $\langle f_1 \dots f_s \rangle = \{\sum_i^s h_i * f_i | h_1 \dots h_s \in K[x_1 \dots x_n]\}$ является идеалом в $K[x_1 \dots x_n]$, а полиномы $\langle f_1 \dots f_s \rangle$ называются образующими идеала.

2.2. Деление полиномов от одной переменной

Теория базисов Грёбнера во многом опирается на операцию деления многочленов. Перед тем, как определить алгоритм деления в кольце полиномов от нескольких переменных рассмотрим алгоритм в кольце полиномов от одной переменной.

Важнейшей частью алгоритма является понятие старшего члена полинома. Пусть $f = \alpha_0 x^m + \alpha_1 x^{m-1} \ldots + a_m$, где $a_i \in K$, $a_0 \neq 0$. Тогда

$$LT(f) = \alpha_0 x^m$$

называется старшим членом полинома f.

Опишем алгоритм деления в $K[x]$. Пусть $g \in K[x]$ – ненулевой полином. Тогда любой полином $f \in K[x]$ может быть записан в виде

$$f = qg + r,$$

где $q, r \in K[x]$ и либо $r = 0$, либо $deg(r) < deg(g)$, причём q и r определены однозначно. Многочлены q и r могут быть найдены следующим алгоритмом.

Листинг 4 – Деление в $K[x]$

function Divide(q, f)
    $q = 0$;
    $r = f$;
    while $r \neq 0$ **&** $LT(g)|LT(r)$ **do**
        $q = q + LT(r)/LT(g)$
        $r = r - (LT(r)/LT(g))g$
    end while
    return q, r
end function

Доказательство корректности данного алгоритма можно найти в ...

2.3. Упорядочения мономов

После прочтения предыдущего параграфа может сложится впечатление, что алгоритм полиномов из $K[x]$ будет работать и в $K[x_1 \ldots x_n]$. К сожалению, это не совсем так. Заметим, что в кольце $K[x]$ у мономов есть естественный порядок – по степеням, которые являются натуральными числами. Но в

$K[x_1 \dots x_n]$ степень монома – не число. Поэтому, в данном параграфе будет дано определение упорядочения мономов в $K[x_1 \dots x_n]$.

Мономиальным упорядочением на $K[x_1 \dots x_n]$ называется любое бинарное отношение $\leqslant$ на $\mathbb{Z}^n_{\geqslant 0}$, обладающее следующими свойствами:

а) $\leqslant$ является линейным упорядочением на $\mathbb{Z}^n_{\geqslant 0}$;

б) если $\alpha \leqslant \beta$ и $\gamma \in \mathbb{Z}^n_{\geqslant 0}$, то $\alpha + \gamma \leqslant \beta + \gamma$;

в) $\leqslant$ вполне упорядочивает $\mathbb{Z}^n_{\geqslant 0}$.

Условие а нужно для того, чтобы мы могли для любого полинома расположить мономы в порядке $\leqslant$. То есть, для любой пары мономов x^α, y^β должно выполняться одно из следующих соотношений

$$x^\alpha < y^\beta, x^\alpha = y^\beta, x^\alpha > y^\beta$$

.

Условие б нужно для того, чтобы упорядоченность мономов была согласована с аксиомами кольца. Заметим, что задача умножения полинома на полинома естественным образом сводится к задаче умножения полинома на моном. Но, если упорядочение не удовлетворяет свойству б, то умножение монома на старший член полинома не будет являться старшим членом. Примером упорядочения, не удовлетворяющего свойству б, может служить max упорядочение: $\leqslant_{max}$: $\alpha \leqslant_{max} \beta \Leftrightarrow max(\alpha) \leqslant max(\beta) \vee max(\alpha) = max(\beta) \wedge \alpha \leqslant_{lex} \beta$. Тогда $x^2y^3 \leqslant_{max} xyz^5$, но $x^2y^3 * y^8 \geqslant_{max} xyz^5 * y^8$.

Условие в необходимо для доказательства корректности алгоритмов их следующих параграфов. А именно, критерий остановки остановки алгоритма будет основан на том, что старший член полинома убывает на каждом шаге алгоритма.

