Министерство образования и науки Российской Федерации
Поморский государственный университет им. М. В. Ломоносова
Физический факультет
Кафедра теоретической физики

И. Н. Пашев

Система компьютерной алгебры «Аксиома»

(методические рекомендации)

Архангельск, 2010

Печатается по решению редакционно-издательской комиссии физического факультета Поморского государственного университета им. М. В. Ломоносова.

Автор: *Игорь Николаевич Пашев*, кандидат физ.-мат. наук, старший преподаватель кафедры теоретической физики физического факультета Поморского государственного университета им. М. В. Ломоносова (pashev.igor@gmail.com).

Рецензенты: *Александр Константинович Титов*, кандидат физ.-мат. наук, доцент кафедры информатики, вычислительной техники и методики преподавания информатики физического факультета Поморского государственного университета им. М. В. Ломоносова.

Дмитрий Борисович Сидоров, кандидат физ.-мат. наук, доцент кафедры теоретической физики физического факультета Поморского государственного университета им. М. В. Ломоносова.

Методические рекомендации предназначены студентам и аспирантам физико-математических специальностей в качестве введения в свободную универсальную систему компьютерной алгебры «Аксиома», которая может быть полезна для выполнении рутинных повседневных вычислений, а также для более глубого пониманий сущности математики.

Содержание

Математик, как и художник
и поэт, создаёт узоры. И если его
узоры долговечнее, то это
потому, что они сотканы из идей.

Г. Г. Харди

1 Введение

1.1 Что такое «Аксиома»

«Аксиома» (Axiom) — свободная[1)] универсальная система компьютерной алгебры. Она состоит из среды интерпретатора, компилятора и библиотеки, описывающей строгую, математически правильную иерархию типов [1, 2].

Разработка «Аксиомы» ведётся с 1971 года. В то время она называлась «Черновик» (Scratchpad) и была большим проектом универсальной системы компьютерной алгебры, созданным в «Междельмаше» (IBM) под руководством Ричарда Дженкса (Richard Jenks). На протяжении 20 лет проект «Аксиома», руководимый Барри Тагером (Barry Tager), был инструментом серьёзных исследований в вычислительной математике (см. библиографию [3]). В 1990-е, когда удача отвернулась от IBM, проект был продан компании «Числовые алгоритмы» (Numerical Algorithms Group) и, став коммерческим продуктом, получил своё нынешнее имя. По разным причинам «Аксиома» продавалась не очень хорошо и в октябре 2001 года покинула рынок программного обеспечения. В сентябре 2002 года «Аксиома» была выпущена под свободной лицензией БСД,[2)] а 27-го августа 2003 года стала доступна для загрузки с сайта Фонда свободного программного обеспечения — «Саванна».[3)] В 2007 году на основе «Аксиомы» были созданы ещё два проекта: «Фрикас» и «Открытая Аксиома» (см. далее). В настоящее время «Аксиома» доступна и может быть использована на многих системах типа Юникс, в том числе Линукс, а также Виндоус.[4)]

Одной из причин коммерческого провала «Аксиомы» можно считать (а можно и не считать) саму её суть: она многое делает не так, как большинство программ. В основе «Аксиомы» лежит строгая математика, и это — её

[1)] Пользователь имеет права («свободы») на неограниченные установку, запуск, использование, изучение, распространение и изменение (совершенствование) программы. Эти права защищены юридически авторскими правами при помощи свободных лицензий.
http://ru.wikipedia.org/wiki/Свободное_программное_обеспечение

[2)] http://ru.wikipedia.org/wiki/Лицензия_BSD

[3)] http://savannah.nongnu.org/projects/axiom

[4)] http://axiom-developer.org/axiom-website/faq.html

приоритет над красивой внешностью или удобством использования. Однако удобство использования — понятие субъективное, и для подготовленного специалиста «Аксиома» — настоящий подарок и удовольствие.

«Аксиома» не только интерактивная программа с построчным вводом, она также — компилятор полноценного языка [4], с помощью которого можно строить и исследовать математические конструкции. Фактически, такие построения составляют суть развития «Аксиомы» как программы. Неполный список включает: понятия группы и кольца, арифметику произвольной точности, комплексные и гиперкомплексные числа, матрицы, ряды, пределы, производные, интегралы, дифуры, а также их комбинации (*sic!*), вроде матриц матриц, рядов из матриц и многое другое [1, 5].

Решающая сила «Аксиомы» кроется в её великолепной структуре, которая позволяет наращивать новые возможности, не увеличивая общую сложность системы (программа «не умрёт под собственной тяжестью», чем страдают многие, не только научные, коммерческие монстры). Дизайн «Аксиомы» позволяет интегрировать её с другими инструментами типа численных библиотек на Фортране или Си [6]. «Аксиома» — литературная программа, технология литературного программирования Дональда Кнута [7, 8] используется по всему исходному коду, что позволяет «Аксиоме» иметь актуальную документацию и быть понятной новым разработчикам.

Ориентация на строгую математику позволяет надеяться, что «Аксиома» будет полезна ещё как минимум 40 лет к уже имеющимся 40. В настоящее время не существует конкурента «Аксиоме» на её поле и, что особенно важно, в плане структуры и организации проекта. «Аксиома» предлагает основу для математических изысканий любой сложности и новизны. Она предоставляет язык для компьютерного описания математических объектов и их отношений, что напоминает о попытке Бертрана Рассела [9, 10].

Хотя «Аксиома» уже сейчас представляет собой мощную систему, перспектива использования её для создания новых разделов математики поистине завораживает. В последние годы «Аксиома» была использована для успешного решения задач теоретической математики, математической физики, комбинаторики, обработки сигналов и параллельных вычислений. С её помощью были получены новые диофантовы приближения для числа π; подтверждена гипотеза Гротендика для некоторых классов линейных дифференциальных уравнений; были изучены арифметические свойства униформизации гиперэллиптических кривых; были построены новые алгоритмы факторизации чисел на основе теории групп; с помощью «Аксиомы»

были получены некоторые результаты в квантовой теории поля.[1)] Появление «Аксиомы» в научном мире вывело символьные вычисления на новый уровень, на котором учёные могут формулировать свои мысли и решать задачи с помощью компьютера.[2)]

1.2 Ложка дёгтя

«Аксиома» имеет пару близких родственников — «форков»[3)] (веток): «Фрикас» (FriCAS) и «Открытая Аксиома» (OpenAxiom).[4)] Название последней понятно, а название первой состоит из стилизованного слова *Free* (свободный) и аббревиатуры *CAS — Computer Algebra System* (система компьютерной алгебры). Как обычно такое бывает, параллельные ветки не делаются просто так, особенно у таких серьёзных проектов, а мотивы создание ветки должны быть понятны пользователям.[5)]

Следует признать, что у оригинальной «Аксиомы» за её долгую историю накопилось несколько проблем, связанных с формой, но не с содержанием: не очень удобная процедура установки, основанная на «мэйкфайлах»;[6)] зависимость от статических (а не динамических) библиотек,[7)] которые редко встречаются в современных операционных системах; ну и изначальная ориентация на системы «Юникс» сильно осложняет установку на Window$ (которая появилась гораздо позже «Аксиомы»). «Форки» созданы для решения этих проблем. Кроме этого, «Фрикас» служит тестовой площадкой для проверки новых идей.

Оригинальная «Аксиома», «Фрикас» и «Открытая Аксиома» функционально *пока* практически полностью тождественны и совместимы, документация двух последних отсылает к оригинальной документации «Аксиомы». Разработчики и «Фрикаса», и «Открытой Аксиомы» стремятся сделать процесс установки более легким и управляемым с помощью «ГНУ автотулс» (GNU Autotools).[8)]

Главное их отличие в том, что разработчики «Фрикаса» решили больше не придерживаться литературного подхода в программировании [7], счи-

[1)] Здесь должны были бы быть ссылки на соответствующие публикации, а то так, на словах, ничего не понятно, автор даже не уверен в правильности перевода на русский язык. В любом случае и оригинал [1], и эта брошюра посвящена не этому. Все приведённые страшные незнакомые слова легко ищутся в этих ваших интернетах ☺

[2)] `http://en.wikipedia.org/wiki/Intelligence_amplification`

[3)] От англ. *fork: вилка, ответвляться, разделяться надвое.*

[4)] `http://fricas.sourceforge.net/`, `http://www.open-axiom.org/`

[5)] `http://www.mail-archive.com/axiom-developer@nongnu.org/msg11143.html`

[6)] `http://ru.wikipedia.org/wiki/Makefile`

[7)] Например, `libXpm.a`.

[8)] `http://en.wikipedia.org/wiki/Autotools`

тая, что это поможет оперативному развитию, поиску и исправлению ошибок.[1)] А разработчики «Открытой Аксиомы» напротив, продолжают его использовать и совершенствовать, заявляя, что «Открытая Аксиома» — это правильная, современная «Аксиома».[2)]

В этой брошюре попеременно использовались все три программы, поэтому детали примеров могут отличаться, например, «Фрикас» ставит скобочки при написании типа результата, а «Аскиома» — нет.

1.3 Что ещё посмотреть

Помимо «Аксиомы», конечно, есть ещё свободные программы компьютерной алгебры: Maxima, Mathemagix, Mathomatic, Reduce (да-да, с декабря 2008). Сравнение различных систем см. в [11]. Из численных систем: библиотека GSL (GNU Scientific Library), программы Octave, FreeMat (совместимы с Matlab), программа для обработки статистических данных R. Комбайны SAGE и Scilab. Ещё можно обратить внимание на языки программирования Haskell, Python и Lisp. Haskell наиболее идеологически близок к «Аксиоме». Их объединяет строгая иерархия типов и функциональная парадигма [12]. Всё перечисленное без труда найдётся в интернете (кроме, пожалуй, R ☺), доступно свободно, бесплатно, в любом количестве, для всех, с документацией и исходными текстами.

[1)] `http://www.math.uni.wroc.pl/~hebisch/fricas/fricas-reg.html`, `http://axiom-wiki.newsynthesis.org/FriCAS`

[2)] «Axiom of Choice» (*англ.*) — игра слов, намекающая на аксиому выбора в теории множеств, `http://axiom-wiki.newsynthesis.org/OpenAxiom`.

2 Начало работы

2.1 Ввод и вывод

Интерактивная среда «Аксиомы» запускается в терминале командой `axiom`.[1)] Чтобы выйти из «Аксиомы», надо набрать команду `)quit` (со скобочкой) и нажать клавишу `Enter`. Всё это написано в приветствии, выводимом программой при запуске (пример 1). При работе в графической среде «линукса» появляется довольно страшненькое окно интерактивной справки. Не стоит сильно пугаться — в планах разработчиков есть переделка этой системы. Однако она действительно очень полезна, позволяя не только читать, но и пробовать приведённые в ней примеры.

Пример 1. Приветствие «Фрикаса»

```
                  FriCAS (AXIOM fork) Computer Algebra System
                           Version: FriCAS 2010-03-23
                  Timestamp: Thursday May 13, 2010 at 20:21:16
------------------------------------------------------------------------------
   Issue )copyright to view copyright notices.
   Issue )summary for a summary of useful system commands.
   Issue )quit to leave FriCAS and return to shell.
------------------------------------------------------------------------------

(1) ->
```

В интерактивном режиме команды вводятся в одну строку, нажатие клавиши `Enter` приводит к выполнению команды и выводу результата. Несколько команд можно ввести в одной строке, разделяя их символом «точка с запятой» (`;`), при этом выводится только результат последней. Длинную строку можно разбить на несколько, заканчивая каждую символом подчёркивания (`_`) и нажимая `Enter`.

Для ссылки на предыдущие результаты вычислений используются символы: `%` — самый последний; `%% n` — результат номер `n`, который лучше заключать в скобки; `%%(-n)` — результат номер `n` с конца, так что `%%(-1)`≡`%` (пример 2). Конечно, удобнее вводить свои обозначения (см. с. 15).

Пример 2. Ссылки на результаты

```
(1) -> 1+2

   (1)  3
                                                        Type: PositiveInteger
(2) -> sin(%pi/2)

   (2)  1
                                                     Type: Expression Integer

```

[1)] `fricas` или `open-axiom`.

```
(3) -> log(x+y)

   (3)  log(y + x)
                                                    Type: Expression Integer
(4) -> tan(%i::Complex Float)

   (4)  0.7615941559 5576488812 %i
                                                         Type: Complex Float
(5) -> %

   (5)  0.7615941559 5576488812 %i
                                                         Type: Complex Float
(6) -> %% 3

   (6)  log(y + x)
                                                    Type: Expression Integer
(7) -> %%(-1)

   (7)  log(y + x)
                                                    Type: Expression Integer
(8) -> %%(-3)

   (8)  0.7615941559 5576488812 %i
                                                         Type: Complex Float
```

«Аксиома» по умолчанию выводит результаты в виде, похожем на «человеческую» запись. Можно попросить её заодно выводить результаты в формате TEX, в формате языка Фортран, MathML и др. (пример 3).

Пример 3. Параметры вывода «Открытой Аксиомы»

```
(5) -> )set output
                  Current Values of  output  Variables

Variable     Description                                  Current Value
------------------------------------------------------------------------------
abbreviate   abbreviate type names                        off
algebra      display output in algebraic form             On:CONSOLE
characters   choose special output character set          plain
fortran      create output in FORTRAN format              Off:CONSOLE
fraction     how fractions are formatted                  vertical
length       line length of output displays               77
openmath     create output in OpenMath style              Off:CONSOLE
script       display output in SCRIPT formula format      Off:CONSOLE
scripts      show subscripts,... linearly                 off
showeditor   view output of )show in editor               off
tex          create output in TeX style                   Off:CONSOLE
mathml       create output in MathML style                Off:CONSOLE

(6) -> )set output tex on
(6) -> )set output mathml on
(6) -> integrate(sin(x)*x^2, x)

                          2
   (6)  2x sin(x) + (- x  + 2)cos(x)
$$
{2 \  x \  {\sin
\left(
```

```
{x}
\right)}}+{{\left(
-{x \sp 2}+2
\right)}
\  {\cos
\left(
{x}
\right)}}
\leqno(6)
$$

<math xmlns="http://www.w3.org/1998/Math/MathML" mathsize="big" display="block">
<mrow><mrow><mn>2</mn><mo>&#x02062;</mo><mi>x</mi><mo>&#x02062;</mo>
<mrow><mo><mo>sin</mo></mo><mo>(</mo><mrow><mi>x</mi></mrow><mo>)</mo>
</mrow></mrow><mo>+</mo><mrow><mrow><mo>(</mo><mo>-</mo>
<mrow><msup><mrow><mi>x</mi></mrow><mrow><mn>2</mn></mrow>
</msup></mrow><mo>+</mo><mn>2</mn><mo>)</mo></mrow><mo>&#x02062;</mo>
<mrow><mo><mo>cos</mo></mo><mo>(</mo><mrow><mi>x</mi></mrow><mo>)</mo>
</mrow></mrow></mrow>
</math>
                                                    Type: Union(Expression Integer,...)
```

Можно запустить интерактивную сессию в программе TeXmacs.[1)] Проще всего это сделать с оригинальной «Аксиомой», и хотя TeXmacs — достаточно тяжёлая, требовательная к ресурсам программа, результат прекрасен (см. рис. на этой странице).

```
→ series(log(1+x), x=0)
```

$$x - \frac{1}{2}x^2 + \frac{1}{3}x^3 - \frac{1}{4}x^4 + \frac{1}{5}x^5 - \frac{1}{6}x^6 + \frac{1}{7}x^7 - \frac{1}{8}x^8 + \frac{1}{9}x^9 - \frac{1}{10}x^{10} + \frac{1}{11}x^{11} + O\left(x^{12}\right) \quad (4)$$

```
                                  Type: UnivariatePuiseuxSeries(Expression Integer,x,0)
→ integrate(sqrt(1+cos(2*x)), x)
```

$$\frac{\sin(2x)\sqrt{\cos(2x)+1}}{\cos(2x)+1} \quad (5)$$

```
                                                    Type: Union(Expression Integer,...)
→ limit((cos x)^(1/x^2), x=0)
```

$$\frac{1}{e^{\frac{1}{2}}} \quad (6)$$

```
                                  Type: Union(OrderedCompletion Expression Integer,...)
→ y:=operator 'y
```

$$y \quad (7)$$

```
                                                                  Type: BasicOperator
→ solve (D(y x, x, 2) - y(x) = 2*x*sin(x), y, x)
```

$$\left[particular = -x\sin\left(x\right) - \cos\left(x\right),\ \ basis = \left[e^{x},\ \ e^{(-x)}\right]\right] \quad (8)$$

```
Type: Union(Record(particular: Expression Integer,basis: List Expression
Integer),...)
→
```

Сеанс работы с «Аксиомой» в программе TeXmacs.

