УДК 510.23 + 811.93

Д. Ю. Власов

ЯЗЫК SMM — УПРОЩЕННЫЙ METAMATH*

Описывается синтаксис и семантика языка smm — компьютерного языка формальной математики, предназначенного для представления современной математики на формальном уровне и надежной проверки, который является упрощенной версией языка формальной математики metamath.

Ключевые слова: формальная математика, представление математического знания, автоматическая проверка доказательств.

1. Задача надежной автоматической проверки теорем

Когда речь заходит об автоматической верификации доказательств теорем [1; 2], неизбежно возникает вопрос о том, насколько можно доверять подобной проверке. На сегодняшний день, когда говорят об автоматической (механической) верификации теорем, конечно же имеют в виду не физический механизм, проверяющий формальные доказательства, а некую программу, которая проверяет входной файл на некотором языке формальной математики в соответствии с семантикой этого языка. Но программы подвержены ошибкам, и поэтому нет гарантии, что в результате ошибки в программе верификатора некоторое верное доказательство будет отвергнуто, либо, что хуже (поскольку труднее обнаружимо), неверное доказательство будет объявлено верным.

На текущий момент имеются мощные методы верификации программ, позволяющие доказывать разные свойства программ, но, к сожалению, в данном контексте такой подход представляется неприменимым по следующей причине: если мы хотим доказать надежность верификатора при помощи другого верификатора, то почему мы должны верить верификатору верификатора (т. е. кто будет проверять проверяющего)? Поэтому в силу фундаментальности задачи (имеется в виду построение надежного фундамента для математики) разумно применить другой метод. Этот метод сводится к тому, что чем проще язык, на котором пишется формальная математика, и чем проще программа верификатора, тем надежнее она будет верифицировать доказательства и тем больше будет доверия к автоматически проверенным доказательствам. В конечном счете, если программа верификатора написана «в несколько строк», то ошибиться в ней будет очень непросто.

*Работа выполнена в НГУ при финансовой поддержке Совета по грантам Президента РФ для поддержки ведущих научных школ (проект НШ-3669.2010.1), Федерального агентства по образованию, грант ГК-П-1008.

ISSN 1818-7897. Вестник НГУ. Серия: Математика, механика, информатика. 2012. Т. 12, вып. 2. С. 13–25

1.1. *Язык metamath*

Яызк формальной математики metamath [3] был разработан Н. Мегилом в начале 90-х гг. XX в. Этот язык резко выделяется из множества различных систем формальной математики по двум ключевым признакам: *универсальности* и *простоте*, т. е., несмотря на свою универсальность, это очень простой язык. Прямым следствием этой простоты является тот факт, что можно написать сравнительно простой, небольшой по объему, верификатор для этого языка. Это означает, что доказательства, записанные на metamath, могут быть *достаточно надежно* проверены. Существует верификатор metamath, написанный на языке python и состоящий всего из 300 строк[1].

Тем не менее есть и фундаментальный недостаток языка metamath, являющийся прямым следствием его простоты: доказательства в этом языке совершенно не читаемы. Доказательства в metamath представляются как неструктурированная последовательность меток утверждений (фактически это обратная польская запись), и потому без помощи специальных программ не могут быть поняты и отредактированы человеком (кроме совсем простых случаев). Таким образом, несмотря на свои выдающиеся качества, у языка metamath есть серьезный недостаток, делающий его использование очень трудным.

1.2. *Язык Russell*

Язык Russell [4] был разработан автором для преодоления главного недостатка языка metamath — невозможность редактировать и понимать доказательства на этом языке без специализированных программ. Вместе с тем требовалось сохранить степень надежности metamath. Поэтому программной реализацией Russell является транслятор, переводящий исходные файлы в язык smm, который совместим с metamath. Это позволяет использовать удобство чтения и редактирования доказательств на языке Russell и надежность верификации доказательств на языке metamath. Реализация системы формальной математики Russell[2] распространяется по лицензии GPLv3.

1.3. *Язык smm*

Язык smm (аббревиатура от «Simplified MetaMath») представляет собой упрощенный вариант языка metamath. Характерные особенности языка metamath — большая общность (универсальность), позволяющая записывать широкий спектр формальных исчислений, и простота, позволяющая писать простые (следовательно, надежные) верификаторы формальных исчислений, наследуются из metamath в силу того, что фактически smm является подмножеством metamath, с аналогичной семантикой и незначительно ослабленными условиями корректности исходного файла (см. семантику плавающих и существенных гипотез). Более того, поскольку язык smm проще, чем metamath, для него можно писать еще более простые (а следовательно, надежные) верификаторы, чем для metamath.

