ISBN 5-94052-066-0 ПРОГРАММНЫЕ СИСТЕМЫ: ТЕОРИЯ И ПРИЛОЖЕНИЯ. ПЕРЕСЛАВЛЬ-ЗАЛЕССКИЙ, 2004

УДК 519.68

С. Д. Мешвелиани

Об исследовании по автоматизации вычислений и доказательств

Аннотация. В статье рассматривается простейший эвристический способ доказательства утверждений в алгебре и программировании, основанный на действиях с равенствами, переписывании термов. Вывод по индукции выражается через добавление равенств. Особенности: "беступиковое" пополнение по Кнуту-Бендиксу, прием распределения запаса шагов. Способ применяется к таким примерам, как высказывание "для любых N, M (N+M = M+N)" для алгоритма сложения натуральных чисел. Рассматривается язык для рассуждений/вычислений в логике равенств. Обсуждается, чего не хватает таким системам для применимости к более содержательным задачам. Библиография содержит 13 ссылок на свободно доступные источники.

Ключевые слова и фразы: переписывание термов, беступиковое пополнение, индуктивное доказательство, компьютерная алгебра.

1. Введение

В области автоматизации научных вычислений важной задачей современности является соединение в одном программном средстве возможности обычных прямых вычислений и рассуждений — доказательств. Это также имеет приложение к оптимизации функциональных программ, к частичным вычислениям.

Предыдущая работа. Развитие языков программирования научных вычислений выражается в том, что они приближаются к обозначениям и способу представления знаний, принятому в книгах по математике. Так, это отразилось в развитии средств функционального и функторного программирования, например, языков ML, Haskell [7], Aldor [1], или системы CA Axiom.

На этом направлении мы развили библиотеку CA под названием `Построитель` (Построитель алгебраических областей, DoCon) [12, 13]. В ней воплощены в большой степени общности методы линейной алгебры, арифметики многочленов, упрощение алгебраических систем, действия над описаниями алгебраических областей и другие. Это есть библиотека, написанная на языке Haskell [7] — функциональном языке высокого уровня, с "ленивым" способом вычисления, функторностью,

полиморфными функциями, параметрическими типами и так далее. Например, в этой системе программирования запись вида

```
class GCDRing a where (+) :: a -> a -> a      -- операции 'кольца'
                      (*) :: a -> a -> a      --
                      gcd :: a -> a -> a      -- и еще НОД
                      ...
class GCDRing a => Field a where inv :: a -> a    -- обращение

instance GCDRing Integer where n+m     = ...   -- сложение целых
                               n*m     = ...
                               gcd n m = ...   -- способ Евклида

instance GCDRing a => Field (Fraction a) where
                                     f+g   = ... -- сложение дробей
                                     f*g   = ...
                                     inv f = ...
```

выражает *намерение* представить программно следующее знание, содержащееся в учебниках алгебры.

“ **(1)** Область с НОД (наибольшим общим делителем) есть любая область ‘a’ с действиями, именуемыми `+`, `*`, `gcd` ... и имеющими указанный тип. **(1.1)** Эти действия обладают свойствами ассоциативности, коммутативности, они связаны дистрибутивностью, ...

(2) Поле (`Field`) есть такая область с НОД, в которой дано дополнительно действие `inv :: a -> a` (одна категория наследует действия из другой), **(2.1)** это действие обладает свойством обращения: `x*(inv x) = 1` для $x \neq 0$.

(3) Область целых чисел `Integer` наделяется случаем категории `GCDRing` при задании данных действий следующим образом ...

(4) Если ‘a’ есть случай категории `GCDRing`, то область дробей над ‘a’, обозначаемая `Fraction a`, наделяется случаями категорий `GCDRing, Field` при задании действий по обычным правилам для дробей. **(4.1)** Если ‘a’ действительно обладает свойствами области с НОД, то область дробей над ‘a’ обладает действиями с такими же свойствами и дополнительно свойством обращения **(2.1)**. ”

В такой системе программирования компилятор проверяет типы действий и наследование наборов действий. Но не берутся в учет явные законы (равенства) — то, что соответствует пунктам **(1.1)**, **(2.1)**, **(3.1)**, **(4.1)**. Например, ассоциативность `(x*y)*z = x*(y*z)` не учтется ни при компиляции, ни при возможном упрощении программы. Так что исходное намерение осуществляется, приблизительно говоря, не более, чем наполовину.

воплощение	реализация
CA	вычислительная (компьютерная) алгебра
CL	лемма о постоянной (правило вывода)
выражение	терм
основное выражение	выражение, не содержащее переменных
IE	устранение импликации (правило вывода)
IP	индуктивный прувер
исполнитель	интерпретатор
KBC	метод Кнута–Бендикса пополнения
OSTRW	упорядоченно-сортное переписывание термов
прувер	программа для (полу)автоматического доказательства теорем
сведе́ние	редукция выражения (к нормальной форме)
TRW	переписывание термов
UKB, ukb	беступиковый метод Кнута–Бендикса
читальщик	парсер
ЯВ, ЯС, ОЯ	(соответственно) язык воплощения, язык стратегий, объектный язык

Таблица 1. Словарь терминов и сокращения

Этим недостатком обладают большинство систем CA. Естественный способ исправить положение состоит в применении языка теории равенства и переписывания термов (TRW) и самой этой теории, и — добавлении прувера, основанного на TRW. Например, в этом смысле внушительно выглядит проект Theorema [3, 4], начатый в середине 1990-х годов.