[1)] «Техмакс» — http://www.texmacs.org/

2.2 Загрузка файлов

В основе «Аксиомы» — полноценный язык программирования под названием «Спад» (Spad), на котором написана вся математическая база «Аксиомы».[1)] Программы на этом языке можно сохранять в отдельных файлах, которые затем можно либо компилировать, либо подгружать в интерактивную сессию. В повседневной работе скорее всего потребуется последнее, а для расширения возможностей «Аксиомы» — первое. Хотя язык в обоих случаях одинаков, требования к оформлению кода несколько отличаются. Примеры можно посмотреть в самой «Аксиоме» — исходные тексты открыты и доступны всем.

Инструкции языка Спад группируются не скобочками, как, например, в языках программирования в Си (C) или Перле (Perl), а отступами, как в языках Питон (Python) и Хаскель (Haskell).

Для загрузки файлов в интерактивную сессию используется команда `)read "имя файла"`. Загружаемые файлы должны иметь расширение `.input`, но при загрузке само расширение указывать не обязательно. Путь файла должен быть указан полностью или относительно текущего каталога. Сменить текущий каталог можно командой `)cd "каталог"`.[2)]

В примере 4 приведено содержимое файла `fib.input`, в котором определены две функции, вычисляющие элементы последовательности Фибоначчи: рекурсивно и по формуле общего члена.[3)] Сеанс работы с этим файлом показан в примере 5 на следующей странице. При загрузке файла в интерактивную сессию «Аксиома» выводит всё его содержимое и результаты вычислений в нём так, как было бы, если вводить все команды вручную.

Пример 4. Файл «fib.input»

```
1 p(n) ==
2     n = 1 => 1
3     n = 2 => 1
4     p(n-1) + p(n-2)
5
6 q(n) == ((1 + sqrt 5)^n - (1 - sqrt 5)^n) / (2^n * sqrt(5))
```

[1)] А сам он написан на Лиспе [4].

[2)] Возможно, надо указывать косую черту в конце имени каталога.

[3)] «Аксиому» совершенно не смущают большие степени или иррациональные числа — она работает с символами, и хотя в формуле есть корни, конечный ответ — натуральный. Однако попробуйте `q(1000)` или больше — и узнайте *истину* ;-) Приведённая рекурсивная формула не самая эффективная.

Пример 5. Работа с файлом «fib.input»

```
(1) -> )read "fib"

p(n) ==
    n = 1 => 1
    n = 2 => 1
    p(n-1) + p(n-2)
                                                                    Type: Void
q(n) == ((1 + sqrt 5)^n - (1 - sqrt 5)^n) / (2^n * sqrt(5))
                                                                    Type: Void
(3) -> p(5)
   Compiling function p with type Integer -> PositiveInteger
   Compiling function p as a recurrence relation.

   (3)  5
                                                        Type: PositiveInteger
(4) -> q(5)
   Compiling function q with type PositiveInteger -> AlgebraicNumber

   (4)  5
                                                        Type: AlgebraicNumber
(5) -> p(50)

   (5)  12586269025
                                                        Type: PositiveInteger
(6) -> q(50)

   (6)  12586269025
                                                        Type: AlgebraicNumber
```

3 Простые примеры

Здесь в качестве иллюстраций, без подробных пояснений, приведены простые вещи, которые может делать «Аксиома» и которые показались автору интересными. Некоторые важные понятия изложены подробнее. Разумеется, программа способна на большее [1, 5]. За незнакомыми словами следует обращаться к Фихтенгольцу [13] или Смирнову [14],[1)] а за задачами — к Демидовичу [15, 16].

3.1 Арифметика

«Аксиома» способна работать с вещественными числами очень большой точности (пример 6). Однако следует понимать, что такие числа при любой точности всё же приближённы. Если нужен числовой результат, следует до самого конца проводить символьные вычисления и лишь затем превращать результат в вещественное число. В конце концов, «Аксиома» — это система компьютерной алгебры, а не «числодробильная» программа! Точность задаётся функцией `digits` (изначально — 20). Для представление результата в числовой (десятичной) форме используется функция `numeric`, для перевода в любую другую систему исчисления — `radix`. Возведение в степень обозначается символом `**` либо `^`.[2)]

Пример 6. Арифметика

```
(1) -> digits 1000; numeric(%pi)

   (1)
  3.1415926535 8979323846 2643383279 5028841971 6939937510 5820974944 592307816
  4 0628620899 8628034825 3421170679 8214808651 3282306647 0938446095 505822317
  2 5359408128 4811174502 8410270193 8521105559 6446229489 5493038196 442881097
  5 6659334461 2847564823 3786783165 2712019091 4564856692 3460348610 454326648
  2 1339360726 0249141273 7245870066 0631558817 4881520920 9628292540 917153643
  6 7892590360 0113305305 4882046652 1384146951 9415116094 3305727036 575959195
  3 0921861173 8193261179 3105118548 0744623799 6274956735 1885752724 891227938
  1 8301194912 9833673362 4406566430 8602139494 6395224737 1907021798 609437027
  7 0539217176 2931767523 8467481846 7669405132 0005681271 4526356082 778577134
  2 7577896091 7363717872 1468440901 2249534301 4654958537 1050792279 689258923
  5 4201995611 2129021960 8640344181 5981362977 4771309960 5187072113 499999983
  7 2978049951 0597317328 1609631859 5024459455 3469083026 4252230825 334468503
  5 2619311881 7101000313 7838752886 5875332083 8142061717 7669147303 598253490
  4 2875546873 1159562863 8823537875 9375195778 1857780532 1712268066 130019278
  7 6611195909 216420199
                                                                  Type: Float

```

[1)] А также к Яндексу, Гуглу и Википедии.

[2)] Полноценная поддержка символа `^` появилась недавно (`http://www.open-axiom.org/1.3/`).

```
(2) -> digits 100; numeric(%e)

   (2)
  2.7182818284 5904523536 0287471352 6624977572 4709369995 9574966967 627724076
  6 3035354759 4571382178 525166427
                                                                  Type: Float
(3) -> digits(10)

   (3)  10
                                                        Type: PositiveInteger
(4) -> sqrt(2)

         +-+
   (4)  \|2
                                                        Type: AlgebraicNumber
(5) -> numeric %

   (5)  1.414213562
                                                                  Type: Float
(6) -> 8*10 + 2 + 9 + 9

   (6)  100
                                                        Type: PositiveInteger
(7) -> (14*6 + 1)*10 + 15*9 + 15

   (7)  1000
                                                        Type: PositiveInteger
(8) -> radix(%, 16)

   (8)  3E8
                                                    Type: RadixExpansion 16
(9) -> numeric %e^%pi

   (9)  23.14069263
                                                                  Type: Float
(10) -> 1/2 + 3/7 - 7/11

          45
   (10)  ---
         154
                                                     Type: Fraction Integer
```

В «Аксиоме» легко реализуется модульная арифметика (пример 7). Например, расчёт времени дня (часы) использует арифметику по модулю 12 или 24. Так что $22+3=1$. Арифметика по модулю 360 используется для отсчёта углов.

Пример 7. Арифметика по модулю 2

```
(1) -> x:IntegerMod 2 := 1

   (1)  1
                                                      Type: IntegerMod 2
(2) -> y:IntegerMod 2
                                                                 Type: Void

```

```
(3) -> x+2

   (3)  1
                                                        Type: IntegerMod 2
(4) -> y := 1

   (4)  1
                                                        Type: IntegerMod 2
(5) -> x+y

   (5)  0
                                                        Type: IntegerMod 2
```

3.2 Переменные, макросы и функции

3.2.1 Переменные и макросы

Громоздкие выражения в математике принято обозначать отдельными новыми переменными, это позволяет также видеть более крупномасштабную структуру выражений. Например, дискриминант квадратного уравнений $\mathcal{D} = \sqrt{b^2 - 4ac}$. То же самое можно делать и в «Аксиоме» с помощью символа «`:=`» вот так — `d:=sqrt(b**2-4*a*c)`.[1)] Теперь везде, где появится переменная `d`, вместо неё будет подставлено выражение `sqrt(b**2-4*a*c)`. Ещё пример — `int:=integrate` — и можно писать интегралы короче.

Использование переменных имеет и «обратную силу», например, можно менять элементы матрицы, обращаясь к ней по имени переменной. Более общий случай подстановок реализуется с помощью макросов — подстановок с параметрами, но они уже обратной силы не имеют, так как означают *буквальную* подстановку (пример 8).

Пример 8. Переменные и макросы

```
(1) -> d := sqrt(b**2 - 4*a*c)

          +-----------+
          |           2
   (1)  \|- 4a c + b
                                                   Type: Expression Integer
(2) -> differentiate(d, b)

                 b
   (2)  --------------
          +-----------+
          |           2
         \|- 4a c + b
                                                   Type: Expression Integer
(3) -> macro d (a,b) == sqrt(b**2 - 4*a*c)
                                                                Type: Void
```

[1)] Заглавную букву `D` использовать нежелательно, так как по умолчанию за ней закреплено сокращение для функции `differentiate`.

```
(4) -> d(1,2)

          +--------+
   (4)  \|- 4c + 4
                                                    Type: AlgebraicNumber
(5) -> d(x,y)

          +----------+
          |  2
   (5)  \| y  - 4c x
                                                  Type: Expression Integer

(6) -> d(d(p,q),y)

          +----------------------+
          |     +---------+
          |     | 2             2
   (6)  \|- 4c\|q  - 4c p  + y
                                                  Type: Expression Integer
(7) -> m := matrix [[1,2],[3,4]]

         +1  2+
   (7)   |    |
         +3  4+
                                                     Type: Matrix Integer
(8) -> m(1,2) := 6

   (8)  6
                                                    Type: PositiveInteger
(9) -> m

         +1  6+
   (9)   |    |
         +3  4+
                                                     Type: Matrix Integer
```

3.2.2 Функции

Функции принципиально отличаются от макросов и являются одним из основных понятий «Аксиомы» и т. н. функциональных языков программирования.[1)] Функция — это последовательность действий (операций). Функция задаётся конструкцией:

`имя (аргументы) == тело функции`.

Результат функции — результат последней операции в её теле.

Функции могут иметь одинаковые имена, а смысл функции (последовательность операций) определяется типом и количеством переданных ей параметров. Например, и для решения алгебраических уравнений, и для решения дифференциальных уравнений, и для решения их систем используется функция `solve` (см. далее).

[1)] http://ru.wikipedia.org/wiki/Функциональное_программирование

Разные типы выражений могут иметь одинаковое представление. Например, `I` может быть переменной или римским число, а `1` может быть целым или вещественным, или даже единичной матрицей. Более сложный пример представляют матрицы: не всякие матрицы образуют кольцо, но квадратные матрицы — образуют кольцо. Есть функции, которые определены только для объектов, образующих кольцо, и если требуется применить их к квадратным матрицам, необходимо явно указать, что матрицы квадратны ☺.

Чтобы узнать, для каких типов объектов применима та или иная функция, служит команда `)display operations <имя функции>`.[1)] Чтобы узнать, какие функции применимы к тому или иному типу объектов, служит команда `)show <тип>`. Эта команда также показыват сокращённые названия типов, например, `INT` для `Integer`, `POLY` для `Polynomial`, так что тип `Polynomial Integer` можно кратко обозначать `POLY INT`.

Тип объекта можно уточнить либо при его объявлении (через одно двоеточие), либо при использовании (через два двоеточия). Если тип не указан, подразумевается самый простой, очевидный вариант (пример 9).

Пример 9. Типы выражений

```
(1) -> 1

   (1)  1
                                                    Type: PositiveInteger
(2) -> x+1

   (2)  x + 1
                                                 Type: Polynomial Integer
(3) -> 1::Integer

   (3)  1
                                                         Type: Integer
(4) -> 1::Float

   (4)  1.0
                                                           Type: Float
(5) -> i

   (5)  i
                                                       Type: Variable I
(6) -> i::ROMAN

   (6)  I
                                                    Type: RomanNumeral
(7) -> sin(1)

   (7)  sin(1)
                                                 Type: Expression Integer

```

[1)] Или короче, `)di op <имя функции>`.

```
(8) -> sin(1::Float)

   (8)  0.8414709848 0789650665
                                                                Type: Float
(9) -> 1::SquareMatrix(3, Integer)

         +1  0  0+
         |       |
   (9)   |0  1  0|
         |       |
         +0  0  1+
                                                 Type: SquareMatrix(3,Integer)

(10) -> A : SquareMatrix(2, SquareMatrix(2, Integer))
                                                                 Type: Void
(11) -> A := matrix [[1,2],[3,4]]

          ++1  0+  +2  0++
          ||    |  |    ||
          |+0  1+  +0  2+|
   (11)   |               |
          |+3  0+  +4  0+|
          ||    |  |    ||
          ++0  3+  +0  4++
                                 Type: SquareMatrix(2,SquareMatrix(2,Integer))
```

Для функций одного аргумента необязательно заключать его в скобки, если это не приводит к неоднозначности (пример 10).

Пример 10. Функции одного аргумента без скобочек

```
(1) -> f: Integer -> Integer
                                                                 Type: Void
(2) -> f x == x^2
                                                                 Type: Void
(3) -> f 2

   (3)  4
                                                    Type: PositiveInteger
(4) -> f 2 +f 4

   (4)  20
                                                    Type: PositiveInteger
(5) -> sin sin sin 1

   (5)  sin(sin(sin(1)))
                                                 Type: Expression Integer
(6) -> numeric(%)

   (6)  0.6784304773 6074022898
                                                                 Type: Float
(7) -> sin sin sin 1+1

   (7)  sin(sin(sin(1))) + 1
                                                 Type: Expression Integer
(8) -> f f 2

   (8)  16
                                                    Type: PositiveInteger
```

Характерной особенностью использования функций являются «ленивость вычислений», суть которой в том, что действия (операции) не производятся до тех пор, пока не понадобится их результат. Это позволяет, например, оперировать бесконечными последовательностями, рядами, делить на нуль и т. п. Функциональное программирование — это отдельная очень интересная тема, для которой здесь, однако, не найдётся места (см. [12] и приведённые там ссылки). Акцент не на объектах, а на операциях с ними привёл в математике к понятиям «алгебра», «поле», «кольцо», «группа» и др.,[1)] которые также представлены в иерархии типов «Аксиомы» [1].