В данной работе будут рассмотрены два упорядочения:

а) Лексикографическое упорядочение – $a \leqslant_{lex} b \Leftrightarrow$ первая ненулевая координата вектора $b - a$ положительна;

б) Градуированное лексикографическое упорядочение – $a \leqslant_{grlex} b \Leftrightarrow$ $|a| < |b| \vee (|a| = |b| \wedge a \leqslant_{lex} b)$.

Доказательства того, что эти упорядочения удовлетворяют условиям, определённым выше, будут предъявлены в следующей главе.

2.4. Алгоритм деления полиномов от нескольких переменных

Перейдём к алгоритму деления полиномов от нескольких переменных. Зафиксируем некоторое мономиальное упорядочение $\leqslant$. Пусть $F = (f_1 \dots f_s)$ – упорядоченный набор полиномов из $k[x_1 \dots x_n]$. Тогда любой полином $f \in k[x_1 \dots x_n]$ может быть представлен в виде

$$f = \alpha_1 f_1 + \dots + \alpha_s f_s + r,$$

где $a_i, r \in k[x_1 \dots x_n]$, причём либо $r = 0$, либо r есть линейная комбинация мономов, ни один из которых не делится ни на один из старших членов $LT(f_1) \dots LT(f_s)$. Данно представление можно найти следующим алгоритмом:

Листинг 5 – Деление в $K[x]$

function DivideMany(f, $f_1 \dots f_s$)
    $a_1 = 0 \dots a_s = 0$;
    $r = 0$;
    $p = f$;
    while $p \neq 0$ **do**
        $i = 1$;
        $hasDiv = false$;
        while $i \leqslant s \wedge !hasDiv$ **do**
            if $LT(f_i) \mid LT(P)$ **then**
                $a_i = a_i + LT(p)/LT(f_i)$
                $p = p - (LT(p)/LT(f_i))f_i$
            else
                $i = i + 1$
            end if
        end while
        if $!hasDiv$ **then**
            $r = r + LT(p)$
            $p = p - LT(p)$
        end if
    end while
    return $a_1 \dots a_s, r$
end function

Алгоритм деления в $k[x_1 \dots x_n]$ во многом похож на алгоритм деления в $k[x]$. Основная разница, помимо упорядочения мономов, состоит в том, что мы делим не на один полином, а сразу на несколько. Поэтому, мы берём старший

член p в остаток только в том случае, когда он не делится ни на один из старших членов $LT(f_1) \dots LT(f_s)$.

Для доказательства корректности алгоритма необходимо:

а) На каждом шаге выполняется равенство: $f = \sum_i^s a_i f_i + p + r$;

б) Ни один из мономов из r не делится ни на один из старших членов $LT(f_1) \dots LT(f_s)$;

в) Алгоритм завершает свою работу.

Доказательства данных пунктов и реализация алгоритма будут приведены в главе ”Реализация”.

2.5. Алгоритм Бухбергера

2.6. Решение задачи о принадлежности многочлена идеалу

Выводы по главе 2

Вывод:

ГЛАВА 3. РЕАЛИЗАЦИЯ

Данная глава будет посвящена введению реализации описанной выше теории на языке интерактивного доказательства теорем lean. В данной работе будут рассматриваться только полиномы над полем рациональных чисел.

3.1. Реализация полинома

В библиотеке mathlib уже есть реализация полинома от нескольких переменных под названием MvPolynomial, которая удовлетворяет всем аксиомам кольца. Но, к сожалению, это реализация явно использует аксиому выбора, что делает её неконструктивной. Иначе говоря, её возможно использовать только для доказательства теорем, но сгенерировать исполняемый код при использовании данной реализации использовании не выйдет. Поэтому в данной работе была написана собственная реализация.

Введём основные определения. Вектором степеней назовём вектор натуральных чисел длины n упорядоченный согласно мономиальному упорядочению ord. Произведением двух векторов одинаковой длины с одинаковым упорядочением назовём их покомпонентную сумму. Возможная путаница в терминалогии возникает из-за того, что данная операция будет использована далее при определении умножения монома на моном, а именно: $x_1^{\alpha_1} \dots x_n^{\alpha_n} * x_1^{\beta_1} \dots x_n^{\beta_n} = x_1^{\alpha_1+\beta_1} \dots x_n^{\alpha_n+\beta_n}$.