Вводное чтение. Советуем ознакомиться (в данном порядке) со статьей [10], сборником [2], статьями [9], [6], [5], руководством по системе [11].

Нашу разработку мы назвали Думатель, кратко dm, Dm, (это слово есть некоторая шутка из повести братьев Стругацких).

В дальнейших объяснениях мы будем применять словарь терминов и сокращения, описанные в таблице 1.

2. Краткое описание проекта

2.1. Основные черты.

2.1.1. *Языки.* Выделены объектный язык (ОЯ), язык стратегий (ЯС) и язык воплощения (ЯВ).

В качестве ЯВ выбран Haskell. Вообще, это мог бы быть любой удобный язык программирования. Он же выбран языком стратегий для прувера: ЯС = ЯВ. В качестве ОЯ полагаем язык *упорядоченно-сортного переписывания термов* (OSTRW).

ЯС есть язык, на котором выражаются рассуждения о теориях, записанных на ОЯ.

Модуль (теория, 'программа') на ОЯ есть теория равенства. Он содержит несколько равенств, описание сортов и операторов. Он разбирается читальщиком (`parseTheory`) в данное ЯВ и подается как пруверу, вместе с целевым высказыванием (например, равенством, которое надо доказать).

2.1.2. *Прувер.* Это есть функция (`prove`), которая порождает (согласно стратегии) подцели, преобразует модуль теории, и так далее, пытаясь упростить цели вплоть до выражения вида `x = x` или `true`, после чего цель удаляется. Пруверу дается в аргументе *число шагов*, которое он распределяет по целям и по истечении которого прувер завершает работу, выдавая оставшиеся цели. С технической точки зрения программирования Dm есть записанная на ЯВ стратегия, опирающаяся на исполнитель TRW, библиотеку способов пополнения для TRW и библиотеку CA; все эти части воплощены на одном функциональном языке высокого уровня ЯВ.

2.1.3. *Библиотека CA.* В основание системы ставится библиотека CA (`Построитель`), воплощенная на ЯВ и присоединяемая к пруверу в виде объектного кода. В объектной теории выделяются как встроенные операторы, имеющие воплощение в CA. Встроенный оператор `f` может быть прямо и особенно быстро вычислен на аргументе — основном выражении, а с другой стороны, прувер обрабатывает равенства, данные для `f` в объектной теории.

2.1.4. *Исполнитель TRW.* Исполнитель (`reduce th t`) воплощает сведéние выражения к несводимому виду, написан на ЯВ и принимает теорию равенства в виде явного аргумента.

2.1.5. *Пополнение по Кнуту-Бендиксу.* Функция

```
ukb rc th g
```

беступикового пополнения [9] воплощена на ЯВ и опирается на Исполнитель. Она выдает итог, когда система равенств станет полной, либо целевое равенство свелось к `x = x` по текущей системе, либо иссяк запас шагов. *Итог:* состоит из текущих цели, пополнения и остатка запаса шагов.

Пополнение есть простейший и логически полный способ вывода равенств.

2.1.6. *Индуктивный прувер IP.* Он основан на том, что **(а)** при операторах, отмеченных как построители сорта S, возможны доказательства индукцией по построению значений сорта S, **(б)** индуктивное предположение добавляется к текущей теории в виде равенства.

2.1.7. *Упорядочение выражений.* Функция частичного упорядочения операторов и выражений $>_T$ является важным параметром стратегии, даваемым как часть объектной Теории.

2.1.8. *Алгебраическое равенство.* К каждой теории прувер добавляет оператор равенства `~` и правило `x ~ x = true`. Законы для него, данные пользователем дополнительно, позволяют пруверу пытаться доказывать алгебраическое равенство данных. А знак `'='` из правил переписывания обозначает равенство *записей*. Например, выражения `a+1` и `1+a` не равны как записи, но для многих теорий равны алгебраически. Из смысла этих двух равенств следует *правило замены алгебраического равенства (AS)*: равенство вида `s ~ t = true` заменяется в Теории на `s = t`.

Хотя пополнение сводит алгебраическое равенство к равенству записей, сведенных по пополнению, но полная система не всегда достигается за конечное число шагов.

2.1.9. *Логические связки как обычные операторы.* Логические постоянные и связки `true, false, not, and, ==>, ...` представлены операторами на сорте `Bool`. Соответственно, логические высказывания появляются как подвыражения и подвергаются обычным сведéнию, отождествлению, пополнению.

Применяются такие разновидности пополнения (`ukb`), при которых не возникает тупика при попытке упорядочить такие равенства, как, например, `x+y = y+x`. Поэтому пополнение срабатывает и как вывод в исчислении высказываний.

2.1.10. *Подход запаса шагов.* Пруверу, пополнению и некоторым другим функциям дается в аргументе число шагов, по истечении которого функция завершает работу, выдавая текущий итог. Запас распределяется по подцелям. Такая функция выдает вместе с итогом и

оставшийся запас — он может быть ненулевым, если подцель достигнута.