Использование функций уже было показано ранее в примерах 4 и 5 на с. 11. Здесь же посмотрим на полиномы Лежандра. Эти полиномы могут быть определены либо рекурсивно:

$$p_n(x) = \frac{1}{n}\left[(2n-1)\,x \cdot p_n(x) - (n-1) \cdot p_{n-2}(x)\right];$$

либо по формуле Родриго:

$$p_n(x) = \frac{1}{2^n n!} \cdot \frac{\mathrm{d}^n}{\mathrm{d}x^n}(x^2-1)^n.$$

Функция `p(n)` принимает в качестве аргумента символ `n`, а как результат возвращает выражение. При этом она выполняет некоторые действия (операции) над символом `n`, и если для какого-то типа `n` (например, матрицы) эти операции не определены, «Аксиома» сообщит об ошибке. Кроме этого, если был указан тип функции, попытка использовать её не по назначению будет отвергнута. Чтобы превратить какое-либо выражение (например, результат длинных вычислений) в новую функцию, надо использовать функцию `function` (пример 11).

Пример 11. Полиномы Лежандра

```
(1) -> p : (Integer) -> Polynomial Fraction Integer
                                                                  Type: Void
(2) -> p(0) == 1
                                                                  Type: Void
(3) -> p(1) == x
                                                                  Type: Void
(4) -> p(n) == ((2*n-1)*x*p(n-1) - (n-1) * p(n-2))/n
                                                                  Type: Void
(5) ->  p(5)
   Compiling function p with type Integer -> Polynomial Fraction
      Integer
   Compiling function p as a recurrence relation.
        63  5   35  3   15
   (5)  -- x  - -- x  + -- x
         8       4       8
```

[1)] Даже арифметика оперирует не огурцами и помидорами, а *числами*.

```
                                                    Type: Polynomial Fraction Integer
(6) -> coefficient(p(5), x, 5)

         63
   (6)   --
         8
                                                    Type: Polynomial Fraction Integer
(7) -> p(-1)

   You did not define p for argument -1 .

(7) -> p(a)
   Conversion failed in the compiled user function p .

   Cannot convert from type Symbol to Integer for value
   a

(8) -> function(%% 5, 'p5, 'x)

   (8)  p5
                                                                    Type: Symbol
(9) -> p5

                 63  5   35  3   15
   (9)  p5 x == -- x  - -- x  + -- x
                 8       4       8
                                                        Type: FunctionCalled p5
(10) -> p5(1/2)
   Compiling function p5 with type Fraction Integer -> Fraction Integer


          23
   (10)  ---
         256
                                                         Type: Fraction Integer
```

Наконец, ещё несколько способов определения функций, смысл которых должен быть понятен и весьма привлекателен (пример 12). Там, где удобно, можно использовать функции без имён — анонимные.

Пример 12. Способы задания функций

```
(1) -> p(x, y | x=y) == 0
                                                                    Type: Void
(2) -> p(x, y | x<y) == -1
                                                                    Type: Void
(3) -> p(x, y | x>y) == 1
                                                                    Type: Void
(4) -> p(1,2)
   Compiling function p with type (PositiveInteger,PositiveInteger) ->
      Integer

   (4)  - 1
                                                                 Type: Integer

```

```
(5) -> p(2,0)
   Compiling function p with type (PositiveInteger,NonNegativeInteger)
       -> Integer

   (5)  1
                                                          Type: PositiveInteger
(6) -> p(a,a)
   Compiling function p with type (Variable a,Variable a) ->
      NonNegativeInteger

   (6)  0
                                                               Type: Integer
(7) -> e(1,2,3) == 1
                                                                  Type: Void
(8) -> e(i,i,l) == 0
                                                                  Type: Void
(9) -> e(i,k,k) == 0
                                                                  Type: Void
(10) -> e(l,k,l) == 0
                                                                  Type: Void
(11) -> e(i,k,l | k < i) == - e(k,i,l)
                                                                  Type: Void
(12) -> e(i,k,l | l < k) == - e(i,l,k)
                                                                  Type: Void
(13) -> e(3,2,1)
   Compiling function e with type (Integer,Integer,Integer) -> Integer

   (13)  - 1
                                                               Type: Integer
(14) -> e(3,2,2)

   (14)  0
                                                        Type: NonNegativeInteger
(15) -> e(1,3,2)

   (15)  - 1
                                                               Type: Integer
(16) -> (x +-> x^2) 5

   (16)  25
                                                          Type: PositiveInteger
(17) -> map ((x +-> x^2), [1,2,3,4,5])

   (17)  [1,4,9,16,25]
                                                      Type: List PositiveInteger
```

3.2.3 Операторы

Операции (действия) с математическими объектами также являются математическими объектами. Абстракные операции (которые «что-то» делают) создаются функцией `operator` с аргументом, означающим имя оператора, которое будет использоваться для обозначения этого оператора (пример 13 на следующей странице). Наиболее частое использование операторов — дифференциальные уравнения (пример 30 на с. 45). Имя оператора

никак не связано с именами переменных, и чтобы не водить «Аксиому» в заблуждение, часто имя оператора «экранируют» с помощью символа ’ (апостро́ф); можно просто заключать его в кавычки, если так привычнее.

Пример 13. Абстрактные операторы

```
(1) -> (operator w) x

   (1)  w(x)
                                                    Type: Expression(Integer)
(2) -> t := operator r

   (2)  r
                                                          Type: BasicOperator
(3) -> t x

   (3)  r(x)
                                                    Type: Expression(Integer)
(4) -> differentiate (t x, x)

          ,
   (4)  r (x)

                                                    Type: Expression(Integer)
```

3.3 Производные

Производные выводятся с помощью функции `differentiate`. Для краткости можно использовать лишь букву `D` (заглавную).

Пример 14. Производные

```
(1) -> differentiate(x^2,x)

   (1)  2x
                                                    Type: Polynomial(Integer)
(2) -> differentiate(t^2*sin(t),t)

                       2
   (2)  2t sin(t) + t cos(t)
                                                    Type: Expression(Integer)
(3) -> D(2*r/(1-r^2),r)

              2
            2r  + 2
   (3)  ------------
          4     2
         r  - 2r  + 1
                                          Type: Fraction(Polynomial(Integer))
(4) -> f == atan(x/y)
                                                                 Type: Void
```

```
(5) -> differentiate(f,x)
   Compiling body of rule f to compute value of type Expression(Integer)

            y
   (5)  -------
          2    2
         y  + x
                                                    Type: Expression(Integer)
(6) -> differentiate(f,x,2)

                 2x y
   (6)  - ---------------
            4     2 2    4
           y  + 2x y  + x
                                                    Type: Expression(Integer)
(7) -> differentiate(f,[x,y])

               2    2
            - y  + x
   (7)  ---------------
          4     2 2    4
         y  + 2x y  + x
                                                    Type: Expression(Integer)

(8) -> differentiate(f,[y,x])

               2    2
            - y  + x
   (8)  ---------------
          4     2 2    4
         y  + 2x y  + x
                                                    Type: Expression(Integer)
(9) -> differentiate(f,[y,y,x])

                  3     2
                2y  - 6x y
   (9)  -----------------------
          6     2 4     4 2    6
         y  + 3x y  + 3x y  + x
                                                    Type: Expression(Integer)
```

3.4 Последовательности

Последовательности, в том числе бесконечные, создаются конструкцией `[f(n) for n in a]`. Для применения некоторой функции ко всем членам последовательности используется функция `map`. В примере 15 на следующей странице показано табулирование функции на заданном отрезке с заданным шагом. Для свёртки конечной последовательности с помощью некоторой операции (сложение, умножение, разность, минимум, максимум) используется функция `reduce`, суммы задаются функцией `sum` (пример 16 на

следующей странице). Есть ряд замечательных функций [2],[1)] которые пытаются определить формулу общего члена последовательности по её нескольким первым членам (пример 17 на с. 26, см. также [17]).

Пример 15. Табулирование функции

```
(1) -> f(x | 0 < x and x < %pi::Float) == (sin(6*x))^8
                                                                  Type: Void
(2) -> f(x | x > %pi::Float) == x^0.3 - log x
                                                                  Type: Void
(3) -> a := 1

   (3)  1
                                                       Type: PositiveInteger
(4) -> h := 28/10

         14
   (4)   --
          5
                                                     Type: Fraction Integer
(5) -> N := 28

   (5)  28
                                                       Type: PositiveInteger

(6) -> xs := [a + (i-1)*h for i in 1..N]

   (6)
       19  33  47  61      89  103  117  131     159  173  187  201     229
   [1, --, --, --, --, 15, --, ---, ---, ---,29, ---, ---, ---, ---,43, ---,
        5   5   5   5       5   5    5    5       5    5    5    5       5
    243  257  271      299  313  327  341      369  383
    ---, ---, ---, 57, ---, ---, ---, ---, 71, ---, ---]
     5    5    5        5    5    5    5        5    5
                                              Type: List Fraction Integer
(7) -> map (f, xs)

   (7)
   [0.0000371538 5911246236 3512, 0.1575702074 190596162,
    - 0.1256486048 963551655, - 0.2821430351 790296267,
    - 0.3835237657 092241016, - 0.4547068202 595548022,
    - 0.5071366515 329071371, - 0.5469551860 733046179,
    - 0.5778099943 731407276, - 0.6020282732 482968866,
    - 0.6211765366 239952008, - 0.6363541081 303804122,
    - 0.6483586132 830399735, - 0.6577848046 87470809,
    - 0.6650863883 193982392, - 0.6706162241 968596487,
    - 0.6746533298 073460229, - 0.6774215302 418387027,
    - 0.6791026527 483784324, - 0.6798460599 767522388,
    - 0.6797756667 896054753, - 0.6789951906 809840526,
    - 0.6775921387 967802875, - 0.6756408759 873226829,
    - 0.6732050142 048152134, - 0.6703392937 76356076,
    - 0.6670910794 384373425, - 0.6635015609 353793276]
                                                 Type: List Expression Float

```

[1)] С удивлением и некоторым разочарованием обнаружилось, что в момент подготовки этого текста они были доступны только во «Фрикасе».

```
(8) -> map (x +-> (x, f x), xs)

   (8)
   [(1.0,0.0000371538 5911246236 3512), (3.8,0.1575702074 190596162),
    (6.6,- 0.1256486048 963551655), (9.4,- 0.2821430351 790296267),
    (12.2,- 0.3835237657 092241016), (15.0,- 0.4547068202 595548022),
    (17.8,- 0.5071366515 329071371), (20.6,- 0.5469551860 733046179),
    (23.4,- 0.5778099943 731407276), (26.2,- 0.6020282732 482968866),
    (29.0,- 0.6211765366 239952008), (31.8,- 0.6363541081 303804122),
    (34.6,- 0.6483586132 830399735), (37.4,- 0.6577848046 87470809),
    (40.2,- 0.6650863883 193982392), (43.0,- 0.6706162241 968596487),
    (45.8,- 0.6746533298 073460229), (48.6,- 0.6774215302 418387027),
    (51.4,- 0.6791026527 483784324), (54.2,- 0.6798460599 767522388),
    (57.0,- 0.6797756667 896054753), (59.8,- 0.6789951906 809840526),
    (62.6,- 0.6775921387 967802875), (65.4,- 0.6756408759 873226829),
    (68.2,- 0.6732050142 048152134), (71.0,- 0.6703392937 76356076),
    (73.8,- 0.6670910794 384373425), (76.6,- 0.6635015609 353793276)]
                                          Type: List Tuple Expression Float
```

Пример 16. Последовательности

```
(1) -> [i for i in 1..]

   (1)  [1,2,3,4,5,6,7,8,9,10,...]
                                                  Type: Stream PositiveInteger
(2) -> f(x) == x^2
                                                               Type: Void
(3) -> map (f, %% 1)

   (3)  [1,4,9,16,25,36,49,64,81,100,...]
                                                  Type: Stream PositiveInteger
(4) -> (%% 3)(30)

   (4)  900
                                                    Type: PositiveInteger
(5) -> ns := [(n-7)*(n+3) for n in 1..10]

   (5)  [- 24,- 25,- 24,- 21,- 16,- 9,0,11,24,39]
                                                       Type: List Integer
(6) -> reduce(min, ns)

   (6)  - 25
                                                            Type: Integer
(7) -> reduce(max, ns)

   (7)  39
                                                    Type: PositiveInteger
(8) -> reduce(+, ns)

   (8)  - 45
                                                            Type: Integer
(9) -> reduce(*, ns)

   (9)  0
                                                 Type: NonNegativeInteger

```

```
(10) -> sum(x^k, k=1..n)
                n
            x x  - x
   (10)  --------
             x - 1
                                                    Type: Expression Integer
(11) -> sum(k^3, k=1..n)

             4     3    2
            n  + 2n  + n
   (11)  -------------
                 4
                                                  Type: Fraction Polynomial Integer
(12) -> sum(k, k=1..n)^2

             4     3    2
            n  + 2n  + n
   (12)  -------------
                 4
                                                  Type: Fraction Polynomial Integer
(13) -> sum (sin(k*x), k=1..n)

             n
            --+
   (13)  >      sin(k x)
            --+
            k= 1
                                                    Type: Expression Integer

(14) -> D(%,x)

             n
            --+
   (14)  >      k cos(k x)
            --+
            k= 1
                                                    Type: Expression Integer
```

Пример 17. Формула общего члена

```
(1) -> guessPRec [1,2,3,4,5]

   (1)  [n + 1]
                                                  Type: List(Expression(Integer))
(2) -> guessPRec [1,3,5,7,9]

   (2)  [2n + 1]
                                                  Type: List(Expression(Integer))
(3) -> guessPRec [1,3,6,10,15,21,28]

             2
            n  + 3n + 2
   (3)  [-----------]
                2
                                                  Type: List(Expression(Integer))
(4) -> guessPRec [1,2,3,5,8,13]

   (4)  [[f(n): - f(n + 2) + f(n + 1) + f(n)= 0,f(0)= 1,f(1)= 2]]
                                                  Type: List(Expression(Integer))
```

```
(5) -> guessPRec [1/2,2/3,3/4,4/5,5/6]

           n + 1
   (5)  [-----]
           n + 2
                                                  Type: List(Expression(Integer))
```

3.5 Матрицы

Матрицы создаются функцией `matrix`, диагональные матрицы можно создать проще — функцией `diagonalMatrix`. Ранг матрицы вычисляется функцией `rank`, определитель — `determinant`, обратная матрица — `inverse` (или возвести в степень -1), транспонированная — `transpose`, след матрицы — `trace`. Доступ к элементам матрицы — `матрица(строка, столбец)`. Разумеется, матрицы можно складывать, умножать друг на друга и на скаляр, возводить в целую степень. Все возможные операции над матрицами покажет команда `)show Matrix`. Следует помнить, что «Аксиома» работает по вполне определённым правилам и лишена творческого начала, поэтому функция `rank` не замечает, что $\sin x / \operatorname{tg} x = \cos x$ (пример 18).