[1] http://us.metamath.org/downloads/mmverify.py

[2] Пакет mdl на сайте http://www.russellmath.org

Основные отличия smm от metamath заключаются в следующем:

- структура области видимости в smm упрощена до предела: она одноуровневая, т. е. тривиальная (в metamath количество уровней вложенности областей видимости не ограничено);
- в smm каждому блоку соответствует ровно одно утверждение;
- в блоках smm жестко зафиксирована структура компонент утверждения, что позволяет формально описать семантику применения метки из доказательства к стеку в один проход, а не в два, как в metamath;
- метки в smm явно содержат индекс адресуемого объекта, поэтому для реализации smm не обязательно вводить таблицу идентификаторов;
- язык smm порождается LL(0) грамматикой, в то время как metamath — LL(1) грамматикой.

В то же время следует отметить, что smm сместим с metamath: smm-доказательства могут без каких-либо изменений верифицироваться оригинальным верификатором metamath, если в последнем отключить проверку двух условий корректности.

1.4. *Предварительные обозначения*

Через ω будем обозначать множество всех натуральных (неотрицательных) чисел. Отображения из множества $\{0, \ldots, n-1\}$ в множество A будем называть конечными A-значными последовательностями длиной n. Последовательные значения для отображения $f : \{0, \ldots, n-1\} \to A$ обозначаются как $f_0, f_1, f_2 \ldots$ Через $\langle a_0, \ldots, a_{n-1} \rangle$ будем обозначать последовательность f длиной n такую, что $f_i = a_i$ для всех индексов $i < n$. Кортеж длиной n из элементов $a_1, \ldots, a_n$ обозначается стандартным образом: $(a_1, \ldots, a_n)$ и будет использоваться для обозначения последовательности фиксированной длины.

2. Синтаксис языка smm

Все синтаксические конструкции языка smm, кроме текста в комментариях, предполагают стандартную кодировку ASCII. Далее будет описана формальная грамматика языка smm в расширенной форме Бэкуса–Наура.

2.1. *Комментарии*

Синтаксис комментариев в языке smm точно такой же, как в языке metamath. А именно, комментарии в языке smm начинаются с двух символов `(*` и заканчиваются последовательностью из двух символов `*)`. Текст, заключенный в комментариях, никак не влияет на семантику оставшегося кода.

2.2. *Лексемы*

Терминальные символы языка smm (лексемы) образуют множество

$\mathrm{Key}_{\mathrm{smm}} \cup \{\texttt{symbol}, \texttt{index}, \texttt{file}\}$.

Список из 17 ключевых слов $\mathrm{Key}_{\mathrm{smm}}$ (разделены пробелом):

```
$[ $] ${ $} $c $v $f $d $e $a $p $= $. f e a p
```

По сравнению с языком metamath, добавлены 4 новых символа: `f e a p`. Также есть три типа лексем, задаваемых регулярными выражениями.

Символы. Терминальные символы грамматики типа `symbol`. Это атомарные элементы, из которых строятся символьные выражения. Символом является любое слово из диапазона ASCII символов от `!` до `:` (кроме символа `$`). Соответствующее регулярное выражение: `[!-#%-~]+`. Примеры символов: `( ) + * some_func1` (символы разделены пробелами).

Индексы. Терминальные символы грамматики типа `index`. Интерпретируются как индексы утверждений и гипотез, представляют собой десятичные целые числа. Соответствующее регулярное выражение: `[0-9]+`.

Имена файлов. Имя файла `file` — это непустая последовательность символов латинского алфавита, символов из множества $\{\_, -, /, .\}$ и цифр. Соответствующее регулярное выражение: `[a-zA-Z0-9_\-/.]+`.