2.1.11. *Исчисление предикатов, сколемизация.* В данном случае прувер имеет дело с доказательством высказываний наподобие

```
Forall [x,y] (Exist [u,v]  (not u~0  and  (u*x+v*y) ~ 0))
```

Заменяя квантор существования на соответствующие новые операторы, прувер получает возможность доказать высказывание от противного. Так, при построении отрицания в данном примере переменные `x, y` заменяются на новые постоянные `a, b`, добавляемые к Теории. Получается равенство

```
not u~0  and  (u*a+v*b) ~ 0  = false,
```

где `u, v` суть переменные под квантором всеобщности, который подразумевается для всех переменных, входящих в равенство `s = t`, `~` оператор алгебраического равенства. Полученное равенство присоединяется к теории, и пополнение стремится вывести противоречие вида `true = false`.

2.1.12. *Начальная стратегия.* В качестве начального приближения принимается некоторая наивная стратегия вывода (FAS), приведенная ниже. Она пробует в определенном порядке правила вывода из очень малого набора. Благодаря применению разделения запаса шагов она справляется со многими простыми задачами, хотя во многих других случаях беспомощна, и, конечно, прувер нуждается в развитии.

2.2. Примечания.

2.2.1. *Цели.* В общей постановке задача доказательства не разрешима алгоритмически. Как и человек, прувер потерпит неудачу на некоторых задачах, с виду просто выраженных. Неудача проявится в иссякании запаса шагов. Проект имеет цели

- воплотить более мощный язык для научных вычислений — рассуждений, с элементами автоматического упрощения, для чего полезен автоматический вывод хотя бы для простейших задач;
- попытаться усилить стратегию и библиотеку так, чтобы стало возможным решение более содержательных задач.

2.2.2. *Подход запаса шагов.* Он отражает то состояние, в котором чаще пребывает автомат, чем исследователь: когда нет хорошего направляющего замысла, а пока надо попробовать несколько разных

возможностей. Этот подход позволяет не пропустить легкого решения, если такое существует среди выбранных ранее возможностей.

2.2.3. *Отмеченные построители сорта.* Например, объявление `ConstructedBy [0, s]` для сорта `Nat` из Примера 2 Раздела 4.2 определяет, что все значения этого сорта получаются применением данных двух операторов. Все это нужно пруверу для узнавания того, что доказательство индукцией по данным построителям есть доказательство данного утверждения с квантором всеобщности для всего сорта.

Способ построения индуктивного предположения через построители и ведение множества целей мы переняли из работ [5, 11]. Хотя эти средства, видимо, давно известны.

2.2.4. *Индукция, истинность в многообразии, в начальной модели.* Часть (IP) стратегии похожа на способ 'прогонки' (driving), применяемый в частичных вычислениях.

Добавление IP расширяет круг доказуемых высказываний. Объяснения по этому предмету приведены, например, в статье Юэ и Оппена из [2]. С помощью только пополнения, выводимы те равенства, что истинны во всех моделях данной теории: "истинны в многообразии." И это суть итоги конечных выводов, состоящих из подстановок одних равенств в другие (теорема Биркгофа). Метод пополнения позволяет при этом сильно сократить число перебираемых подстановок.

С применением же индукции, возможен вывод всех равенств, истинных в так называемой начальной модели теории. Высказывание *истинно в начальной модели*, когда всякая подстановка в него основных выражений вместо всех переменных дает высказывание истинное *в многообразии*.

Пример. Рассмотрим теорию `natural` из раздела 4.2. Равенство `0+n = n`, очевидно, нельзя вывести только подстановками равенств из этой теории. Но оно истинно в начальной модели, соответственно, существует его индуктивный вывод по построению значения `n`.

2.3. Начальная стратегия. Эта стратегия будет показана на примерах в следующих разделах. Она состоит в выполнении в данном порядке таких действий-попыток.

2.3.1. *(Rc).* Если множество целей пусто, или запас шагов есть 0, то остановиться и выдать текущее множество целей.

Прувер упрощает цели и удаляет цели вида `( ...true )`.

2.3.2. *(R).* Обработать цели по одной. Пусть цель содержит высказывание Ф. Она также включает в себя текущую теорию и неповторимую метку цели.

2.3.3. *(S). Упростить* Ф по равенствам текущей теории и получить Φ' (при этом текущая теория может измениться).

Если Φ' `= true`, то (под)цель удаляется.

2.3.4. *(Im).* Если Φ' есть импликация, применить правило IE — устранение импликации.

2.3.5. *(Const,Ind).* Если Φ' есть `(Forall xs ...)`, то необходимо применить сначала лемму о постоянных (CL), дав на эту подцель половину запаса шагов. Если CL не дала успеха, то применить `Индукцию` — с оставшейся частью запаса. Здесь прувер порождает последовательность `e(1), e(2)`, ... *выражений для индукции*, представляющих различные *функции*, по значению которых возможна индукция.

Начальная стратегия просто выбирает в качестве `e(k)` сначала только *переменные* из текущего высказывания, потом — его подвыражения.