Пример 18. Матрицы

```
(1) -> M := matrix [[1,2], [3,4]]

          +1  2+
   (1)    |    |
          +3  4+
                                                  Type: Matrix(Integer)
(2) -> determinant M

   (2)  - 2
                                                     Type: Integer
(3) -> rank M

   (3)  2
                                                  Type: PositiveInteger
(4) -> M' := inverse M

          +- 2   1 +
          |         |
   (4)    | 3     1|
          | -   - -|
          + 2     2+
                                  Type: Union(Matrix(Fraction(Integer)),...)
(5) -> M' * M

          +1  0+
   (5)    |    |
          +0  1+
                                                  Type: Matrix(Fraction(Integer))
```

```
(6) -> N := matrix [[1,2,3],[4,5,6]]

          +1  2  3+
     (6)  |       |
          +4  5  6+
                                                    Type: Matrix(Integer)
(7) -> N*M

   >> Error detected within library code:
   can't multiply matrices of incompatible dimensions

(7) -> M*N

          +9   12  15+
     (7)  |          |
          +19  26  33+
                                                    Type: Matrix(Integer)
(8) -> F := matrix [[x, x^2], [sin(x), 1/x]]

          +          2+
          |  x      x |
     (8)  |           |
          |         1 |
          |sin(x)   - |
          +         x +
                                          Type: Matrix(Expression(Integer))
(9) -> determinant F
              2
     (9)  - x sin(x) + 1
                                                  Type: Expression(Integer)
(10) -> D(%, x)

                          2
     (10)  - 2x sin(x) - x cos(x)
                                                  Type: Expression(Integer)
(11) -> diagonalMatrix [1,2,3]

           +1  0  0+
           |       |
     (11)  |0  2  0|
           |       |
           +0  0  3+
                                                    Type: Matrix(Integer)
(12) -> rank matrix [[x, cos(x)], [2*x, 2*cos(x)]]

     (12)  1
                                                 Type: PositiveInteger
(13) -> rank matrix [[x, cos(x)], [x, sin(x)/tan(x)]]

     (13)  2
                                                 Type: PositiveInteger
(14) -> P := matrix [[2,3], [4,5]]

           +2  3+
     (14)  |    |
           +4  5+
                                                    Type: Matrix(Integer)
```

```
(15) -> P*M - M*P

          + 1    3 +
   (15)   |        |
          +- 3  - 1+
                                                    Type: Matrix(Integer)
(16) -> % * (-2)

          +- 2  - 6+
   (16)   |        |
          + 6    2 +
                                                    Type: Matrix(Integer)
(17) -> matrix [ [a**i+j for i in 1..3] for j in 10..12]

          +          2         3      +
          |a + 10  a  + 10  a  + 10|
          |                          |
   (17)   |          2         3     |
          |a + 11  a  + 11  a  + 11|
          |                          |
          |          2         3     |
          +a + 12  a  + 12  a  + 12+
                                          Type: Matrix(Polynomial(Integer))
```

3.6 Векторы

Векторы создаются функцией `vector`. Вектор имеет фиксированное число элементов (в отличие от списка или массива [1]) любого, но только одного типа. Если для элементов массива определены операции сложения, вычитания, умножения на скаляр, то они определены и для вектора. Число элементов вектора можно узнать, добавив к нему символ `#`. Доступ к элементам вектора — `вектор(номер)`. Векторы можно умножать на матрицы. Скалярное произведение векторов — `dot`, векторное — `cross`. Полный список операций с векторами, как обычно, можно узнать по команде `)show Vector` (пример 19).

Пример 19. Векторы

```
(1) -> a:=vector [1,2,3]

   (1)  [1,2,3]
                                          Type: Vector PositiveInteger
(2) -> b:=vector [x,y,z]

   (2)  [x,y,z]
                                Type: Vector OrderedVariableList [x,y,z]
(3) -> a+b

   (3)  [x + 1,y + 2,z + 3]
                                       Type: Vector Polynomial Integer

```

```
(4) -> d*a

   (4)  [d,2d,3d]
                                                Type: Vector Polynomial Integer
(5) -> %-b

   (5)  [- x + d,- y + 2d,- z + 3d]
                                                Type: Vector Polynomial Integer
(6) -> m:=matrix [[1,2,3], [4,5,6], [7,8,9]]

        +1  2  3+
        |       |
   (6)  |4  5  6|
        |       |
        +7  8  9+
                                                     Type: Matrix Integer
(7) -> m*b

   (7)  [3z + 2y + x,6z + 5y + 4x,9z + 8y + 7x]
                                                Type: Vector Polynomial Integer

(8) -> length b
   There are 3 exposed and 6 unexposed library operations named length
      having 1 argument(s) but none was determined to be applicable.
      Use HyperDoc Browse, or issue
                              )display op length
      to learn more about the available operations. Perhaps
      package-calling the operation or using coercions on the arguments
      will allow you to apply the operation.

   Cannot find a definition or applicable library operation named
      length with argument type(s)
                     Vector(OrderedVariableList([x,y,z]))

      Perhaps you should use "@" to indicate the required return type,
      or "$" to specify which version of the function you need.

(9) -> length a

        +--+
   (9)  \|14
                                                     Type: AlgebraicNumber
(10) -> length (b::Vector Polynomial Integer)

          +------------+
          | 2    2    2
   (10)  \|z  + y  + x
                                                  Type: Expression Integer
(11) -> b(1) + b(2)

   (11)  y + x
                                                 Type: Polynomial Integer
(12) -> #a

   (12)  3
                                                     Type: PositiveInteger

```

```
(13) -> dot (a, b)

   (13)  3z + 2y + x
                                                        Type: Polynomial Integer
(14) -> cross (a, b)

   (14)  [2z - 3y,- z + 3x,y - 2x]
                                                  Type: Vector Polynomial Integer
```

3.7 Римские числа

Римские числа были добавлены в «Аксиому» в `MCMLXXXVI` году для обозначения производных высших порядков. Напечатать число в римской системе можно с помощью функции `roman` либо объявив их тип как `RomanNumeral` (или кратко, заглавными, `ROMAN`). Римские числа могут быть использованы наравне с арабскими, например, в матрицах, многочленах и проч. Всё, что можно делать с римскими числами в «Аксиоме», покажет команда `)show RomanNumeral` (пример 20).

Пример 20. Римские числа

```
(1) -> f:=operator 'f

   (1)  f
                                                        Type: BasicOperator
(2) -> D(f t, t, 4)

          (iv)
   (2)  f    (t)

                                                     Type: Expression(Integer)
(3) -> a : RomanNumeral := 9

   (3)  IX
                                                          Type: RomanNumeral

(4) -> b : RomanNumeral
                                                                  Type: Void
(5) -> b:=11

   (5)  XI
                                                          Type: RomanNumeral
(6) -> a+b

   (6)  XX
                                                          Type: RomanNumeral
(7) -> a/b

         IX
   (7)   --
         XI
                                                  Type: Fraction(RomanNumeral)

```

```
(8) -> roman(1981)

   (8)  MCMLXXXI
                                                              Type: RomanNumeral
(9) -> M: MATRIX FRAC ROMAN
                                                                   Type: Void
(10) -> M := matrix [[1,2,3], [4,5,6], [7,8,9]]

          + I    II   III+
          |              |
   (10)   |IV    V    VI |
          |              |
          +VII  VIII  IX +
                                          Type: Matrix(Fraction(RomanNumeral))
(11) -> rational (MCMLXXXVI::ROMAN)

   (11)  1986
                                                        Type: Fraction(Integer)
```

3.8 Комплексные числа

«Аксиома» — система символьной алгебры, поэтому некоторые ожидания не оправдываются, но лишь потому, что ожидаются числа. Так, $\sin(1+\mathrm{i})$ — это всего лишь символ. Ожидая увидеть его в виде $a+\mathrm{i}\,b$, человек совершает ошибку, потому что это *другой* символ. Математика как раз занимается отношениями между символами, а «Аксиоме» надо объяснить, что от неё требуется (пример 21).

Пример 21. Комплексные числа

```
(1) -> digits(6)

   (1)  20
                                                      Type: PositiveInteger
(2) -> a:=2+3*%i

   (2)  2 + 3%i
                                                      Type: Complex Integer

(3) -> b:=1-2*%i

   (3)  1 - 2%i
                                                      Type: Complex Integer
(4) -> a*b

   (4)  8 - %i
                                                      Type: Complex Integer
(5) -> a/b

          4   7
   (5)  - - + - %i
          5   5
                                              Type: Complex Fraction Integer

```

```
(6) -> sqrt(a)

          +-------+
   (6)  \|2 + 3%i
                                                  Type: Expression Complex Integer
(7) -> sqrt(a::Complex Float)

   (7)  1.67415 + 0.895978 %i
                                                               Type: Complex Float
(8) -> sin(2+%i)

   (8)  sin(2 + %i)
                                                  Type: Expression Complex Integer
(9) -> sin((2+%i)::COMPLEX FLOAT)

   (9)  1.40312 - 0.489057 %i
                                                               Type: Complex Float
(10) -> M := matrix [[a, a^2], [b^2, b]]

          + 2 + 3%i    - 5 + 12%i+
   (10)   |                       |
          +- 3 - 4%i    1 - 2%i   +
                                                    Type: Matrix Complex Integer
(11) -> M' := inverse M

          +       1       - 3 + 2%i +
          |- ----------   ---------- |
          |  17 + 19%i     15 + 5%i  |
   (11)   |                          |
          |  1 - 2%i           %i    |
          | ----------    - ---------|
          + 17 + 19%i       15 + 5%i+
                              Type: Union(Matrix Fraction Complex Integer,...)
(12) -> M' * M

          +1  0+
   (12)   |    |
          +0  1+
                                           Type: Matrix Fraction Complex Integer
```

3.9 Решение уравнений

«Аксиома» может работать с корнями многочленов (алгебраическими числами), просто зная, что они корни какого-то многочлена, не прибегая к их явному виду (пример 22). Функция `rootOf` означает корень указанного многочлена, `rootsOf` — все его корни. Функции `zeroOf` и `zerosOf` аналогичны, но часто выражают корень с помощью радикалов.

Пример 22. Корни многочленов

```
(1) -> a := rootOf(x^2+x+1, x)

   (1)  x
                                                    Type: Expression Integer

```

```
(2) -> a^2 + a +1

   (2)  0
                                                        Type: Expression Integer
(3) -> a^2 + a

   (3)  - 1
                                                        Type: Expression Integer
(4) -> rootsOf(x^2+x+1, x)

   (4)  [%x0,- %x0 - 1]
                                                   Type: List Expression Integer
(5) -> %x0^3

   (5)  1
                                                        Type: Expression Integer
(6) -> a := zerosOf(x^2+x+1, x)

          +---+         +---+
         \|- 3  - 1 - \|- 3  - 1
   (6)  [----------,------------]
              2            2
                                                   Type: List Expression Integer
```

Линейные уравнения и их системы решаются с помощью функции `solve`. Если система уравнений вырождена (имеет множество решений), функция `solve` покажет это, введя параметры (независимые переменные). Систему уравнений можно также задавать в матричном виде (пример 23). В этом случае результат возвращается в виде частного (*particular*) решения неоднородного уравнения и базиса (*basis*).[1)]

Пример 23. Решение линейных уравнений

```
(1) -> solve ([x+y=a, x-y=b], [x,y])

              b + a    - b + a
   (1)  [[x= -----,y= -------]]
                2         2
                        Type: List List Equation Fraction Polynomial Integer

(2) -> solve([[1, 1], [1, -1]], [a, b])

                       b + a - b + a
   (2)  [particular= [-----,-------],basis= [[0,0]]]
                         2      2
(3) -> solve ([x+y+z=a, x-y-z=b, -x+y-z=c], [x,y,z])

              b + a    c + a    - c - b
   (3)  [[x= -----,y= -----,z= -------]]
                2        2         2
                        Type: List List Equation Fraction Polynomial Integer
(4) -> solve([x+y+z=0], [x,y,z])

   (4)  [[x= - %B - %A,y= %A,z= %B]]
                        Type: List List Equation Fraction Polynomial Integer
```

[1)] Ср. с дифференциальными уравнениями на с. 44.

С помощью функции `radicalSolve` можно попробовать выразить корни многочленов и их систем через радикалы (пример 24).

Пример 24. Решение многочленов в радикалах

```
(1) -> f(x) == x^2+10*x-1
                                                                  Type: Void
(2) -> radicalSolve(f x = 0)
   Compiling function f with type Variable x -> Polynomial Integer

                 +--+          +--+
   (2)  [x= - \|26  - 5,x= \|26  - 5]
                                        Type: List Equation Expression Integer
(3) -> g(x) == x^3+(3/4)*x-1
                                                                  Type: Void
(4) -> radicalSolve(g x = 0)
   Compiling function g with type Variable x -> Polynomial Fraction
      Integer

   (4)
                       +----------+2                          +----------+2
            +---+     3| +--+                     +---+     3| +--+
       (- \|- 3  + 1)\|\|17  + 4   + 2     (- \|- 3  - 1)\|\|17  + 4   - 2
   [x= ------------------------------, x= ------------------------------,
                          +----------+                          +----------+
              +---+     3| +--+                     +---+     3| +--+
         (2\|- 3  + 2)\|\|17  + 4             (2\|- 3  - 2)\|\|17  + 4
        +----------+2
       3| +--+
       \|\|17  + 4   - 1
    x= -----------------]
           +----------+
          3| +--+
          2\|\|17  + 4
                                        Type: List Equation Expression Integer
```

Вот так выглядит последний результат, если попросить «Аксиому» выводить его в формате TEX (как это сделать, см. пример 3 на с. 9):

$$\left[x=\frac{\left(-\sqrt{-3}+1\right)\sqrt[3]{\sqrt{17}+4}^{2}+2}{\left(2\sqrt{-3}+2\right)\sqrt[3]{\sqrt{17}+4}},\ x=\frac{\left(-\sqrt{-3}-1\right)\sqrt[3]{\sqrt{17}+4}^{2}-2}{\left(2\sqrt{-3}-2\right)\sqrt[3]{\sqrt{17}+4}},\ x=\frac{\sqrt[3]{\sqrt{17}+4}^{2}-1}{2\sqrt[3]{\sqrt{17}+4}}\right]$$

Для численного поиска корней многочленов и их систем используется опять-таки функция `solve`, которой во втором параметре указывается желаемая точность результата. Функция `complexSolve` аналогично ищет также и комплексные корни (пример 25).