2.3. *Грамматика*

Формальная грамматика языка smm является LL(0) грамматикой и состоит из 12 продукций:

```
SOURCE      = {CONSTANT | ASSERTION | INCLUSION}
INCLUSION   = "$[" file "$]"
CONSTANT    = "$c" symbol {symbol} "$."
ASSERTION   = "${" {VARIABLE} {DISJOINTED} {ESSENTIAL}
              {FLOATING} (AXIOMATIC | PROVABLE) "$}"
VARIABLE    = "$v" symbol {symbol} "$."
DISJOINTED  = "$d" symbol symbol {symbol} "$."
FLOATING    = "f" index "$f" symbol symbol "$."
ESSENTIAL   = "e" index "$e" symbol {symbol} "$."
AXIOMATIC   = "a" index "$a" symbol {symbol} "$."
PROVABLE    = "p" index "$p" symbol {symbol} "$=" PROOF
PREFIX      = "f" | "e" | "a" | "p"
PROOF       = PREFIX index {PREFIX index} "$."
```

В данной нотации запись `{A}` означает повторение элемента `A` 0, 1 или более раз, `[A]` означает повторение элемента 0 или 1 раз, слова в кавычках обозначают соответствующие терминальные символы грамматики (ключевые слова и зарезервированные символы), слова в нижнем регистре — терминальные символы грамматики, и слова в верхнем регистре — нетерминальные символы грамматики.

Стартовым нетерминалом является нетерминал `SOURCE`. Для совместимости с языком metamath требуется выполнение следующего условия: между любым префиксом из списка `"f"`, `"e"`, `"a"`, `"p"` и последующим токеном (это всегда `index`) не должно быть пробельных символов. Тогда комбинация из префикса и индекса в языке metamath будет корректно интерпретрироваться как единая лексема (метка).

3. Семантика языка smm

Для определения семантики языка smm будет построено отображение ν, выполняющее математическую интерпретацию синтаксических конструкций (синтагм) исходного файла S на языке smm, а также описан процесс верификации доказуемых утверждений, входящих в S.

3.1. *Исходный файл*

Стартовым нетерминалом в грамматике языка smm являестя нетерминал `SOURCE` — исходный файл. РБНФ правило для стартового нетерминала:

```
SOURCE      = {CONSTANT | ASSERTION | INCLUSION}
```

Как видно из определения, исходный файл в языке smm представляет собой последовательность деклараций констант, определения утверждений и включения других файлов. Процесс верификации исходного файла S сводится к последовательной верификации всех утверждений, расположенных в S, после включения в него всех требуемых файлов.

3.2. *Включение файла*

Для разбиения больших файлов на отдельные части можно воспользоваться конструкцией включения файла:

```
INCLUSION   = "$[" file "$]"
```

При обработке исходного файла, после обнаружения в нем такой декларации, в это место вставляется содержимое файла с именем `file`. При этом включение может итерироваться: вложенный файл также может содержать декларацию включения файла.

3.3. *Декларация констант*

Символы, входящие в выражения языка smm, могут быть двух типов: переменные и константы. Переменные — это такие символы, в которые может быть совершена подстановка некоторого выражения. Константы — это символы, в которые нельзя производить никакие подстановки. Все константы, использующиеся в выражениях, должны быть предварительно задекларированны при помощи синтаксической конструкции декларации констант:

```
CONSTANT = "$c" symbol {symbol} "$."
```

Внутри декларации символы констант разделены пробелами. Пример декларации констант: `$c ( ) + * some_func $.`

3.4. *Декларация переменных*

Декларация переменных в языке smm отделена от определения их типов, и в блоке декларации просто перечисляются символы, которые считаются переменными.

```
VARIABLE = "$v" symbol {symbol} "$."
```

Внутри декларации символы переменных разделены пробелами. Пример декларации переменных: `$v x y z ph ps gamma $.` Для корректности определения этого множества требуется, чтобы любой символ из декларации переменных не был ранее задекларирован как константа.

3.5. *Декларация дизъюнктных ограничений*

Эта синтаксическая конструкция задает перечисление переменных (которые далее считаются дизъюнктными), разделенных пробелами:

```
DISJOINTED = "$d" symbol symbol {symbol} "$."
```

Внутри декларации должно быть не менее двух символов переменных, разделеных пробелами. В языке smm нет механизма для вывода множеств дизъюнктных ограничений в доказуемых утверждениях, поэтому множества дизъюнктных переменных всегда должны быть явно задекларированны и в аксиоматических утверждениях, и в доказуемых. Пример декларации дизъюнктных переменных: `$d x y z $.`

Декларацию дизъюнктных переменных d будем интерпретировать как множество соответствующих символов: $\nu(d) = \{v_0, \ldots, v_{k-1}\}$. Для корректности определения дизъюнктных переменных требуется, чтобы любой символ из множества $\nu(d)$ был ранее задекларирован как переменная.