Листинг 6 – Вектор степеней

```
def Variables (n: Nat) (ord: Type) := Vector Nat n

def Variables.mul (v₁ v₂: Variables n ord): Variables n ord :=
  map₂ (fun x y => x + y) v₁ v₂
```

Моном определяется как пара из рационального числа и вектора длины n с упорядочением ord.

Листинг 7 – Моном

```
def Monomial (n: Nat) (ord: Type) := Rat × (Variables n ord)

def Monomial.mul (m₁ m₂: Monomial n ord) : Monomial n ord :=
  (m₁.fst * m₂.fst, Variables.mul m₁.snd m₂.snd)
```

Данные определения находятся в файле PolynomialCommon.lean.

Полином был реализован на красно-чёрном дереве, элементами которого являются мономы с числом переменных n, упорядоченных согласно ord, причём для ord обязан существовать экземпляр MonomialOrder (Variablesn ord n), иначе говоря, ord обязан быть мономиальным упорядочением. Функция m_cmp, определённая в файле MonomialOrderInterface.lean, позволяте использовать упорядочение ord как функцию сравнения для мономов. Подробнее об MonomialOrder будет написано в следующей главе.

Листинг 8 – Полином

```
def Polynomial (n: Nat) (ord: Type)
    [MonomialOrder $ Variables n ord] :=
    Std.RBSet (Monomial n ord) ordering.m_cmp
```

Все функции и экземпляры классов типов для полинома и монома определены в файле Polynomial.lean.

В процессе работы не получилось доказать, что данное определение полинома с операцияму $+$ и $*$ удовлетворяют аксиомам кольца из-за довольно сложного изменения структуры красно-чёрного дерева при применении данных операций к полиномам. Для того, чтобы было возможно проводить какие-либо доказательства, аксиомы кольца были постулированы при помощи команды axiom, что можно увидеть в файле PolynomialRing.lean.

3.2. Мономиальные упорядочения

Для доказательства того, что реализованные упорядочения удовлетворяют аксиомам мономиального упорядочения, был реализован класс типов MonomialOrder, наследованный от класса типов LinearOrder(линейное упорядочение) и класса типов WellFoundedRelation(вполне упорядочивание)

В данной работе были определены два мономиальных упорядочения: лексикографическое(далее Lex) и градуированное лексикогрфическое(далее GrLex).

Ниже представлено доказательство рефлексивности лексикографического упорядочения, проведённое методом индукции по конструктору типа Variables.

Доказательства остальных свойств для Lex и GrLex упорядочений находятся в файле MonomialOrder.lean.

Листинг 9 – Lex упорядочение

```
def Order.lex_impl (v₁ v₂: Vector Nat n): Prop :=
  match v₁, v₂ with
    | ⟨[], _⟩, ⟨[], _⟩ => True
    | ⟨x :: _, _⟩, ⟨y :: _, _⟩ => if x = y then lex_impl v₁.tail v₂.tail
                                  else x ⩽ y

def Order.lex (v₁ v₂: Variables n order.Lex): Prop := Order.
   lex_impl v₁ v₂
```

Листинг 10 – Доказательство рефлексивности

```
theorem lex_le_refl : ∀ (a : Variables n order.Lex), Order.lex a a
    := by
  intro a
  let rec aux (m: Nat) (v: Variables m order.Lex) : Order.lex_impl
      v v := by
    match v with
      | ⟨[], p⟩ => rw [Order.lex_impl]
                   split
                   simp at *
                   simp at p
                   simp at *
      | ⟨x :: xs, _⟩ => rw [Order.lex_impl]
                        split
                        simp
                        simp at *
                        simp at *
                        simp [Nat.le_refl]
                        rename_i x₁ _ x₂ _ h₁ h₂
                        have h₃ := Eq.symm h₁.left
                        have h₄ := Eq.symm h₂.left
                        rw [h₃, h₄]
                        simp [Nat.le_refl]
                        apply aux (m-1) (tail ⟨x :: xs, _⟩)
```

Листинг 11 – GrLex упорядочение

```
def Order.grlex (vs₁ vs₂: Variables n order.GrLex): Prop :=
  let sum₁ := elem_sum vs₁
  let sum₂ := elem_sum vs₂
  if sum₁ < sum₂ then True
  else if sum₁ = sum₂ then if Order.lex vs₁ vs₂ then True
                           else False
  else False
  where
    elem_sum (vs: Variables n order.GrLex): Nat :=
      List.foldl (fun x y => x + y) 0 vs.toList
```

Помимо основных свойств линейного упорядочения, а именно рефлексивности, транзитивности, антисимметричности и требования, чтобы любые два элемента были сравнимы, lean требует ещё два. А именно:

а) Отношение $\leqslant$ должно быть вычислимым(decidable);

б) Lean автоматически строит отношение $<$. Поэтому, нужно проверка согласованности отношений $<$ и $\leqslant$. А именно – $a < b \Leftrightarrow (a \leqslant b \wedge b \nleqslant a)$.