Пример 25. Численное решение многочленов и их систем

```
(1) -> f(x,y) == x^4 - y^4
                                                                  Type: Void
(2) -> g(x,y) == 2*x^2 - y^4
                                                                  Type: Void

```

```
(3) -> solve([f(x,y)=0, g(x,y)=0], 1/10)

   (3)
        368985    184665       377955      175005          377955    175005
   [[y= ------,x= ------], [y= ------,x= - ------], [y= - ------,x= ------],
        262144    131072       262144      131072          262144    131072
          368985      184665
    [y= - ------,x= - ------], [y= 0,x= 0]]
          262144      131072
                        Type: List List Equation Polynomial Fraction Integer
(4) -> complexSolve([f(x,y)=0, g(x,y)=0], 1/10)

   (4)
          89089    7446621       7543809     8099
   [[y= -------- + ------- %i,x= ------- - ------- %i],
        10485760   5242880       5242880   2621440
            89089    7446621          7543809     8099
    [y= - -------- - ------- %i,x= - ------- + ------- %i],
          10485760   5242880          5242880   2621440
            89089    7446621          7543809     8099
    [y= - -------- + ------- %i,x= - ------- - ------- %i],
          10485760   5242880          5242880   2621440
          89089    7446621       7543809     8099
    [y= -------- - ------- %i,x= ------- + ------- %i],
        10485760   5242880       5242880   2621440
        368985      184665       368985    184665          377955    175005
  [y= - ------,x= - ------], [y= ------,x= ------], [y= - ------,x= ------],
        262144      131072       262144    131072          262144    131072
        377955      175005
    [y= ------,x= - ------], [y= 0,x= 0]]
        262144      131072
              Type: List List Equation Polynomial Complex Fraction Integer
(5) -> numeric(f(-175005/131072, 377955/262144))

   (5)  - 1.1431032112 020925644
                                                                  Type: Float
(6) -> numeric(g(-175005/131072, 377955/262144))

   (6)  - 0.7557404726 5300695665
                                                                  Type: Float
```

3.10 Интегралы

Неопределённые и определённые интегралы берутся с помощью функции `integrate`.

3.10.1 Неопределённые интегралы (первообразные)

Интегралы от элементарных функций часто не выражаются через элементарные же функции (например, $\int \frac{\sin x}{x}\,\mathrm{d}x$). В таких случаях «Аксиома» выводит формальный интеграл, но только лишь если может доказать его «неэлементарность»; иначе она сообщает об ошибке.

Если вид ответа зависит от знаков выражений, входящих в исходное выражение, например, интеграл ниже,[1)] то функция `integrate` покажет все возможные варианты (пример 26). Функция `complexIntegrate` этим не заморачивается, считая все переменные и функции комплексными.

$$\int \frac{\mathrm{d}x}{ax^2+bx+c} = /\mathcal{D} = b^2-4ac/ =$$
$$= \frac{1}{a}\int \frac{\mathrm{d}x}{\left(x+\frac{b}{2a}\right)^2 - \frac{\mathcal{D}}{4a^2}} = \begin{cases} \frac{1}{\sqrt{\mathcal{D}}}\ln\left|\frac{2ax+b-\sqrt{\mathcal{D}}}{2ax+b+\sqrt{\mathcal{D}}}\right| + C & (\mathcal{D}>0) \\ -\frac{2}{2ax+b} + C & (\mathcal{D}=0) \\ \frac{2}{\sqrt{-\mathcal{D}}}\operatorname{arctg}\frac{2ax+b}{\sqrt{-\mathcal{D}}} + C & (\mathcal{D}<0) \end{cases}$$

Пример 26. Неопределённые интегралы

```
(1) -> integrate(1/(a*x**2+b*x+c), x)

   (1)
   [
       log
                                                +-----------+
                  2 2                       2  |         2       2        2
                (2a x  + 2a b x - 2a c + b )\|- 4a c + b   + (8a c - 2a b )x
             +
                                3
                4a b c - b
          /
                 2
              a x  + b x + c
    /
         +-----------+
         |          2
        \|- 4a c + b
     ,
                        +---------+
                        |        2
           (2a x + b)\|4a c - b
    2atan(---------------------)
                          2
                 4a c - b
    -----------------------------]
              +---------+
              |        2
             \|4a c - b
                                    Type: Union(List Expression Integer,...)

```

[1)] Сто́ит заметить, что второй случай равен пределу первого при $\mathcal{D} \to 0$ (см. пример 28 на с. 40).

```
(2) -> integrate(cos(x)/x,x)

   (2)  Ci(x)
                                                Type: Union(Expression Integer,...)
(3) -> integrate(cos(x)/x^2,x)

           x
          ++  cos(%I)
   (3)    |   ------- d%I
          ++       2
                 %I
                                                Type: Union(Expression Integer,...)
(4) -> integrate(x^2*cos(x^3),x)
              3
         sin(x )
   (4)   -------
              3
                                                Type: Union(Expression Integer,...)
(5) -> integrate(1/(sin(x)^2*cos(x)),x)
                    sin(x) + cos(x) + 1                sin(x) - cos(x) - 1
         sin(x)log(--------------------) - sin(x)log(--------------------) - 1
                         cos(x) + 1                         cos(x) + 1
   (5)  ------------------------------------------------------------------------
                                       sin(x)
                                        Type: Union(Expression Integer,...)
(6) -> integrate(1/(sin(x)*cos(x)^3),x)
                2      sin(x)              2        2cos(x)           2
         2cos(x) log(----------) - 2cos(x) log(- ----------) - cos(x)  + 1
                       cos(x) + 1                  cos(x) + 1
   (6)  ------------------------------------------------------------------------
                                          2
                                    2cos(x)
                                       Type: Union(Expression Integer,...)
(7) -> integrate(x^2/(1+x),x)
                        2
         2log(x + 1) + x  - 2x
   (7)   ---------------------
                    2
                                       Type: Union(Expression Integer,...)
(8) -> integrate((log(x)/x)^2,x)
                  2
         - log(x)  - 2log(x) - 2
   (8)   ------------------------
                     x
                                       Type: Union(Expression Integer,...)

(9) -> integrate(x*atan(x)^2,x)
               2           2            2x   2              2x
         4log(x  + 1) + (x  + 1)atan(------)  + 4x atan(------)
                                      2                   2
                                     x  - 1              x  - 1
   (9)   ------------------------------------------------------
                                    8
                                       Type: Union(Expression Integer,...)

```

```
(10) -> complexIntegrate(1/(a*x**2+b*x+c), x)

   (10)
                                      +-----------+
                                    2 |       1
                          (4a c - b ) |- ---------  + 2a x + b
      +-----------+                   |           2
      |       1                      \|  4a c - b
      |- --------- log(-------------------------------------)
      |           2                          2a
    \|  4a c - b
  +
                                        +-----------+
                                      2 |       1
                            (- 4a c + b ) |- ---------  + 2a x + b
      +-----------+                       |           2
      |       1                          \|  4a c - b
    - |- --------- log(-----------------------------------------)
      |           2                            2a
    \|  4a c - b
                                                        Type: Expression Integer
```

3.10.2 Определённые интегралы

«Аксиома» проверяет, определена ли функция на указанном интервале. Если это не так, она выводит сообщение об ошибке. Для несобственных интегралов (в том числе и для «потенциально» несобственных в силу наличия параметров) «Аксиома» пишет слово `potentialPole` и отказывается вычислять интеграл, если её не успокоить ☺ (пример 27).

Осторожно! Интеграл $\int_0^{100\pi} \sqrt{1-\cos 2x}\,\mathrm{d}x = 200\sqrt{2}$ вычислен неправильно, и «Аксиому» можно понять — она забыла модуль: $\sqrt{1-\cos 2x} = \sqrt{2}\,|\sin x|$.[1)] О том, как с помощью «Аксиомы» упрощать громоздкие выражения, см. на с. 48.

Пример 27. Определённые интегралы

```

(1) -> integrate(x*sin(x), x = 0 .. 2*%pi)

   (1)  - 2%pi
                  Type: Union(f1: OrderedCompletion Expression Integer,...)

(2) -> integrate(x^6*sin(x), x = 0 .. 2*%pi)

                6          4          2
   (2)  - 64%pi  + 480%pi  - 1440%pi
                  Type: Union(f1: OrderedCompletion Expression Integer,...)

```

[1)] Интересно сравнить с «Ма́ксимой» (Maxima): она правильно нашла и определённый интеграл, но для первообразной выдала страшное выражение, которое необходимо было упростить.

```
(3) -> integrate(%e^(-x), x = 0 .. %plusInfinity)

   (3)  1
                    Type: Union(f1: OrderedCompletion Expression Integer,...)
(4) -> integrate(sqrt(1-cos(2*x)), x=0..100*%pi)

   (4)  potentialPole
                                          Type: Union(pole: potentialPole,...)
(5) -> integrate(sqrt(1-cos(2*x)), x=0..100*%pi, "noPole")

   (5)  0
                    Type: Union(f1: OrderedCompletion Expression Integer,...)
(6) -> integrate(1/(x^2-2*x*cos(a)+1), x=-1..1, "noPole")

                     +-------------+          +-------------+
                     |        2                |        2
                    \|- cos(a)  + 1          \|- cos(a)  + 1
           - atan(-----------------) + atan(-----------------)
                       cos(a) - 1               cos(a) + 1
   (6)  [0,-------------------------------------------------]
                               +-------------+
                               |        2
                              \|- cos(a)  + 1
             Type: Union(f2: List OrderedCompletion Expression Integer,...)
```

3.11 Пределы

Функция `limit` предназначена для нахождения пределов и хорошо справляется с неопределённостями типа $0/0$ или 1^∞. Символы `%infinity`, `%plusInfinity` и `%minusInfinity` означают ∞, $+\infty$ и $-\infty$. Если пределы в точке слева и справа не совпадают, «Аксиома» покажет оба. «Аксиома» не ошиблась, написав, что предел $\lim\limits_{x\to-\infty}[\ln(2x+1)-\ln(x+2)]$ не существует — она строго следовала определению предела. Логарифм отрицательного числа как вещественное число не существует, зато существует как комплексное. Однако в комплексной плоскости про этот предел сказали бы, что он вообще не существует (зависит от направления). Функция `limit` предполагает, что переменная вещественна, в отличие от `complexLimit` (пример 28).

Пример 28. Пределы

```
(1) -> ex := (1/sqrt(d)) * log ((2*a*x +b - sqrt(d))/(2*a*x +b + sqrt(d)))

                 +-+
              - \|d  + 2a x + b
         log(------------------)
                +-+
               \|d  + 2a x + b
   (1)  ----------------------
                    +-+
                   \|d
                                                    Type: Expression Integer
```

```
(2) -> limit ((a^x-1)/x,x=0)

   (2)  log(a)
                        Type: Union(OrderedCompletion Expression Integer,...)
(3) -> limit(ex, d=0)

                                                              2
   (3)  [leftHandLimit= "failed",rightHandLimit= - --------]
                                                            2a x + b
(4) -> a1 := (1+1/n)^n

          n + 1 n
   (4)  (-----)
             n
                                                     Type: Expression Integer
(5) -> limit(a1, n=%plusInfinity)

   (5)  %e
                        Type: Union(OrderedCompletion Expression Integer,...)
(6) -> a2 := (a1/%e)^n

           n + 1 n n
          (-----)
             n
   (6)  (--------)
             %e
                                                     Type: Expression Integer
(7) -> limit(a2, n=%plusInfinity)

          1
   (7)  ---
          1
          -
          2
         %e
                        Type: Union(OrderedCompletion Expression Integer,...)

(8) -> limit (log(2*x+1) - log(x+2),x=%plusInfinity)

   (8)  log(2)
                        Type: Union(OrderedCompletion Expression Integer,...)
(9) -> limit (log(2*x+1) - log(x+2),x=%minusInfinity)

   (9)  "failed"
                                                   Type: Union("failed",...)
(10) -> limit ((1+sin(x))^(1/x),x=0)

   (10)  %e
                        Type: Union(OrderedCompletion Expression Integer,...)
```

3.12 Ряды

Ряды создаются функцией `series`. Они также являются символами, и «Аксиома» умеет их складывать, умножать и проч. Число `n` членов ряда, выводимого в результате (изначально — 10), задаётся командой `)set`

`streams calculate n`. Функция `series` работает с более общим случаем рядов — рядами Пюизе (*Puiseux series*), у которых показатели степеней рациональны, то есть могут быть дробными.[1)] Для создания рядов Тейлора используется функция `taylor`.

Пример 29. Ряды

```
(1) -> )set streams
                  Current Values of  streams  Variables

Variable     Description                                          Current Value
------------------------------------------------------------------------------
calculate    specify number of elements to calculate    10
showall      display all stream elements computed       off

(1) -> )set streams calculate 6
(1) -> )set streams
                  Current Values of  streams  Variables

Variable     Description                                          Current Value
------------------------------------------------------------------------------
calculate    specify number of elements to calculate    6
showall      display all stream elements computed       off

(1) -> series(1/(1-x), x=0)

                  2    3    4    5    6      7
   (1)  1 + x + x  + x  + x  + x  + x  + O(x )
                       Type: UnivariatePuiseuxSeries(Expression Integer,x,0)
(2) -> series(log(1-x), x=0)

              1  2   1  3   1  4   1  5   1  6   1  7      8
   (2)  - x - - x  - - x  - - x  - - x  - - x  - - x  + O(x )
              2      3      4      5      6      7
                       Type: UnivariatePuiseuxSeries(Expression Integer,x,0)
(3) -> series((8+x)^(1/3), x=0)

   (3)
      1     1  2     5   3      5    4      11    5       77     6      7
 2 + --x - ---x  + -----x  - ------ x  + ------- x  - --------- x  + O(x )
     12    288     20736     248832      5971968      429981696
                       Type: UnivariatePuiseuxSeries(Expression Integer,x,0)
(4) -> series(%e^cos(x), x=0)

              %e  2   %e  4   31%e  6      7
   (4)  %e - -- x  + -- x  - ---- x  + O(x )
              2       6       720
                       Type: UnivariatePuiseuxSeries(Expression Integer,x,0)
(5) -> a := series(tan(x), x=0)

             1  3    2  5    17  7      8
   (5)  x + - x  + -- x  + --- x  + O(x )
             3      15      315
                       Type: UnivariatePuiseuxSeries(Expression Integer,x,0)
```

[1)] http://en.wikipedia.org/wiki/Puiseux_series

```
(6) -> b:=series(exp(x), x=0)

                 1  2   1  3    1  4    1   5    1   6      7
   (6)  1 + x + - x  + - x  + -- x  + --- x  + --- x  + O(x )
                 2      6      24      120      720
                       Type: UniversalPuiseuxSeries(Expression Integer,x,0)
(7) -> a*b

              2   5  3   1  4    41  5    71  6    137  7      8
   (7)  x + x  + - x  + - x  + --- x  + --- x  + ---- x  + O(x )
                 6      2      120      360      1008
                       Type: UniversalPuiseuxSeries(Expression Integer,x,0)
(8) -> a/b

              2   5  3   1  4    41  5    71  6    137  7      8
   (8)  x - x  + - x  - - x  + --- x  - --- x  + ---- x  + O(x )
                 6      2      120      360      1008
                       Type: UniversalPuiseuxSeries(Expression Integer,x,0)
(9) -> a+b

                 1  2   1  3    1  4    17  5    1   6      7
   (9)  1 + 2x + - x  + - x  + -- x  + --- x  + --- x  + O(x )
                 2      2      24      120      720
                       Type: UniversalPuiseuxSeries(Expression Integer,x,0)
(10) -> exp(a)

                 1  2   1  3   3  4    37  5    59  6      7
   (10)  1 + x + - x  + - x  + - x  + --- x  + --- x  + O(x )
                 2      2      8      120      240
                       Type: UniversalPuiseuxSeries(Expression Integer,x,0)
(11) -> sin(a)

              1  3    1  5    55   7      8
   (11)  x + - x  - -- x  - ---- x  + O(x )
              6      40      1008
                       Type: UniversalPuiseuxSeries(Expression Integer,x,0)
(12) -> sin_ser := series(sin(x), x=0)

              1  3    1   5     1   7      8
   (12)  x - - x  + --- x  - ---- x  + O(x )
              6      120      5040
                       Type: UniversalPuiseuxSeries(Expression Integer,x,0)
(13) -> cos_ser := series(cos(x), x=0)

              1  2    1  4    1   6      7
   (13)  1 - - x  + -- x  - --- x  + O(x )
              2      24      720
                       Type: UniversalPuiseuxSeries(Expression Integer,x,0)

(14) -> sin_ser * cos_ser

              2  3    2  5    4   7      8
   (14)  x - - x  + -- x  - --- x  + O(x )
              3      15      315
                       Type: UniversalPuiseuxSeries(Expression Integer,x,0)

```

```
(15) -> series(sin(2*x)/2, x=0)

                2  3    2  5    4   7      8
   (15)  x - - x  + -- x  - --- x  + O(x )
                3      15      315
                          Type: UnivariatePuiseuxSeries(Expression Integer,x,0)
```

3.13 Дифференциальные уравнения

«Аксиома» умеет в конечном виде (без рядов) решать обыкновенные дифференциальные уравнения в двух случаях: линейные уравнения и уравнения в полных дифференциалах, в том числе с интегрирующим множителем. При этом не надо заботиться об интегрирующем множителе: «Аксиома» всё сделает сама. Проверка показала, что она также решает и системы таких уравнений (пример 34 на с. 48). Помимо этих двух случаев, она может искать решения в виде рядов, в том числе и для систем дифференциальных уравнений. Случай разделения переменных вообще можно не считать «дифуром» ☺. Об особенностях решения дифференциальных уравнений см. [18, 19].

