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 M34351 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

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. Синтаксис термов в данном исчислении может быть записан в следующем виде:

$$e ::= T|P|x|e\ e|\lambda x : e.e|\forall x : e.e,$$

где

а) T – это типы.
б) P – это тип, к которому относятся все утверждения.
в) x – переменная.

Исчисление конструкций позволяет строить суждения о типах. Так, например, выражение

$$x_1 : \alpha_1 \dots x_n : \alpha_n \vdash y : \beta$$

можно прочесть как импликацию: если переменные $x_1 \dots x_n$ имеют типы $\alpha_1 \dots \alpha_n$, то терм t имеет тип β.

Суждение Ψ является допустимым, если из контекста, аксиом и утвержений, полученных при помощи правил вывода, можно вывести Ψ. Подробнее о правилах вывода в исчислении конструкций можно прочесть в книге Бенжамина Пирса "Типы в языках программирования. 2009"на странице 357.

Важным свойством исчисления конструкций является наличие зависимых типов – типов, которые зависят от некоторого значения. Примером такого типа в lean является Vector, который зависит от натурального числа. То есть Vector n – это вектор размерности n. Наличие зависимых типов позволяет писать программы, которые корректны по построению. Например, можно реализовать функцию sort, которая имеет тип $Vector\ n \to Vector\ n$, который показывает, что функция возвращает вектор той же длинны, что и на входе. Более того, можно написать функцию sort таким образом, чтобы её тип говорил о том, что возвращаемый вектор отсортирован. Но, к сожалению, написаний подобных функций, обычно, довольно сложно. Ведь, помимо реализации самого алгоритма необходимо предоставить полный вывод того, что суждение о возвращаемом типе является допустимым.

1.3. Процесс доказательства в lean4

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 доказательства(называемую так же Tactic state) и текущую цель доказательства. 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 $6 + x > 5$.

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.

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 ре-

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
```

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
```

флексивно, 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, и отобразит их в Tactic state. 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 использовать запрещено, что символизирует крест в конце имени. Чтобы явно использовать переменную в доказательстве её нужно переименовать, для чего используется тактика rename_i.

Тактика simp преобразует текущую цель при помощи всех теорем, помеченных тегом @[simp], и закрывает её, если возможно. Так, цель $6 + Nat.zero > 5$ упрощается в $6 > 5$ что, очевидно, верно. Данная тактика может принимать аргументы, которые она так же использует в процессе упрощения.

Тактика rw переписывает текущую цель при помощи переданных ей аргументов, которыми 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 основных тактик в языке lean можно найти на сайте URL: https://leanprover.github.io/theorem_proving_in_lean4/tactics.html.

1.4. Программирование в lean4

Процесс программирования на lean очень похож на программирование на многих других функциональных языках программирования, таких как Haskell и OCaml. В данном параграфе буде дано описание основных конструкций, использованных в этой работе.

Для описания объектов предметной области, удобно группировать объекты разных типов в один. Данная функциональность реализована при помощи структур, которые объявляются при помощи ключевого слова structure. Пример объявления структуры можно увидеть в листинге ...

Листинг 4 – Point

```
structure Point where
    x: Nat
    y: Nat
```

В lean структура может не содержать ни единого поля. При этом structure A и structure B будут разными типами. Структуры без полей удобно использовать в качестве меток для своих типов. Например, в данной работе, такие структуры используются для выбора способа упорядочения мономов в полиноме.

Наконец, тип поля может быть сколь угодно сложным. Например, в листинге ... представленна структура, содержашее натуральное число x и доказательство того, что $x > 5$. Благодаря такой возможности, становится более удобным формулировать спецификацию типа функции.

Листинг 5 – Difficult type example

```
structure T where
    x: Nat
    h: x > 5
```

В языке lean присутсвует способ наложения ограничений на тип, известный как класс типов(далее typeclass). Рассмотрим, например, функцию print, которая выводит строку на экран. Тогда, для того, чтобы вывести значение своего типа MyType на экран, необходимо, сначала преобразовать его в строку, а потом вызвать функцию print. Либо, создать свою функцию print, которая принимает не строку, а MyType, переводит его в строку, а потом выводит. В обоих реализациях есть минус – нужно писать много дополнительного кода. Этого можно избежать, если сделать функцию print полиморфной, а на тип принимаемого аргумента наложить ограничение, что для него существует экземпляр класса типа ToString.

Описание класса типа ToString и пример создания его экземпляра можно увидеть в листинге ...

Свойства, которые могут быть наложены при помощи typeclass могут быть самыми разными. Например, в данной работе используется класс типов

Листинг 6 – Print

```
class ToString (α: Type) where
    toString x : α → String

structure MyType

instance: ToString MyType where
    toString x := "Hello, MyType"

def print [ToString α](x: \alpha) := ...
```

LinearOrder α, который говорит, что для типа α существует линейное упорядочение.