Но в будущем мы собираемся рассматривать более общий способ и эвристики для порождения `e(k)`.

Прувер последовательно пробует подцели для `e(k)` для доказательства индукцией по значению `e(k)` и останавливается, как только одно из этих доказательств преуспело.

2.4. Примечание. Если утверждение доказуемо по индукции, то вовсе не обязательно будет успешным его доказательство индукцией только по значению подвыражений из этого утверждения. Например, утверждение

$$\forall xyf(x,y) = 0$$

может успешно доказываться индукцией по значению $x + y^2$, а индукция по x, y может оказаться бесполезной.

3. Пример теории: булева алгебра

Следующее определение задает теорию в виде данного ЯВ вида `Theory {..}`.

```
bool =
  Theory {
    Sorts     [ Bool  ConstructedBy [true, false] ],
    Operators [
```

```
  true  : -> Bool, false : -> Bool,
  not_  : -> Bool               (Precedences [7]  ),
  _and_ : Bool Bool -> Bool     (Precedences [4,4])
                                (With [Assoc, Comm, Identity true]),
  _==>_ : Bool Bool -> Bool ...,
  _~_   : Bool Bool -> Bool ... (With [Comm]),      -- равенство
  ...
         ],
Variables [[x,y,z] : Bool],
Equations
  [x ~ x           -> true
   not x           -> x xor true,
   x ==> y         -> x and y xor x xor true,
   (x xor y) xor z =  x xor (y xor z),    -- ассоциативность
   x and y         =  y and x,
   ...
  ],
OpPrecedence ["==>" > and > or > xor > not > false > true],
OpWeights    [...],     -- веса операторов
greaterTerm  = lpo ...   -- lex-сравнение выражений
    }
```

Вот пояснения.

3.1. Теория как данное. Dm и программы пользователя действует с теорией как с обычным данным. Например,

```
reduce bool (x or not x) -> true
```

Программа на ЯВ доступается явно к частям теории через устроенный для этого интерфейс. Например, выражения программы

```
signature th, equations th
```

выделяют из теории соответственно описание сортов и операторов, список равенств.

3.2. Сорта. Выражения имеют сорта (типы) согласно объявлениям Теории. Например, переменная, объявленная `x : Bool`, не может принять значение `0 : Nat` — если сорт `Nat` не объявлен как подсорт `Bool`.

3.3. Объявления операторов. Например,

```
_and_ : Bool Bool -> Bool  (Precedences [4,4])
                           (With [Assoc, Comm, Identity true])
```

значит, что

(а) оператор `and` есть инфиксный (для читальщика),

(б) имеет один левый и один правый аргумент сорта `Bool`, имеет значение сорта `Bool`,

(в) имеет предпочтение (для читальщика) 4 для левого аргумента и 4 для правого,

(г) объявлен ассоциативным и коммутативным (помимо добавления возможных явных равенств),

(д) постоянная `true` есть 'единица' для этого оператора.

В частности, такие объявления задают способ чтения выражений из инфиксной записи.

3.4. Теория и программа, сведéние, упорядочение. Dm и прувер (`prove`) понимают Теорию как множество равенств, выводимых из данных равенств (или — как множество моделей ...).

Исполнитель (`reduce`) понимает теорию как *программу* сведéния выражения `t` по системе равенств. *Сведéние* состоит в поиске первого *уменьшающего* применения равенства `u = v`, применения его *слева направо или справа налево* и повторения поиска, пока не получится несводимое выражение. Применение слева направо есть нахождение подстановки σ, сопоставляющей `u` с любым подвыражением в `t`, и замена этого подвыражения на σ `v`. *Особенность*: применяются лишь уменьшающие сопоставления. То есть для применимости слева направо должно выполняться условие σ `u` $>_T \sigma$ `v` для данного в Теории упорядочения $>_T$ [9]. Применение справа налево естественным образом определяется через перестановку `u` $\leftrightarrow$ `v`.

Все это приводит к завершаемости алгоритма `reduce` для знакомым образом записываемых 'программ' — только требуется, чтобы отношение $>_T$ частичного порядка удовлетворяло некоторому естественному ограничению: оно должно быть *полным на основных упрощающим упорядочением* [9].

Порядок попыток применения равенств — не установлен. Для *несоединимых* систем (не полных, не Черч–Россеровых, not confluent) разные пути сведéния могут иметь разные итоги. Например, прувер считает естественным, если `x+1+a-1` по одному пути сводится к `x+a`, а по другому — к `a+x`, и будет использовать дальше эти упрощения. По крайней мере, можно положиться на то, что эти выражения равны в силу этой Теории.

3.5. Задание упорядочения $>_T$. Объявление

```
OpPrecedence [...]
```

упорядочивает операторы и порождает таблицу сравнения `preced`. Объявление вида `greaterTerm = ...` устанавливает функцию для сравнения по порядку $>_T$. Эта функция имеет тип

```
Precedence -> Term -> Term -> Bool
```

и имеет первым аргументом таблицу `preced`. Для некоторых порядков нужны еще *веса* операторов, которые также могут объявляться в теории. Полная *таблица операторов* читается из объявлений, хранит опознавательные метки, сортности и так далее.