Решение в конечном виде ищется с помощью функции `solve`. Для линейного уравнения результат возвращается[1)] в виде частного (*particular*) решения неоднородного уравнения и базиса (*basis*) решения однородного уравнения (пример 30 на противоположной странице). Если же уравнение оказывается уравнением в полных дифференциалах, то ответ выводится в виде неявной функции, которую надо приравнять к постоянной (постоянная интегрирования, пример 31 на с. 46). Решение в виде рядов ищется с помощью функции `seriesSolve`. При этом можно решать нелинейные уравнения любого порядка (пример 33 на с. 47). Решение систем дифференциальных уравнений показано в примере 34 на с. 48. Функция `solve` может также решить задачу Коши — дифференциальное уравнение с начальными условиями — приятная мелочь (пример 32 на с. 46).

Ещё раз сто́ит сказать: компьютер вообще и «Аксиома» в частности лишь помогают человеку, выполняя рутинные вещи, которым он их научил. Надеяться, что программа сможет решить все его проблемы, может разве что школьник-двоечник. А за решением серьёзного уравнения лучше обратиться к справочнику (метод Градштейна–Рыжика ☺).

[1)] Ср. с линейными уравнениями на с. 34.

3.13.1 Линейные дифференциальные уравнения

Неизвестная функция является неким абстрактным оператором (пример 13 на с. 22). Здесь для удобства введено обозначение для этого оператора и созданы производные функции (пример 30).

Пример 30. Линейные дифференциальные уравнения

```
(1) -> y := operator 'y

   (1)  y
                                                          Type: BasicOperator
(2) -> y'(x) == D(y x, x)
                                                                Type: Void
(3) -> y''(x) == D(y x, x, 2)
                                                                Type: Void
(4) -> solve(y'(x) + y(x) = 1, y, x)
   Compiling function y' with type Variable x -> Expression Integer

                                    - x
   (4)  [particular= 1,basis= [%e   ]]
          Type: Union(Record(particular: Expression Integer,basis: List ...))
(5) -> solve(y'(x) + y(x) = x, y, x)

                                      - x
   (5)  [particular= x - 1,basis= [%e   ]]
          Type: Union(Record(particular: Expression Integer,basis: List ...))
(6) -> solve(y'(x) + x*y(x) = x, y, x)

                                      2
                                     x
                                   - --
                                      2
   (6)  [particular= 1,basis= [%e    ]]
          Type: Union(Record(particular: Expression Integer,basis: List ...))
(7) -> solve(y''(x) + y(x) = 1, y, x)
   Compiling function y'' with type Variable x -> Expression Integer

                             2         2
   (7)  [particular= sin(x)  + cos(x) ,basis= [cos(x),sin(x)]]
          Type: Union(Record(particular: Expression Integer,basis: List ...))
(8) -> solve(x*y''(x) + y(x) = 1, y, x)

   (8)  [particular= 1,basis= []]
          Type: Union(Record(particular: Expression Integer,basis: List ...))
(9) -> solve(y''(x) + a^2*y(x) = 0, y, x)

   (9)  [particular= 0,basis= [cos(a x),sin(a x)]]
          Type: Union(Record(particular: Expression Integer,basis: List ...))
(10) -> solve(y''(x) + y'(x) + y(x) = 0, y, x)

                                            x      x
                                      +-+   - -    - -      +-+
                                     x\|3     2      2    x\|3
   (10)  [particular= 0,basis= [cos(-----)%e    ,%e   sin(-----)]]
                                       2                     2
          Type: Union(Record(particular: Expression Integer,basis: List ...))

```

```
51
52 (11) -> solve(x^2*y''(x) + x*y'(x) + y(x) = 0, y, x)
53
54                                        +---+          +---+
55                                    - \|- 1 log(x)   \|- 1 log(x)
56    (11)  [particular= 0,basis= [%e               ,%e             ]]
57           Type: Union(Record(particular: Expression Integer,basis: List ...))
```

3.13.2 Уравнения в полных дифференциалах

Например, уравнение $(x+2y)\,\mathrm{d}y+(x+y)\,\mathrm{d}x=0$ записано в виде $y'=-\frac{x+y}{x+2y}$. Определение `y'(x)` — в примере 30 на предыдущей странице.

Пример 31. Уравнения в полных дифференциалах

```
1  (1) -> y'(x) = - (x+y(x)) / (x+2*y(x))
2
3            ,      - y(x) - x
4     (1)  y (x)= ----------
5                  2y(x) + x
6                                                  Type: Equation Expression Integer
7  (2) -> solve(%, y, x)
8
9                2                2
10          2y(x)  + 2x y(x) + x
11    (2)   ---------------------
12                    2
13                                              Type: Union(Expression Integer,...)
14 (3) -> y'(x)= -(x*sin y(x) + y(x) * cos y(x)) / (x*cos y(x) - y(x)*sin y(x))
15
16           ,      x sin(y(x)) + y(x)cos(y(x))
17    (3)  y (x)= ---------------------------
18                y(x)sin(y(x)) - x cos(y(x))
19                                               Type: Equation Expression Integer
20 (4) -> solve(%, y, x)
21
22                     x                    x
23    (4)  (- x + 1)%e sin(y(x)) - y(x)%e cos(y(x))
24                                            Type: Union(Expression Integer,...)
```

3.13.3 Задача Коши (уравнения с начальными условиями)

Определения `y'(x)` и `y''(x)` см. в примере 30 на предшествующей странице. При поиске решения задачи Коши для уравнения в полных дифференциалах ответ надо приравнивать к нулю.

Пример 32. Задача Коши для дифференциальных уравнений

```
1 (1) -> solve(y''(x) + y(x) = 0, y, x=0, [1,1])
2
3    (1)  sin(x) + cos(x)
4                                          Type: Union(Expression(Integer),...)
5
```

```
(2) -> solve(y''(x) + y(x) = 0, y, x=0, [1,0])

   (2)  cos(x)
                                                Type: Union(Expression(Integer),...)
(3) -> solve(y''(x) + y(x) = 0, y, x=0, [0,0])

   (3)  0
                                                Type: Union(Expression(Integer),...)
(4) -> solve(y'(x) = - (x+y(x)) / (x+2*y(x)), y, x=0, [0])

             2              2
         2y(x)  + 2x y(x) + x
   (4)  ---------------------
                   2
                                                Type: Union(Expression(Integer),...)
(5) -> solve(y'(x) = - (x+y(x)) / (x+2*y(x)), y, x=0, [1])

             2              2
         2y(x)  + 2x y(x) + x  - 2
   (5)  -------------------------
                     2
                                                Type: Union(Expression(Integer),...)
(6) -> solve(y'(x) = - (x+y(x)) / (x+2*y(x)), y, x=0, [2])

             2              2
         2y(x)  + 2x y(x) + x  - 8
   (6)  -------------------------
                     2
                                                Type: Union(Expression(Integer),...)
```

3.13.4 Решение дифференциальных уравнений в виде рядов

Решение дифференциальных уравнений в виде рядов необходимо является задачей Коши. Определения `y'(x)` и `y''(x)` см. в примере 30 на с. 45.

Пример 33. Решение дифференциальных уравнений в виде рядов

```
(1) -> )set streams calculate 7
(2) -> seriesSolve(y'(x) = x + y(x), y, x=0, [1])
   Compiling function %BH with type List UnivariateTaylorSeries(
      Expression Integer,x,0) -> UnivariateTaylorSeries(Expression
      Integer,x,0)

                  2   1  3    1  4    1  5    1   6     1   7      8
   (2)  1 + x + x  + - x  + -- x  + -- x  + --- x  + ---- x  + O(x )
                     3      12      60      360      2520
                       Type: UnivariateTaylorSeries(Expression Integer,x,0)
(3) -> seriesSolve(y''(x) - x*y(x) = 0, y, x=0, [y0, y'0])
   Compiling function %BF with type List UnivariateTaylorSeries(
      Expression Integer,x,0) -> UnivariateTaylorSeries(Expression
      Integer,x,0)

                     y0  3   y'0  4    y0  6   y'0  7      8
   (3)  y0 + y'0 x + -- x  + --- x  + --- x  + --- x  + O(x )
                      6       12      180     504
                       Type: UnivariateTaylorSeries(Expression Integer,x,0)
```

```
(4) -> seriesSolve(y''(x) - 2*x*y'(x) + 2*n*y(x)= 0, y, x=0, [a, b])
   Compiling function %BD with type List UnivariateTaylorSeries(
      Expression Integer,x,0) -> UnivariateTaylorSeries(Expression
      Integer,x,0)

   (4)
                                                2                2
                    2   - b n + b  3   a n  - 2a n  4   b n  - 4b n + 3b  5
     a + b x - a n x  + --------- x  + ----------- x  + ---------------- x
                              3               6                  30
   +
           3       2                   3       2
     - a n  + 6a n  - 8a n  6   - b n  + 9b n  - 23b n + 15b  7      8
     -------------------- x  + --------------------------- x  + O(x )
               90                              630
                        Type: UnivariateTaylorSeries(Expression Integer,x,0)
```

3.13.5 Решение систем дифференциальных уравнений

Однородная система из двух уравнений $y'(t) = x$ и $x'(t) = -y$ имеет решение $x\,(t) = C_1 \cos t + C_2 \sin t$ и $y\,(t) = C_1 \sin t - C_2 \cos t$.

Неоднородная система из двух уравнений $x'(t) + y(t) - x(t) = \frac{3}{2}\,t^2$ и $y'(t) + 2\,y(t) + 4\,x(t) = 1 + 4t$ имеет решение $y\,(t) = C_1 e^{2t} + C_2 e^{-3t} + t^2 + t$ и $x\,(t) = -C_1 e^{2t} + C_2 e^{-3t}/4 - t^2/2$. Определение `y'(t)` см. в примере 30 на с. 45, аналогично определена `x'(t)`.[1)]

Пример 34. Системы дифференциальных уравнений

```
(1) -> solve([x'(t) = -y(t), y'(t) = x(t)], [x, y], t)

   (1)  [particular= [0,0],basis= [[cos(t),sin(t)],[sin(t),- cos(t)]]]
                Type: Union(Record(particular: Vector Expression Integer,...)
(2) -> solve([x'(t) + y(t) - x(t) = (3/2)*t^2,_
     y'(t) + 2*y(t) + 4*x(t) = 1+4*t], [y, x], t)

                                    2                                    - 3t
                          2        t               2t     2t     - 3t %e
   (2)  [particular= [t  + t,- --],basis= [[%e   ,- %e  ],[%e    ,------]]]
                                    2                                    4
                Type: Union(Record(particular: Vector Expression Integer,...)
```

3.14 «Упрощение» выражений

Вообще, следует понимать, что «упрощения» не бывает; бывает преобразование, «переписывание» выражений с помощью других символов.[2)]

[1)] Символ подчёркивания означает, что выражение продолжается на следующей строке.

[2)] В конце концов, можно любое сложное выражение «упростить», обозначив его новым символом. Например, $\Gamma(x) \equiv \int_0^{+\infty} t^{x-1} e^{-t}\,\mathrm{d}t$ и $\mathrm{B}(x,y) \equiv \int_0^1 t^{x-1}\,(1-t)^{y-1}\,\mathrm{d}t$ — ничего особенного. Однако связь этих символов $\mathrm{B}\,(x,y) = \frac{\Gamma(x)\Gamma(y)}{\Gamma(x+y)}$ — уже интереснее.

Многие такие преобразования «Аксиома» делает сама, на то она и система компьютерной аглебры. В основу такой системы заложены разумные, очевидные грамотному человеку правила, типа $\sin\pi \to 0$, $\left(\sqrt{2}\right)^2 \to 2$ или $\frac{1}{2}+\frac{3}{4} \to \frac{5}{4}$. Эти правила выполняются автоматически, но иногда требуется уточнить желаемые преобразования, и «Аксиома» предоставляет такую возможность. Так как «внешний вид» выражения определяется его типом, то часто может помочь простая смена типа выражения.

Для «упрощения» выражений могут быть полезны функции: `simplify` — для некоторых очевидных сокращений; `normalize` — для приведения к некоторому каноническому виду.[1)]

Есть функции, делающие некоторые специальные преобразования, например: `factor` — разложить многочлен (или число) на множители; `expand` — для обратного преобразования (перемножения); `factorFraction` — разложить на множители числитель и знаменатель рациональной дроби; `rootSimp` — вынести всё, что можно, из-под радикалов;[2)] `sin2csc` — заменить синусы на косекансы; `expandLog` — «растащить» логарифмы произведения и частного; `complexElementary` — представить синусы и косинусы в виде комплексной экспоненты (`trigs` — наоборот); аналогично, `realElementary` и `htrigs` — вещественная экспонента и гипергеометрические функции.

Наконец, функции `subst` и `rule` позволяют переписать выражение заменой переменных или более сложным сопоставлением с образцом (пример 36 на с. 51),

Пример 35. «Упрощение» выражений

```
(1) -> a:=1+%i; b:=1-2*%i; a/b

           1   3
   (1)  - - + - %i
           5   5
                                                  Type: Complex Fraction Integer
(2) -> %::Fraction Complex Integer

         - 1 + %i
   (2)  --------
          2 + %i
                                                  Type: Fraction Complex Integer
(3) -> factor (x^4+x^2+1)

           2           2
   (3)  (x  - x + 1)(x  + x + 1)
                                                Type: Factored Polynomial Integer

```

[1)] Для тригонометрических функций она замечена в тотальном использовании универсальной тригонометрической подстановки.