Выводы по главе 1

Вывод: в данной главе был проведён обзор теории, которая стоит за системой типов, а так же были описаны основный аспекты доказательства и программирования на язык lean4.

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 7 – Деление в $K[x]$

```
function Divide(q, f)
    q = 0;
    r = f;
    while r ≠ 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 данного алгоритма можно найти в книге ”Кокс Д., Литтл-Дж., О’ШиД. Идеалы, многообразия и алгоритмы. Введение в вычисли- вычислительные аспекты алгебраической геометрии и коммутативной алгебры. 2000. С.56”. Формального доказательства данного алгоритма не проводилось, так как он является частным случаем алгоритма деления в $K[x_1 x_n]$.

2.3. Упорядочения мономов

После прочтения предыдущего параграфа может сложиться впечатление, что алгоритм деления полиномов из $K[x]$ будет работать и в $K[x_1 \dots x_n]$. 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

$$f = \alpha_1 f_1 + \ldots + \alpha_s f_s + r, \tag{5}$$

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:

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. Базисы Грёбнера и алгоритм Бухбергера

Дадим определение базису Грёбнера. Пусть зафиксированно мономиальное упорядочение. Тогда множество $g_1 \dots g_m \subset I$ называется базисом Грёбнера идеала I в том и только в том случае, когда старший член любого полинома $p \in I$ делится хотя бы на один старший член $LT(g_i)$.

Листинг 8 – Деление в $K[x]$

```
function DivideMany(f, f_1 ... f_s)
    a_1 = 0 ... a_s = 0;
    r = 0;
    p = f;
    while p ≠ 0 do
        i = 1;
        hasDiv = false;
        while i ⩽ s∧!hasDiv do
            if LT(f_i) | 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 ... a_s, r
end function
```

Пусть $f, g \in K[x_1 \ldots xn]$–ненулевые полиномы. Тогда, назовём S-полиномом следующую конструкцию:

$$S(f,g) = \frac{x^{\gamma}}{LT(f)} * f - \frac{x^{\gamma}}{LT(g)} * g, \tag{6}$$

где x^{γ} – наименьшее общее кратное мономов $LT(f)$ и $LT(g)$. S-полином специально сконструирован для сокращения старших членов полиномов, что показано в лемме 5 в книге...

Критерий Бухбергера. Пусть I - полиномиальный идеал. Тогда базис $G = g_1 \ldots g_s$ идеала I является базисом Грёбнера в том и только в том случа, когда для всех пар $i \neq j$ остаток от деления $S(g_i, gj)$ равен нулю. Доказательство данного критерия можно найти в теореме 6...

Из данного критерия следует алгоритм построения базиса Грёбнера, псевдокод которого приведён в листинге 6.

Листинг 9 – Алгоритм Бухбургера

```
function BuildGroebner(F = f_1 ... f_s)
    G = F;
    while True do
        G' = G;
        for ∀⟨p, q⟩, p ≠ q ∈ G' do
            S = S(p,q) mod G';
            if S ≠ 0 then
                G = G ∪ S
            end if
        end for
        if G == G' then break;
        end if
    end while
    return G
end function
```

Во-первых – покажем, что идеал, порождённый множеством $F = f_1 \dots f_s$, и идеал, порождённый множеством, которое возвращает функция BuildGroebner совпадают. Заметим, что остаток от деления S-полинома на базис идеала I лежит в I. Дествительно, в формуле 5 $f \in I$, а множитель, назовём его α, при f лежит в кольце $K[x_1 \dots x_n]$. Тогда, по определению идеала $\alpha * f \in I$. Для g справедливы те же самые рассуждения. Тогда, из замкнутости идела относительно операции сложения, S-полином от f и g лежит в I. Далее, при помощи алгоритма деления в $K[x_1 \dots x_n]$, перепишем S-полином в виде формулы 5:

$$S(f,g) = a_1 g_1 + \ldots + a_s g_s + r.$$

Так как, $S(f,g) \in I$ и комбинация $a_1 g_1 + \ldots + a_s g_s \in I$, то и $r \in I$, что и требовалось показать. Формальное доказательство данного факта будет приведено в главе ”Реализация”.

Во-вторых – нужно показать, что данный алгоритм завершает свою работу. В данной работе это не будет формально доказано, о чём будет подробнее сказано в главе ”Реализация”. С исходным доказательством можно ознакомиться в ...

2.6. Задача принадлежности идеалу

Выводы по главе 2

Вывод: в данной главе был приведён обзор основной теории построения базисов Грёбнера.

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 10 – 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 11 – 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 упорядоченим. 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 12 – 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 операциямy $+$ 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. Полиномиальные идеалы

В данной работе была взята реализация идеала из библиотеки mathlib. Но, так как, во всех алгоритмах идёт работат со списками, нужно было дополнительно доказать несколько теорем.

Во-первых, нужно было показать, что список полиномов, в некотором смысле, эквивалентен множеству.

Во-вторых, нужно было реализовать функцию, которая по списку полиномов строит идеал.

В-третьих, нужно было показать, что исходный списко содержится в построенном идеале.

Возможно, в процессе прочтения данного параграфа, возник вопрос – почему бы сразу не использовать множества, а не списки. Ответ простой – множества в lean, это функция, которая возвращает по элементу x значение типа Prop. Иначе говоря, множество – это некоторый предикат. А раз так, то

Листинг 13 – asSet