3.6. Равенства и правила. Всякое равенство `s = t` автоматически преобразуется:

- в (направленное) *правило* `s -> t`, если `s` $>_T$ `t`;
- в (направленное) *правило* `t -> s`, если `t` $>_T$ `s`.

Равенство `s = t` остается как есть, если `s` и `t` не сравнимы.

Запись `s -> t` значит, что уже известно, что `s` $>_T$ `t`.

4. Примеры

4.1. Пример 2: арифметика натуральных чисел.

```
natural = theoryUnion bool nat'
  where
  nat' = Theory { Sorts [ Nat  ConstructedBy [0, s] ],
                  Operators
                    [0   : -> Nat,
                     s_  : Nat -> Nat      ..., -- "следующий"
                     _+_ : Nat Nat -> Nat  ...,
                     _-_ : Nat Nat -> Nat  ...,
                     _*_ : Nat Nat -> Nat  ...,
                     _~_ : Nat Nat -> Bool ...  -- равенство
                    ],
                  Variables [ [n, m] : Nat ],
                  Equations
                    [n ~ n   = true,      -- равенство

                     n + 0   =  n,       -- сложение в унарной системе
                     n + s m =  s (n + m),

                     n   - 0   =  n,      -- вычитание
                     0   - s m =  0,
                     s n - s m =  n - m
                     ...
                    ],
                  OpPrecedence [ _*_  >  _+_  >  _-_  >  _~_  >  0 ],
```

```
  ...
}
```

Дадим пояснения. Теория `natural` получается объединением

(`theoryUnion`)

теорий `bool` и `nat'`, причем согласно данным опциям разводятся или отождествляются пересечения имен сортов в первой теории и второй, также — и имен операторов, объединяются таблицы сравнения, и так далее. Читатель догадывается, что вышеприведенные равенства задают обычные рекурсивные способы сложения, умножения и вычитания.

Пример.

```
reduce natural ((s s 0 + s s 0 - s 0) ~ s s 0)
```

сведется к `s s s 0 ~ s s 0`, значение `false` не достигается, так как теория не объявила достаточных для такого вывода равенств.

Какую *алгебру* (модель) задает данная теория? Если пренебречь объявлением построителей сорта, то таких алгебр будет много. Например, область рациональных чисел удовлетворяет данным уравнениям. Но объявленные построители сорта выделяют единственную (с точностью до изоморфизма) модель: область натуральных чисел.

Перейдем к примерам доказательства.

4.2. Пример 2.1: `0+n = n`. Трудность состоит в том, что нуль прибавлен "не с той стороны", а закона перестановочности для сложения (нарочно) не задано. Это задача записывается в виде

```
prove rcS rc [Prove InInitial [1] natural (Forall [n] (0+n ~ n))]
```

— проверить истинность высказывания в начальной модели теории `natural`. Это равносильно доказательству с возможным применением индукции.

Аргумент прувера `rcS` есть натуральное число: количество шагов, выделяемое на каждую попытку упрощения (шаг **(S)** стратегии); `rc` есть полный запас шагов; `goals` — список целей. Цель имеет вид

```
Prove видВывода меткаЦели теория высказывание
```

`видВывода = InInitial` значит, что можно применять индукцию. Функция `prove` выдает список оставшихся целей и оставшийся запас шагов.

Применим к этому примеру начальную стратегию: сначала FAS (Раздел 2.3) упрощает цель `[1]`, затрачивая `rcS` шагов. Ни одно из уравнений не применимо, цель не изменилась.

Вообще, будем считать, что пруверу "не везет", и складывать затраты на неудачные попытки. Несмотря на это ему оказываются под силу многие простые задачи.

Дальше Стратегия проверяет высказывание: оно не есть импликация. Применяется *Лемма постоянных (CL)*. Ее математический смысл: если нечто верно для произвольной постоянной, подставленной вместо переменной x, то это верно "для любого x".

Продолжим доказательство. Половина запаса дается на применение (CL). `n` заменяется на новую свободную постоянную `c`, которая присоединяется к теории в виде оператора соответствующего сорта. Получилась *основное* высказывание. К нему применяется исходная стратегия. Никаких упрощений не происходит; переменных больше нет, и потому применяется только вид вывода `InVariety`; ни одно правило из стратегии не применимо, кроме упрощения; запас шагов на данную попытку иссякает; возвращена исходная цель. Согласно стратегии наступает очередь пробовать индукцию.

Индукция. В этом примере первыми выражениями для проведения индукции оказываются `n, 0+n, ...` Далее функция построения индуктивных целей для `n` порождает здесь одну цель для *основания* индукции и одну для *шага* индукции. А вызов заменяется на

```
prove rcS rc' [gBase, gStep]
  where
  gBase = Prove InVariety [1,1] natural (0+0 ~ 0)          -- основание
  gStep = Prove InInitial [1,2] natural                    -- "шаг"
                  (Forall [n] (0+n ~ n ==> 0 + s n ~ s n))
```

Эта запись отражает обычное понимание индуктивного доказательства по построению значения `n : Nat`. Заметим еще, что новые цели получили новые метки. Теперь Упрощатель применяет правило `n+0 -> 0` к высказыванию из цели `[1,1]`, сводя его к `0 = 0`, к `(...true)`, и цель удаляется. Вообще, это есть

Доказательство сведéнием:

если `(reduce th s) == (reduce th t)`,

то равенство `s = t` истинно в теории `th`.