[2)] Автоматически это не делается, потому что, например, $\sqrt{x^2} \neq x$

```
(4) -> expand %

          4    2
   (4)  x  + x  + 1
                                                    Type: Polynomial Integer
(5) -> factor 90699264

          9 11
   (5)  2 3
                                                      Type: Factored Integer
(6) -> 2^9 * 3^11

   (6)  90699264
                                                       Type: PositiveInteger
(7) -> expand ((x+y+z)^3)

         3             2      2             2      3       2     2     3
   (7)  z  + (3y + 3x)z  + (3y  + 6x y + 3x )z + y  + 3x y  + 3x y + x
                                              Type: Expression AlgebraicNumber
(8) -> normalize ((sin x)^2 + 2 + (cos x)^2)

   (8)  3
                                                    Type: Expression Integer

(9) -> normalize (3*(sin x)^2 + 2 + (cos x)^2)

             x 4          x 2
         3tan(-)  + 14tan(-)  + 3
             2            2
   (9)  ------------------------
             x 4         x 2
          tan(-)  + 2tan(-)  + 1
             2           2
                                                    Type: Expression Integer
(10) -> f := cos(x)/sec(x) * log(sin(x)**2/(cos(x)**2+sin(x)**2))

                                2
                           sin(x)
          cos(x)log(-----------------)
                            2         2
                     sin(x)  + cos(x)
   (10)  ----------------------------
                        sec(x)
                                                    Type: Expression Integer
(11) -> g := simplify f

                2            2
   (11)  cos(x) log(- cos(x)  + 1)
                                                    Type: Expression Integer

```

```
(12) -> h := sin2csc cos2sec g

                    2
               sec(x)  - 1
          log(-----------)
                       2
                 sec(x)
   (12)  ----------------
                      2
                sec(x)
                                                    Type: Expression Integer
(13) -> expandLog h

                    2
          log(sec(x)  - 1) - 2log(sec(x))
   (13)  -------------------------------
                          2
                       sec(x)
                                                    Type: Expression Integer
(14) -> htrigs %e^x

                              x
   (14)  (sinh(1) + cosh(1))
                                                    Type: Expression Integer
(15) -> htrigs exp(x)

   (15)  sinh(x) + cosh(x)
                                                    Type: Expression Integer
(16) -> sqrt((x+1)^4)

           +------------------------+
           | 4     3     2
   (16)  \|x  + 4x  + 6x  + 4x + 1
                                                    Type: Expression Integer
(17) -> rootSimp %

           2
   (17)  x  + 2x + 1
                                                    Type: Expression Integer
```

Пример 36. Подстановки и замены

```
(1) -> expr := x^3*(1-x)^(1/3)

          3 3+-------+
   (1)  x  \|- x + 1
                                                    Type: Expression Integer
(2) -> expr := x^3*(1-x)^(1/3)*dx

             3 3+-------+
   (2)  dx x  \|- x + 1
                                                    Type: Expression Integer
(3) -> sub := 1 - t^2

           2
   (3)  - t  + 1
                                                    Type: Polynomial Integer

```

```
(4) -> subst(subst(expr, x=sub), dx=D(sub,t))

                                  +--+
           7     5     3       3| 2
   (4)  (2t  - 6t  + 6t  - 2t)\|t
                                                        Type: Expression Integer
(5) -> logrule := rule (log(x) + log(y) == log(x * y);_
    x*log(y) == log(y^x);_
    log(x) - log(y) == log(x / y))

   (5)
                                                                    x
   {log(y) + log(x) + %I == log(x y) + %I, x log(y) == log(y ),
                                    x
    - log(y) + log(x) + %J == log(-) + %J}
                                    y
                             Type: Ruleset(Integer,Integer,Expression Integer)
(6) -> logrule ((x+2)*log(x+2) - 2*(x+1)*log(x+1)+x*log(x))

               2            x       - 2x       x
             (x  + 4x + 4)x (x + 1)    (x + 2)
   (6)  log(-----------------------------------)
                          2
                         x  + 2x + 1
                                                        Type: Expression Integer
```

3.15 Построение графиков

«Аксиома» может рисовать двумерные графики явных функций одной переменной; графики функций, заданных параметрически; графики неявных функций, заданных многочленами (т. н. алгебраические кривые), однако лишь без особых точек; а также трёхмерные кривые и поверхности Всё это делается с помощью одной функции `draw`, соответственно:

```
draw(f(x), x=a..b)
draw(curve(x(t), y(t)), t=a..b)
draw(p(x, y)=0, x, y, range==[x1..x2, y1..y2])
draw(f(x, y), x=a..b, y=c..d)
draw(curve(x(t), y(t), z(t)), t=a..b)
```

Функция `draw` имеет и другие формы использования, а также множество параметров, управляющих представление графиков, приводить которые (графики и параметры) здесь не имеет смысла (см. [1]).[1)]

[1)] Конечно, есть более мощные программы и языки построения графиков, позволяющие получить очень аккуратные и приятные для глаз и печати изображения. Например, свободные gnuplot (`http://ru.wikipedia.org/wiki/Gnuplot`) и Asymptote (`http://ru.wikipedia.org/wiki/Asymptote`). Однако «прикинуть», как выглядит функция, можно не покидая интерактивную сессию «Аксиомы». В отличие от формул, графики рисуются «по-настоящему» (появляется новое окошко).

4 Задачи посложнее

Этот раздел написан литературно [7], как и сама «Аксиома». Это доставило автору удовольствие и избавило от лишних хлопот. В качестве инструмента литературного программирования использован набор программ `noweb` [8].

4.1 Квантовая механика: $3j$-символы

$3j$-символы используются в квантовой механике при описании сложения моментов импульса. Общее выражение для них громоздко и содержит неприятность в виде суммы [20, § 106]:

$$\begin{pmatrix} j_1 & j_2 & j_3 \\ m_1 & m_2 & m_3 \end{pmatrix} = \sqrt{\frac{(j_1+j_2-j_3)!\,(j_1-j_2+j_3)!\,(-j_1+j_2+j_3)!}{(j_1+j_2+j_3+1)!}} \times$$
$$\times \sqrt{(j_1+m_1)!\,(j_1-m_1)!\,(j_2+m_2)!\,(j_2-m_2)!\,(j_3+m_3)!\,(j_3-m_3)!} \times$$
$$\times \Big[\sum_z \frac{(-1)^{z+j_1-j_2-m_3}}{z!\,(j_1+j_2-j_3-z)!} \cdot$$
$$\cdot \frac{1}{(j_1-m_1-z)!\,(j_2+m_2-z)!\,(j_3-j_2+m_1+z)!\,(j_3-j_1-m_2+z)!} \Big]$$

4.1.1 Описание вычислений

Все необходимые функции записываются в файл `3j.input`, полный текст которого приведён на следующей странице. Сумма в приведённой выше формуле конечна в силу того, что факториал от отрицательного числа равен бесконечности (факториал — частный случай гамма-функции). Для вычисление этой суммы напишем отдельную функцию `j3Sum`, в которой определим пределы суммирования так, чтобы факториалы были от положительных чисел — остальные слагаемые равны нулю.[1)]

53 ⟨*3j.input* 53⟩≡ 54a▷

```
j3Sum (j1, j2, j3, m1, m2, m3) ==
  maxz := reduce (min, [j1+j2-j3, j1-m1, j2+m2])
  minz := max(0, max ( -(j3-j2+m1), -(j3-j1-m2) ))
  minz > maxz => 0
  maxz < 0    => 0
  sum ( (-1)^(z+j1-j2-m3) / _
    ( factorial(z) * factorial(j1+j2-j3-z) * factorial(j1-m1-z) * _
      factorial(j2+m2-z) * factorial(j3-j2+m1+z) * factorial(j3-j1-m2+z) ), _
    z=minz..maxz)
```

[1)] Вместо `reduce(min, list)` можно было бы использовать два `min`. Функции `min` и `max` — бинарные.

Функция, вычисляющая $3j$-символы. В ней явно выпишем частные случаи, точнее, «нулевые» случаи. Можно также дополнить и другими, более простыми частными формулами [20, § 106].

54a ⟨*3j.input* 53⟩+≡ ◁53 54b▷

```
j3 (j1, j2, j3, m1, m2, m3) ==
  m1 + m2 + m3 ~= 0  => 0
  abs(j1 - j2) > j3  => 0
  j1 + j2 < j3       => 0
  abs(m1) > j1       => 0
  abs(m2) > j2       => 0
  abs(m3) > j3       => 0
  not integer? (j1+j2+j3) => 0
  sqrt ( _
    factorial(j1+j2-j3) * factorial(j1-j2+j3) * factorial(-j1+j2+j3) / _
                          factorial(j1+j2+j3+1) * _
        factorial(j1+m1) * factorial(j1-m1) * _
        factorial(j2+m2) * factorial(j2-m2) * _
        factorial(j3+m3) * factorial(j3-m3)
    ) * j3Sum (j1, j2, j3, m1, m2, m3)
```

Коэффициенты Клебша–Гордана связаны с $3j$-символами.

54b ⟨*3j.input* 53⟩+≡ ◁54a

```
clebschGordan (j1, j2, j, m1, m2, m) ==
    (-1)^(j1-j2+m) * sqrt(2*j+1) * j3(j1, j2, j, m1, m2, -m)
```

4.1.2 Полный текст программы

Пример 37. Файл «3j.input»

```
1  j3Sum (j1, j2, j3, m1, m2, m3) ==
2    maxz := reduce (min, [j1+j2-j3, j1-m1, j2+m2])
3    minz := max(0, max ( -(j3-j2+m1), -(j3-j1-m2) ))
4    minz > maxz => 0
5    maxz < 0    => 0
6    sum ( (-1)^(z+j1-j2-m3) / _
7      ( factorial(z) * factorial(j1+j2-j3-z) * factorial(j1-m1-z) * _
8        factorial(j2+m2-z) * factorial(j3-j2+m1+z) * factorial(j3-j1-m2+z) ), _
9      z=minz..maxz)
10
11 j3 (j1, j2, j3, m1, m2, m3) ==
12   m1 + m2 + m3 ~= 0  => 0
13   abs(j1 - j2) > j3  => 0
14   j1 + j2 < j3       => 0
15   abs(m1) > j1       => 0
16   abs(m2) > j2       => 0
17   abs(m3) > j3       => 0
18   not integer? (j1+j2+j3) => 0
19   sqrt ( _
20     factorial(j1+j2-j3) * factorial(j1-j2+j3) * factorial(-j1+j2+j3) / _
21                           factorial(j1+j2+j3+1) * _
22         factorial(j1+m1) * factorial(j1-m1) * _
```

```
23          factorial(j2+m2) * factorial(j2-m2) * _
24          factorial(j3+m3) * factorial(j3-m3)
25      ) * j3Sum (j1, j2, j3, m1, m2, m3)
26
27  clebschGordan (j1, j2, j, m1, m2, m) ==
28      (-1)^(j1-j2+m) * sqrt(2*j+1) * j3(j1, j2, j, m1, m2, -m)
```

4.1.3 Пример работы с программой

Для проверки можно обратиться к «ландафшицу» [20, § 106], а численные значения можно быстрой найти в Википедии.[1)]

Пример 38. Сеанс работы с файлом «3j.input»

```
1  (1) -> )read 3j
2
3  (5) -> clebschGordan(3/2, 1, 3/2, 1/2, 0, 1/2)
4
5             1
6     (5)  -----
7           +--+
8          \|15
9                                                    Type: Expression Integer
10 (6) -> clebschGordan(3/2, 1, 1/2, 1/2, 0, 1/2)
11
12            +-+
13           \|2
14    (6)  - ----
15            +-+
16           \|6
17                                                   Type: Expression Integer
18 (7) -> clebschGordan(3/2, 1, 5/2, 1/2, 0, 1/2)
19
20           +-+ +-+
21          \|2 \|6
22    (7)  --------
23              +-+
24           2\|5
25                                                   Type: Expression Integer
```

4.2 Общая теория относительности

Математика общей теории относительности проста, но её много. Большинство вычислений элементарны и рутинны, но большое их количество может приводить к многочисленным глупым ошибкам, да и просто утомительно.

Посмотрим, как с этим справляется «Аксиома», а именно — с вычислением символов Кристоффеля, тензоров Римана и Риччи и других величин общей теории относительности при известном (полностью или частично) метрическом тензоре. Конкретно, ниже идёт речь в основном о решении Шварцшильда. Теория изложена в «ландафшице» [21].

[1)] http://en.wikipedia.org/wiki/Table_of_Clebsch-Gordan_coefficients

4.2.1 Описание вычислений

Все функции и определения записываются в файл `gr.input`, полный текст которого приведён на с. 59. Сначала определяем символы координат и размерность пространства.

56a ⟨*gr.input* 56a⟩≡ 56b▷
```
x   := vector ['t, 'r, '%theta, '%phi];
dim := #x;
```

Метрический тензор g_{ik} содержит неизвестные функции $\nu(r)$ и $\lambda(r)$.[1)]

56b ⟨*gr.input* 56a⟩+≡ ◁56a 56c▷
```
%nu     := operator '%nu;
%lambda := operator '%lambda;
lg := matrix [
    [exp(%nu r),       0,          0,    0], _
    [       0,  - exp(%lambda r), 0,    0], _
    [       0,      0,          -r^2,   0], _
    [       0,      0,            0,  -r^2*sin(%theta)^2]  _
    ];
```

Нам также понадобится полностью контравариантный тензор g^{ik}.

56c ⟨*gr.input* 56a⟩+≡ ◁56b 56d▷
```
ug := inverse lg;
```

Для того, чтобы иметь возможность работать и с другими метриками, не покидая интерактивную сессию, создадим функцию, с помощью которой можно задавать метрику и названия координат. Слово `free` означает глобальную переменную («свободную»).

56d ⟨*gr.input* 56a⟩+≡ ◁56c 56e▷
```
grSetup(metric, names) ==
    free x
    free dim
    free lg
    free ug
    x   := names
    dim := #x
    lg  := metric
    ug  := inverse lg
```

Так как при вычислениях часто потребуется суммирование, определим для удобства функцию, вычисляющую сумму элементов списка.

56e ⟨*gr.input* 56a⟩+≡ ◁56d 57a▷
```
sum(list) == reduce (+, list)
```

[1)] Физический смысл см. в [21, § 100], зависимость от времени опускаем.

Символы Кристоффеля $\Gamma^i_{kl} = \frac{1}{2} g^{im} \left(\frac{\partial g_{km}}{\partial x^l} + \frac{\partial g_{ml}}{\partial x^k} - \frac{\partial g_{kl}}{\partial x^m} \right)$, в которых подразумевается сумма по m, вычислим, создав список для различных m и просуммировав все его элементы.[1)]

57a ⟨*gr.input* 56a⟩+≡ ◁56e 57b▷

```
christoffel (k,l,i) ==
 (1/2) * sum [ ug(i,m)*(D(lg(k,m), x(l)) + D(lg(m,l), x(k)) _
               - D(lg(k,l), x(m)))
         for m in 1..dim ]
```

В тензоре Римана (он же — тензор кривизны) $R^i_{klm} = \frac{\partial \Gamma^i_{km}}{\partial x^l} - \frac{\partial \Gamma^i_{kl}}{\partial x^m} + + \Gamma^i_{nl}\Gamma^n_{km} - \Gamma^i_{nm}\Gamma^n_{kl}$ имеется сумма по n.

57b ⟨*gr.input* 56a⟩+≡ ◁57a 57c▷

```
riemann (k,l,m,i) ==
 D(christoffel(k,m,i), x(l)) -
  D(christoffel(k,l,i), x(m)) +
   sum [
    christoffel(n,l,i)*christoffel(k,m,n) -
     christoffel(n,m,i)*christoffel(k,l,n)
      for n in 1..dim ]
```

Тензор Риччи $R_{ik} = R^l_{ilk}$ является свёрткой тензора Римана. Так и запишем.