```
def asSet [MonomialOrder (Variables n ord)]
         (ps: List (Polynomial n ord)): Set (Polynomial n ord) :=
            {x | x $\in$ ps }

theorem listInSet [MonomialOrder (Variables n ord)]
                 (l: List (Polynomial n ord)): $\forall$ x $\in$
                    l, x $\in$ asSet l := by
  intros y h
  simp [asSet]
  exact h
```

Листинг 14 – asIdeal

```
def asIdeal [MonomialOrder (Variables n ord)]
           (l: List (Polynomial n ord)):
           Ideal (Polynomial n ord) := Ideal.span (asSet l)
```

Листинг 15 – listInIdeal

```
theorem listInIdeal [MonomialOrder (Variables n ord)]
                   (l: List $ Polynomial n ord): $\forall$ x $\
                      in$ l, x $\in$ asIdeal l := by
  intros y h
  rw [asIdeal, asSet]
  have in_set := listInSet l y h
  rw [asSet] at in_set
  apply Ideal.subset_span
  exact in_set
```

пришлось бы отдельно доказывать вычислимость фукнции принадлежности, которая необходима для перебора полиномов в алгоритмах. Подробнее об вычислимости будет сказано в следующем параграфе.

Отдельно хочется сказать, что идеал в mathlib реализован как подмодуль полукольца R по R. Поэтому, нётеревость колец(условие обрыва возрастающих цепочек идеалов, далее УОВЦ) формулируется иначе. А именно, что каждый подмодуль конечно порождён. Так как УОВЦ, по-сути, отсутсвует, то адаптировать оригинальное доказательство того, что алгоритм Бухбергера завершает свою работу не вышло.

3.3. 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).

Листинг 16 – 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
```

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

Листинг 17 – 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 18 – 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
```

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

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.

3.4. Деление полиномов от нескольких переменных

Для того, чтобы показать, что результат деления обладает свойствами, определёнными в предыдущей главе, был реализован тип DivisionResult, обладающий следующими полями:

а) p – частное от деления. Так как для построени базиса Грёбнера нужно уметь получать только остатки, то для простоты доказательства, будем считать частным упрощённый полином $a_1g_1 + \ldots + a_sg_s$;
б) r – остаток от деления.
в) sum_eq – утверждение, что сумма частного и остатка равна делимому;
г) p_in_ideal – утверждение, что частное всегда лежит в иделе;
д) r_in_ideal – утверждение, что если делимое лежит в идеале, то и остаток лежит в идеале.

У типа DivisionResult есть два шаблонных параметра divisible - делимое, и ideal - идеал, образованный делителями.

Листинг 19 – Результат деления
```
structure DivisionResult [MonomialOrder (Variables n ord)]
                         (divisible: Polynomial n ord)
                         (ideal: Ideal (Polynomial n ord)) where
  p: Polynomial n ord
  r: Polynomial n ord
  sum_eq: divisible = p + r
  p_in_ideal: p ∈ ideal
  r_in_ideal: divisible ∈ ideal → r ∈ ideal
```

К сожалению, алгоритм деления слишком сложный, чтобы lean мог вычислить результат тактики rw. Поэтому, свойства делимости записаны в DivisionResult, а не вынесены в отдельные теоремы.

Доказательство каждого свойства в типе DivisionResult проводится в процессе деления. Так, например, свойство sum_eq строится следующим образом: во-первых, перед началом деления переменные p и r инициализированны значениями $divisible$ и 0, для которых, очевидно, данной свойство выполено. Во-вторых, когда процесс деления доходит до шага, на котором становится понятно, что старший член $divisible$ не делится ни на какой старший член делителей, доказывается, что $divisible = p - LT(p) + r + LT(p)$. Алгоритм деления и доказательство остальных свойств можно найти в листинге...

3.5. Алгоритм Бухбергера

3.6. 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. Возращает 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.

3.7. 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