О доказательстве пополнением: оно использует сведéние и построение наложений равенств (критичесих пар). В примерах из данной статьи оказывается достаточной стратегия с таким упрощением **(S)**,

когда наложения оказываются лишними. Но для многих примеров другого вида (допустим, вывод единственности обращения из трех законов теории групп) пополнение оказывается существенным.

Вернемся к примеру. Здесь равенства для сложения сводят цель `[1,2]` к

```
Prove InInitial [1,2] natural (Forall [n] (0+n ~ n ==> s (0+n) ~ s n))
```

С половинным запасом шагов применяется правило CL. Переменная обращается в новую постоянную `c`, которая добавляется к теории. Квантор для `n` удаляется, цель преобразуется в

```
Prove InVariety [1,2] (natural U [c]) (0+c ~ c ==> s (0+c) ~ s c)
```

Применяется *правило импликации IE:*

посылка преобразуется в несколько равенств и переносится в Теорию (разновидность правила modus ponens).

В нашем примере — `0+c ~ c` добавляется к теории в виде

```
0+c ~ c = true.
```

Но правило (AS) (Раздел 2.1) преобразует это в `0+c = c`. Новое состояние прувера есть

```
prove ...[Prove InVariety [1,2] (natural U[c, 0+c = c])(s (0+c)~s c)].
```

Теперь шаг (S) применяет сведéние и по новому правилу. В любом естественном упорядочении $\texttt{0+c} >_T \texttt{c}$, поэтому новое правило сводит целевое высказывание к `s c ~ s c` и к `true`. Цель удаляется, и пустой список целей показывает успех всего доказательства.

Об объеме вычислений. В рассматриваемых здесь примерах, кроме последнего, он мал, если на шаг (S) давать малый запас `rcS`.

4.3. Дальнейшие примеры арифметики. Таким же образом, и с небольшими затратами, прувер доказывает в теории `natural` высказывания

```
Forall [n] (n-n ~ 0),   Forall [n,m] (n+m ~ m+n)
```

При доказательстве второго утверждения возникает вложенная индукция, три уровня подцелей, и среди подцелей есть такая:

```
Forall [n] (0+n ~ n)
```

Естественно, прувер доказывает ее заново — если ему явно не указано хранилище ранее доказанных утверждений.

4.4. Примеры с программами обработки списков. В нижеприводимой теории

sort `List` выражает списки элементов сорта `Elt`,

оператор — построитель `_._` добавляет элемент спереди к списку,

оператор `rev` переворота списка 'запрограммирован' рекурсивно: он переносит очередной первый элемент в конец, применяя оператор соединения `++`,

операторы — постоянные `a, b` введены для задания примеров со списками, составленными из элементов `a, b`.

```
list = theoryUnion bool list'
  where
  list' =
    Theory { Sorts [Elt,                -- "элемент списка"
                    List  ConstructedBy [nil, _._]
                   ],
             Operators
               [_~_   : List List -> Bool ...,    -- равенство
                a     : -> Elt,
                b     : -> Elt,
                nil   : -> List,
                _._   : Elt  List -> List,      -- cons
                _++_  : List List -> List,      -- соединение
                rev_  : List -> List                  -- переворот
               ],
             Variables [ [x, y] : Elt, [xs, ys] : List ],
             Equations
               [xs ~ xs = true,

                nil    ++ ys =  ys,
                (x.xs) ++ ys =  x . (xs++ys),

                rev nil    =  nil,
                rev (x.xs) =  (rev xs)++(x.nil)
               ],
             OpPrecedence [~ > rev > ++ > _._ > nil ...]
             ...
           }
```

Как и в примере `natural`, при естественном упорядочении функция `reduce` действует здесь как привычный программисту интерпретатор программ. Например,

```
reduce list (rev ((a.b.nil)++(b.a.a.nil))) --> a.a.b.b.a.nil
```

Для прувера важное отличие от примера `natural` состоит в том, что построитель числа имеет один аргумент, а построитель списка — два.

Сначала мы рассмотрели пример задачи `xs++nil ~ xs`.

Подобно арифметическому примеру `0+n ~ n`, прувер легко доказал это индукцией по построению `xs`.