Остановимся подробнее на первом. Как было сказано выше, в lean есть две основных вселенных типов – Type и Prop. Заметим, что свойства линейного упорядочения – это математические утверждения, то есть они принадлежат типу Prop. Следовательно, утверждение, что $\alpha \leqslant \beta$ так же принадлежит типу Prop. Но математическое утверждение не обязано быть разрешимым. Примером тому может служить проблема останова. Но было бы довольно не практично, если бы некоторое линейное упорядочение было неразрешимыми, ведь сразу перестаёт работать множество теорем и тактик для доказательства свойств каких-либо объектов, использующих данное упорядочение. Поэтому lean дополнительно требует доказательство того, что упорядочение разрешимо.

Чтобы доказать разрешимость можно воспользоваться следующим трюком – по пропозициональной формуле ϕ построим выражение ψ, возвращающую значение, имеющее тип из вселенной Type, после чего докажем, что если ψ возвращает x, то x удовлетворяет формуле ψ. В случае с упорядочениями, ϕ – это утверждения, приведённые в листингах 6 и 7, а ψ – это аналогичные функции, возращающие Bool вместо Prop. Доказательства данного свойства для lex и grlex упорядочений можно найти в файле MonomialOrder.lean.

3.3. Деление полиномов от нескольких переменных

1) Информация о результате деления должна быть отображена в типе, которы й возвращает функция деления

2)

3.4. Алгоритм Бухбергера

3.5. Консольная утилита

Одной из целей данной работы была реализация взаимодействия пользователя с программой. Для выполнения данной цели балы написана простая консольная утилита.

Утилита работает следующим образом: до тех пор, пока не была введена команда exit, программа будет ожидать пользовательский ввод. После того, как пользователь ввёл строку, заканчивающуюся символом перевода строки, происходит парсинг команды и её аргументом. Если команда и её аргументы были успешно распаршены, начинается выполнение команды, иначе – пользователь получит сообщение об ошибке.

Поддержаны следующие команды для работы с полиномами:

а) set_n – устанавливает число переменных в полиномах. По умолчанию работа происходи с полиномами от трёх переменных;
б) simp – принимает название мономиального упорядочения ord и набор полиномов. Возвращает набор упрощённых полиномов, упорядоченных согласно ord;
в) is_in – принимает многочлен p и набор полиномов ps. Проверяет, принадлежит ли полином p идеалу $\langle ps \rangle$;
г) groebner – принимает название мономиального упорядочения ord и набор полиномов ps. Возвращает базис Грёбнера идеала $\langle ps \rangle$ для ord упорядочения.

Реализация и парсинг команд находится в файле Interactive.lean. Пользовательский ввод-вывод реализован в файле Main.lean. Парсинг многочленов реализован в файле Parser.lean.

3.6. Тестирование

В языке lean нет стандарный средств для тестирования кода. Поэтому были написаны стандартные функции, используемые при тестировании – AssertEq, проверяющая, возвращающая строку Ok, если переданные ей аргументы равно, AssertNEq, возвращающая Ok, если аргументы не равны и AssertTrue, возвращающая Ok, если переданный ей булевый аргумент имеет значение true.

В работе тестировались: мономиальные упорядочения, парсинг полиномов, операции сложения, умножения, деления и алгоритм построения базиса Грёбнера. Тестирование проводилось на полиномах от трёх переменных.

В параграфе Консольная утилита была упомянута возможность указывать число переменных. Она так же была протестирована.

Все тесты и вспомогательные функции находятся в директории tests.

Выводы по главе 3

Вывод:

ЗАКЛЮЧЕНИЕ

В данном разделе размещается заключение.

ПРИЛОЖЕНИЕ А. ПРИМЕР ПРИЛОЖЕНИЯ