57c ⟨*gr.input* 56a⟩+≡ ◁57b 57d▷

```
ricci (i,k) == sum [ riemann(i,l,k,l) for l in 1..dim ]
```

Скалярная кривизна $R = R^i_i = g^{ik} R_{ik}$.

57d ⟨*gr.input* 56a⟩+≡ ◁57c 57e▷

```
scalarCurvature () == sum [ sum [
                       ug(i,k) * ricci(i,k)
                        for i in 1..dim ]  for k in 1..dim ]
```

Наконец, для исследования свойств тензора Римана полезно иметь его в полностью ковариантном виде $R_{iklm} = g_{in} R^n_{klm}$. Для демонстрации возможностей «Аксиомы» зададим его свойства симметрии явно, хотя в этом и нет необходимости, ведь считать всё равно будет компьютер.[2)]

57e ⟨*gr.input* 56a⟩+≡ ◁57d 58a▷

```
lRiemann (i,i,l,m) == 0
lRiemann (i,k,l,l) == 0
lRiemann (i,k,l,m | i > k) == - lRiemann (k,i,l,m)
lRiemann (i,k,l,m | l > m) == - lRiemann (i,k,m,l)
lRiemann (i,k,l,m) == sum [ lg(i,n) * riemann(k,l,m,n) for n in 1..dim ]
```

[1)] Обсуждение способов суммирования см. в списке рассылки `http://sourceforge.net/mailarchive/forum.php?forum_name=open-axiom-help` (искать слово `summation`).

[2)] Однако этим мы лишим себя возможности действительно проверить наличие этих свойств, так как они уже заданы заранее. Желающие поэкспериментировать могут удалить всё, кроме последней строки.

Многие символы Кристоффеля, компоненты тензоров Риччи и Римана равны нулю. Напишем функции, которые выводят только ненулевые величины. Заодно при выводе сдвинем нумерацию индексов так, чтобы она начиналась с нуля, и сгруппируем с учётом симметрии.[1)]

58a ⟨*gr.input* 56a⟩+≡ ◁57e 58b▷

```
showChristoffel () ==
 for k in 1..dim repeat
  for l in 1..k repeat
   for i in 1..dim repeat
    if christoffel(k,l,i) ~= 0 then
        k > l => output infix ('=, [script('%Gamma,[[k-1,l-1],[i-1]]), _
                    script('%Gamma,[[l-1,k-1],[i-1]]), _
                    christoffel(k,l,i)::OUTFORM])
        k = l => output infix ('=, _
                  [script('%Gamma,[[k-1,l-1],[i-1]]), _
                   christoffel(k,l,i)::OUTFORM])
```

58b ⟨*gr.input* 56a⟩+≡ ◁58a 58c▷

```
showRicci () ==
 for i in 1..dim repeat
   for k in 1..i repeat
    if ricci(i,k) ~= 0 then
        i = k => output infix ('=, [subscript('R,[i-1,k-1]),
                                    ricci(i,k)::OUTFORM])
        i > k => output infix ('=, [subscript('R,[i-1,k-1]), _
                                    subscript('R,[k-1,i-1]), _
                                    ricci(i,k)::OUTFORM])
```

58c ⟨*gr.input* 56a⟩+≡ ◁58b

```
showRiemann () ==
 for k in 1..dim repeat
  for l in 1..dim repeat
   for m in 1..dim repeat
    for i in 1..dim repeat
     if riemann(k,l,m,i) ~= 0 then
        output infix ('=, _
          [script('R, [[k-1,l-1,m-1 ], [i-1]]), riemann(k,l,m,i)::OUTFORM])
```

[1)] Апостроф перед символом означает, что это просто символ, и его не надо вычислять. Можно также заключать этот символ в кавычки. А вообще, форматированный вывод в «Аксиоме» достаточно непривычен. Функция `infix` работает так: `infix (a, [x,y,z])` → `x a y a z`. Запись `ricci(i,k)::OUTFORM` означает не математический объект, а его графическое представление. Функции `script`, `subscript` и `superscript` расставляют верхние и нижние индексы.

4.2.2 Полный текст программы

Пример 39. Файл «gr.input»

```
x   := vector ['t, 'r, '%theta, '%phi];
dim := #x;

%nu     := operator '%nu;
%lambda := operator '%lambda;
lg := matrix [
    [exp(%nu r),       0,          0,     0], _
    [       0,  - exp(%lambda r), 0,     0], _
    [       0,       0,          -r^2,   0], _
    [       0,       0,            0,  -r^2*sin(%theta)^2]  _
    ];

ug := inverse lg;

grSetup(metric, names) ==
    free x
    free dim
    free lg
    free ug
    x   := names
    dim := #x
    lg  := metric
    ug  := inverse lg

sum(list) == reduce (+, list)
christoffel (k,l,i) ==
 (1/2) * sum [ ug(i,m)*(D(lg(k,m), x(l)) + D(lg(m,l), x(k)) _
               - D(lg(k,l), x(m)))
         for m in 1..dim ]

riemann (k,l,m,i) ==
 D(christoffel(k,m,i), x(l)) -
  D(christoffel(k,l,i), x(m)) +
   sum [
    christoffel(n,l,i)*christoffel(k,m,n) -
     christoffel(n,m,i)*christoffel(k,l,n)
      for n in 1..dim ]

ricci (i,k) == sum [ riemann(i,l,k,l) for l in 1..dim ]

scalarCurvature () == sum [ sum [
                       ug(i,k) * ricci(i,k)
                        for i in 1..dim ]  for k in 1..dim ]

lRiemann (i,i,l,m) == 0
lRiemann (i,k,l,l) == 0
lRiemann (i,k,l,m | i > k) == - lRiemann (k,i,l,m)
lRiemann (i,k,l,m | l > m) == - lRiemann (i,k,m,l)
lRiemann (i,k,l,m) == sum [ lg(i,n) * riemann(k,l,m,n) for n in 1..dim ]
showChristoffel () ==
 for k in 1..dim repeat
  for l in 1..k repeat
   for i in 1..dim repeat
    if christoffel(k,l,i) ~= 0 then
```

```
55          k > l => output infix ('=, [script('%Gamma,[[k-1,l-1],[i-1]]), _
56                          script('%Gamma,[[l-1,k-1],[i-1]]), _
57                          christoffel(k,l,i)::OUTFORM])
58          k = l => output infix ('=, _
59                           [script('%Gamma,[[k-1,l-1],[i-1]]), _
60                            christoffel(k,l,i)::OUTFORM])
61  showRicci () ==
62   for i in 1..dim repeat
63     for k in 1..i repeat
64      if ricci(i,k) ~= 0 then
65          i = k => output infix ('=, [subscript('R,[i-1,k-1]),
66                                      ricci(i,k)::OUTFORM])
67          i > k => output infix ('=, [subscript('R,[i-1,k-1]), _
68                                      subscript('R,[k-1,i-1]), _
69                                      ricci(i,k)::OUTFORM])
70  showRiemann () ==
71   for k in 1..dim repeat
72    for l in 1..dim repeat
73     for m in 1..dim repeat
74      for i in 1..dim repeat
75       if riemann(k,l,m,i) ~= 0 then
76          output infix ('=, _
77            [script('R, [[k-1,l-1,m-1 ], [i-1]]), riemann(k,l,m,i)::OUTFORM])
```

4.2.3 Пример работы с программой

Очень важный момент — при расчётах много раз потребуются значения функций при одних и тех же параметрах, и чтобы не вычислять их снова и снова (на это требуется время), можно попросить «Аксиому» запоминать их с помощью команды `)set functions cache all`. Для очистки памяти надо использовать команду `)clear`. Пример работы см. ниже.[1)] Ответы аксиомы можно сравнить с «ландафшицом» [21, § 100].

Пример 40. Сеанс работы с файлом «gr.input»

```
1  (1) -> )read gr
2
3  (21) -> showChristoffel()
4                  %nu(r)    ,
5                %e      %nu (r)
6           1
7     %Gamma    = ---------------
8           0,0       %lambda(r)
9                  2%e
10                                ,
11                              %nu (r)
12          0            0
13    %Gamma    = %Gamma    = -------
14          1,0          0,1      2
15                        ,
16               %lambda (r)
17          1
18    %Gamma    = -----------
19          1,1         2
```

[1)] При выполнении команды `)read gr` «Аксиома» выводит содержимое этого файла. Оно здесь не показано, как и другие сообщения.

```
20
21          2           2    1
22    %Gamma    = %Gamma    = -
23          2,1         1,2  r
24
25          1             r
26    %Gamma    = - ------------
27          2,2       %lambda(r)
28                   %e
29          3           3    1
30    %Gamma    = %Gamma    = -
31          3,1         1,3  r
32          3           3    cos(%theta)
33    %Gamma    = %Gamma    = -----------
34          3,2         2,3  sin(%theta)
35                              2
36          1       r sin(%theta)
37    %Gamma    = - --------------
38          3,3       %lambda(r)
39                   %e
40          2
41    %Gamma    = - cos(%theta)sin(%theta)
42          3,3
43                                                                  Type: Void
44  (22) -> ricci(3,3)
45
46                  ,               ,          %lambda(r)
47          - r%nu (r) + r%lambda (r) + 2%e             - 2
48
49    (22)  ---------------------------------------------
50                              %lambda(r)
51                           2%e
52                                                      Type: Expression Integer
53  (23) -> g2 := matrix [[1,0],[0,-v^2]]
54
55          +1   0  +
56    (23)  |       |
57          |      2|
58          +0  - v +
59                                             Type: Matrix Polynomial Integer
60  (24) -> x2 := [v,u]
61
62    (24)  [v,u]
63                                          Type: List OrderedVariableList [v,u]
64  (25) -> grSetup(g2, x2);
65                             Type: Union(Matrix Fraction Polynomial Integer,...)
66  (26) -> showChristoffel()
67          1           1    1
68    %Gamma    = %Gamma    = -
69          1,0         0,1  v
70          0
71    %Gamma    = v
72          1,1
73                                                                  Type: Void
74  (27) -> scalarCurvature()
75
76    (27)  0
77                                               Type: Fraction Polynomial Integer
```

5 Заключение

Результат, полученный с помощью систем компьютерной алгебры с закрытым кодом, не может считаться частью математического доказательства, так как сам код проверить невозможно.

«Аксиома» — система компьютерной алгебры с открытым исходным кодом.

Список примеров

Источники

1. Axiom: the scientific computation system. — Vol. 0. Axiom Jenks and Sutor. — 1187 pp. http://axiom-developer.org/. См. с. 4, 5, 6, 13, 19, 29 и 52.

2. *Joyner David.* Axiom // *ACM Communications in Computer Algebra.* — 2008. — Vol. 42, no. 2. — Pp. 39–47. http://www.sigsam.org/bulletin/issues/issue164.html. См. с. 4 и 24.

3. *Beebe Nelson H. F.* A Bibliography of Publications about the AXIOM (formerly, Scratchpad) Symbolic Algebra Language. — Salt Lake City, USA: University of Utah, 2007. — 17 pp. http://www.math.utah.edu/ftp/pub/tex/bib/axiom.ps.gz. См. с. 4.

4. Axiom: the scientific computation system. — Vol. 9. Axiom compiler. — 35 pp. http://axiom-developer.org/. См. с. 5 и 11.

5. Axiom sandbox. http://axiom-wiki.newsynthesis.org/SandBox. См. с. 5 и 13.

6. *Reis Gabriel Dos, Mai Stefan.* Foreign Function Interface for the OpenAxiom Scientific Computation Platform. — College Station, Texas 77843, USA: Texas A&M University, Department of Computer Science and Engineering, 2009. http://parasol.tamu.edu/~gdr/OpenAxiom/ffi-draft.pdf. См. с. 5.

7. *Knuth Donald E.* Literate programming // *The computer journal.* — 1984. — Vol. 27, no. 2. — Pp. 97–111. http://www.literateprogramming.com/. См. с. 5, 6 и 53.

8. *Johnson Andrew L., Johnson Brad C.* Literate programming using noweb // *Linux Journal.* — 1997. — no. 42. — Pp. 64–69. http://www.cs.tufts.edu/~nr/noweb/. См. с. 5 и 53.

9. *Рассел Б.* Введение в математическую философию. Избранные работы. — Новосибирск: Сибирское университетское издательство, 2007. — 264 с. См. с. 5.

10. *Whitehead A. N., Russell B.* Principia Mathematica: set of 3 vols. — 2-nd edition. — Cambridge: University Press, 1927. http://name.umdl.umich.edu/AAT3201.0003.001. См. с. 5.

11. The Rosetta document: comparison of different computer algebra systems. http://axiom-developer.org/. См. с. 7.

12. *Практика функционального программирования.* — 2009. — № 1. http://fprog.ru/. См. с. 7 и 19.

13. *Фихтенгольц Г. М.* Курс дифференциального и интегрального исчисления. В 3 т. — 9, стереотип. изд. — СПб.: Лань, 2009. См. с. 13.

14. *Смирнов И. В.* Курс высшей математики. В 5 т. — 24 изд. — СПб.: БХВ-Петербург, 2008. См. с. 13.

15. Сборник задач и упражнений по математическому анализу: учебное пособие для вузов / Под ред. Б. П. Демидовича. — М.: АСТ: Астрель, 2007. — 558 с. См. с. 13.

16. Задачи и упражнения по математическому анализу для втузов / Под ред. Б. П. Демидовича. — М.: АСТ: Астрель, 2006. — 495 с. См. с. 13.

17. Энциклопедия числовых последовательностей. http://www.research.att.com/~njas/sequences/. См. с. 24.

18. *Robidoux Nicolas.* Does Axiom Solve Systems of O.D.E.'s Like Mathematica? Technical Report LA-UR-93-2235. — Los Alamos, NM: Los Alamos National Laboratory, 1993. http://axiom-portal.newsynthesis.org/refs/articles. См. с. 44.

19. *Seiler W. M.* Applying AXIOM to Partial Differential Equations. Internal Report 95-17. — Karlsruhe: Karlsruhe Institute of Technology, 1995. — 52 pp. http://axiom-portal.newsynthesis.org/refs/articles. См. с. 44.

20. *Ландау Л. Д., Лифшиц Е. М.* Теоретическая физика: учебное пособие для вузов. В 10 т. / Под ред. Л. П. Питаевского. — 5, стереот. изд. — М.: ФИЗМАТЛИТ, 2001. — Т. III. Квантовая механика. — 808 с. См. с. 53, 54 и 55.

21. *Ландау Л. Д., Лифшиц Е. М.* Теоретическая физика: учебное пособие для вузов. В 10 т. / Под ред. Л. П. Питаевского. — 5, стереот. изд. — М.: ФИЗМАТЛИТ, 2006. — Т. II. Теория поля. — 536 с. См. с. 55, 56 и 60.

Все источники, снабжённые ссылками, свободно доступны по ним.

Игорь Николаевич Пашев

Система компьютерной алгебры «Аксиома»

(методические рекомендации)

Подписано в печать 04.08.2010.
Формат 60 × 84 1/16. Бумага офисная.
Объём 4,2 п. л. Тираж 60 экз. Заказ № 209.

Типография ООО «Кира»
163061, г. Архангельск, ул. Поморская, 34
Тел./факс: (8182) 65-47-11
Электропочта: `ookira@atnet.ru`