Также оказывается успешным доказательство утверждения о перевороте "с конца":

```
Prove InInitial [1] list (Forall [xs,x] ((rev (xs++x)) ~ (x . (rev xs))))
```

4.5. Первая неудача: двойной переворот. При доказательстве утверждения

```
Prove InInitial [1] list  (Forall [xs] (rev rev xs ~ xs))
```

начальная стратегия порождает среди целей индуктивный шаг

```
Prove InInitial [1,2] list
      (Forall [xs] (rev rev xs  ~  xs  ==>
                    Forall [x] (rev ((rev xs)++x)  ~  x.xs)
                   ))
```

Лемма постоянных вводит неопределенную постоянную `xsC : List` и преобразует цель в

```
Prove InInitial [1,2] (list U xsC)
                      (rev rev xsC  ~  xsC  ==>
                       Forall [x] (rev ((rev xsC)++x)  ~  x.xsC))
```

Правило IE присоединяет посылку к теории:

```
Prove InInitial [1,2] (list U xsC U (rev rev xsC = xsC))

                      (Forall [x] (rev ((rev xsC)++x)  ~  x.xsC))
```

Здесь Упрощение не находит напрямую равенств, упрощающих цель, применяется *индукция* уже к более сложному высказыванию:

```
Forall [xs] (rev rev xs ~ xs ==> Forall [x] (rev ((rev xs)++x) ~ x.xs))
```

Далее те же действия повторяются на другом уровне, и всюду возникает под-высказывание вида `rev ((rev xs)++x) ~ x.xs`.

Прувер зацикливается, запас шагов иссякает.

Если же пруверу разрешить пользоваться подсказкой и в качестве таковой подать Лемму

```
rev (ys++x) ~ x.(rev ys),
```

— например, присоединить это равенство к теории, — то пример будет решен. Заметим, что начальная стратегия способна отдельно доказать эту лемму. Но "догадаться" выдвинуть эту лемму в виде подцели она не смогла.

Мы собираемся усилить стратегию так, чтобы этот и многие другие нерешенные примеры стали решаться. Но надо иметь в виду, что во многих случаях выдвижение удачной леммы есть все равно творческая задача большой сложности. Дело здесь обстоит так же, как и с выбором выражения для проведения индукции: пространство выбора в некотором смысле *'бесконечномерно'*, а умных выборов мало.

5. Проблемы

5.1. Хранилище знаний. Естественно, должны пригождаться особые готовые знания (леммы, теоремы) по той или иной предметной области. Например, при доказательстве утверждений для многочленов особые знания о многочленах должны естественным образом увеличивать мощь прувера. Пока мы выражаем знания лишь в виде равенств и в виде башни категорий (и это очень выразительные средства). Прувер должен обмениваться с хранилищем знаний.

Здесь важно *распознавание применимости*. Недостаточно отыскать в хранилище, скажем, теорему Пифагора, надо еще уметь распознать, когда ее применение возможно и к тому же ведет к успеху. Например, математики знают, что часто формула распознается как применимая к случаю лишь, скажем, после какой-либо мудрой подстановки. И, как всегда, в трудных случаях распознавание не состоится.

Надеемся представить такое "распознавание по модулю доказательства" в виде подобной же задачи вывода. Но запас шагов на нее дается с учетом того, что она вызывается в цикле при просмотре хранилища знаний: "облегченный вывод".

5.2. Выдвижение параметра для индукции. Существуют разные эвристики, какую переменную пробовать раньше. Надо обобщить это построение.

5.3. Выдвижение леммы. Об этом уоминалось выше. Данная проблема похожа на предыдущую.

5.4. Выбор упорядочения. В этой области мудрый выбор упорядочения термов еще важнее, чем в CA. Например, от него может зависеть, будет ли прувер выражать оператор `f` как программу вычисления через оператор `g` или наоборот, 'вычислять' `g` через `f`. Это важно, например, для упрощения программ.

5.5. Расширение объектного языка. Мы стремимся к обогащению объектного языка: OSTRW, функторность, операторы, способные строить сорта (алгебраические области), и так далее.

В частности, OSTRW есть обобщение многосортного TRW. Оно значит такое распознавание вложений областей (сортов), при котором распознаются *расширения* одноименных операторов [6, 11]. Это соответствует полезным и давно применяемым математическим обозначениям. Но

- положение в науке таково, что для OSTRW алгоритм пополнения предстоит доразрабатывать;
- предвидим такие же затруднения, например, для операторов высшего порядка.

5.6. AC-пополнение, пополнение в исчислении предикатов. Значительное усиление обещают разновидности пополнения по КВС на классах эквивалентности, например, по модулю ассоциативности и коммутативности (AC). Мы также пытаемся продолжить линию алгебраизации логики [8], применяя и развивая особую разновидность способа базисов Гребнера к поиску вывода в исчислении предикатов.

6. Текущее состояние разработки и намерения

Вышеприведенные примеры выполнялись в выпуске прувера, воплощенном в системе Maude, с сильно недоработанной функцией пополнения.

Теперь же система переведена на язык и инструмент Haskell, и воплощены

(а) читальщик и исполнитель для OSTRW,

(б) довольно основательный метод пополнения для многосортного TRW.

Выдача записи доказательства производится пока лишь через общий режим трассировки программы на ЯВ.

Предстоит

- проверить новый прувер на Начальной стратегии на изложенных выше примерах;
- добавить накопление и выдачу записи доказательства;
- разработать условное TRW для многосортного случая и через него воплотить OSTRW;

- заниматься проблемами, названными в Разделе 5.

7. Сравнение с другими разработками

Здесь мы упоминаем только проекты Maude [5, 6, 11] и Theorema [3, 4], так как они кажутся нам наиболее значимыми.

Пока мы придерживаемся такой линии в отношении других разработок: знакомимся с теоретическими исследованиями, а программное воплощение развиваем только своими средствами, на основе инструмента с открытым исходным кодом и документацией.

Проект Theorema далеко развит: имеет богатый язык условного TRW, богатую библиотеку CA, базу знаний, специальные пруверы, и так далее.

Для ознакомления с этой разработкой наибольшим препятствием для нас сейчас является техническое: этот проект воплощен на основе коммерческого инструмента — системы Mathematica (закрытость исходного кода воплощения, со всеми последствиями).

Проект Maude замечателен развитостью средств OSTRW, сведéния по модулю ассоциативности и коммутативности и самоприменимостью языка. Но

(а) ему не хватает библиотеки CA (очень трудоемкая часть) и способов пополнения;

(б) при попытке воплотить прувер на языке Maude оказалось, что полученная программа выглядит гораздо сложнее, чем наша нынешняя Haskell программа;

(в) наш прием разделения запаса шагов представляется нам важным при стремлении к полностью автоматическим доказательствам.

Список литературы

[1] Watt S. M. et al. // Aldor Compiler User Guide: IBM Thomas J. Watson Research Center, http://www.aldor.org. ↑1

[2] Антимиров В. М., Воронков А. А., Дегтярев А. И., Захарьящев М. В., Проценко В. С. // Математическая логика в программировании: Сборник статей. — Москва: Мир, 1991. ↑1, 2.2.4

[3] Buchberger B. *Mathematica as a rewrite language* // Proceedings of the 2nd Fuji International Workshop on Functional and Logic Programming: Статья в сборнике трудов конференции ред. Ida T., Ohori A., Takeichi M. — Shonan Village Center, Japan: World Scientific, 1996, с. 1–13. ↑1, 7

[4] Buchberger B., Dupré C., Jebelean T., Kriftner F., Nakagawa K., Văsaru D., Windsteiger W. *The TH∃OREM∀ Project: A Progress Report* // Symbolic Computation and Automated Reasoning. Proceedings of CALCULEMUS 2000, Symposium on the Integration of Symbolic Computation and Mechanized Reasoning. August 6-7, 2000, St. Andrews, Scotland, A.K.: Статья в сборнике трудов симпозиума, 2000, с. 98–113, http://www.risc.uni-linz.ac.at/people/buchberg/main_publications.html. ↑1, 7

[5] Clavel M. // The ITP Tool (Inductive Theorem Prover): Программа и руководство по ней, 2000, http://maude.csl.sri.com/tools/itp/, http://sophia.unav.es/~clavel/itp/. ↑1, 2.2.3, 7

[6] Goguen J., Meseguer J. *Order-Sorted Algebra I: Equational Deduction for Multiple Inheritance, Polymorphism, and Partial Operations*, выбрать View or download PS.gz на странице http://citeseer.nj.nec.com/goguen92ordersorted.html,. ↑1, 5.5, 7

[7] // Haskell 98: A Non-Strict, Purely Functional Language. Report of February 1999: Описание языка программирования, http://www.haskell.org. ↑1

[8] Сян Ж. *Доказательство теорем с помощью переписывающих систем* // Кибернетический сборник, Новая серия: Статья в сборнике. Перевод с английского статьи 1985 года. — Выпуск 25. — Москва: Мир, 1988, с. 5–57. ↑5.6

[9] Hsiang J., Rusinowitch M. *On word problems in equational theories* // Proceedings of the Fourteenth International Conference on Automata, Languages and Programming, Karlsruhe, West Germany, July 1987 ред. Ottman Th. — Т. **267**: Springer Verlag, 1987, с. 54–71. ↑1, 2.1.5, 3.4

[10] Knuth D., Bendix P. *Simple word problems in universal algebras* // Computational Problems in Abstract Algebra: Статья в сборнике ред. John Leech: Pergamos Press, 1970, с. 263–297. ↑1

[11] Clavel M., Meseguer J (et al.) // Maude: Specification and Programming in Rewriting Logic: Программная система, руководство, статьи, материалы. — Maude 2.0, 2003, http://maude.cs.uiuc.edu. ↑1, 2.2.3, 5.5, 7

[12] Мешвелиани С. Д.. 2002. *Функциональное программирование и категорный подход в вычислительной алгебре*, Кандидатская диссертация, Институт программных систем РАН, Переславль-Залесский. ↑1

[13] Мешвелиани С. Д. // DoCon, the Algebraic Domain Constructor: Программная система в исходных текстах и Руководство (на английском языке), 2002, http://www.botik.ru/~mechvel/papers.html (выбрать docon). ↑1

S. D. Mechveliani. *On research in automating computation and proofs.* (in russian.)

Abstract. We consider certain simplest heuristical proof method for assertions in algebra and programming based on the equality operations, term rewriting. Inductive reasoning is organized through adding of equalities. The particular features of approach are: failless Knuth-Bendix completion, resource distribution between the proof subgoals. There are considered such examples of assertions as "Forall N M (N+M = M+N)" for the addition algorithm for the natural numbers. It is also introduced a language for equational reasoning/computing. It is considered the question of what should be added to a system of such kind to allow it solving more sensible tasks. The bibliography contains 13 references to freely accessible sources.