3.6. *Плавающая гипотеза*

Плавающие гипотезы — это один из видов предпосылок (гипотез) утверждений smm, которые по существу представляют собой гипотезы о типах переменных. Плавающая гипотеза

```
FLOATING = "f" index "$f" symbol symbol "$."
```

фактически означает то, что второе вхождение символа `symbol` — это переменная, а первое вхождение символа `symbol` — это символ типа этой переменной. `index` — это индекс данной плавающей гипотезы, должен соответствовать реальному индексу данной плавающей гипотезы в утверждении.

Сочетание токенов `"f" index` (пробельных символов между ними нет) будет в metamath интерпретироваться как метка. Поскольку в metamath *все* метки (в том числе и плавающих гипотез) должны быть уникальными глобально, чего нельзя гарантировать в smm, для использования верификатора языка metamath нужно в программном коде

отключить проверку данного условия, иначе при работе с оттранслированным из Russell файлом верификатор выдаст огромное количество предупреждений о дублированных метках гипотез.

Примеры плавающих гипотез: `f3 $f wff ph $.` — переменная `ph` имеет тип `wff` (well formed formula), `f0 $f set x $.` — переменная `x` имеет тип `set` (множество).

Интерпретацией плавающей гипотезы f будем считать пару $\nu(f) = (l, \langle t, v\rangle)$, где l — это индекс f, v — символ переменной, t — символ типа переменной v. Для корректности определения плавающей гипотезы требуется, чтобы символ t ранее в исходном файле был декларирован как константа, а символ v — задекларирован как переменная в текущем утверждении.

3.7. *Существенная гипотеза*

Существенные гипотезы — это содержательные гипотезы, соответствующие предпосылкам (гипотезам) утверждений.

```
ESSENTIAL = "e" index "$e" symbol {symbol} "$."
```

Здесь `symbol` — последовательность символов выражения гипотезы, `index` — индекс существенной гипотезы, который должен соответствовать реальному индексу данной существенной гипотезы в утверждении.

Кроме условия глобальной уникальности меток гипотез, в metamath есть еще одно ограничение: существенная гипотеза не должна начинаться с переменной. В smm нет таких ограничений, более того, оттранслированный из Russell файл на языке smm, как правило, содержит существенные гипотезы, начинающиеся с переменной. Для того, чтобы избежать многочисленных предупреждений при верификации файла smm, в коде верификатора metamath вместе с проверкой глобальной уникальности меток гипотез нужно также отключить данные предупреждения. Примеры существенных гипотез: `e1 $e ( ph -> ps ) $. e2 $e ph $.`

Интерпретацией существенной гипотезы e будм считать пару $\nu(e) = (l, x)$, где l — это индекс e, x — последовательность символов выражения гипотезы e. Для корректности определения существенной гипотезы требуется, чтобы любой символ s из последовательности x либо ранее в исходном файле был декларирован как константа, либо задекларирован как переменная в текущем утверждении.

3.8. *Аксиоматическое заключение*

Аксиоматическое заключение в языке smm может использоваться в доказательствах, но само не требует доказательства.

```
AXIOMATIC = "a" index "$a" symbol {symbol} "$."
```

Здесь `symbol` — это выражение аксиоматического заключения, `index` — индекс заключения утверждения. Нумерация заключений (аксиоматических и доказуемых) сквозная для всего исходного файла, и индекс любого заключения должен соответствовать его порядковому номеру в исходном файле. Это гарантирует глобальную уникальность соответствующих меток утверждений в metamath. Пример: `a5 $a ( ph -> ( ps -> ph ))$.`

Интерпретацией аксиоматического заключения a будем считать пару $\nu(a) = (l, x)$, где l — индекс a, а x — последовательность символов выражения a.

3.9. *Доказуемое заключение*

В отличие от аксиоматического доказуемое заключение имеет доказательство.

```
PROVABLE = "p" index "$p" symbol {symbol} "$=" PROOF
```

Здесь `PROOF` — доказательство заключения, выражение которого составляет последовательность символов после `$p`.

Пример доказуемого заключения: `p8 $p ( ps -> ph ) $= <proof>`. Интерпретацией доказуемого заключения t будем считать пару $\nu(t) = (l, x)$, где l — индекс t, а x — последовательность символов выражения t. Доказательство не входит в интерпретацию t с целью унификации формы, общей для обоих типов заключений: аксиоматического и доказуемого.

3.10. *Префикс*

Префикс индекса указывает на тип индексируемого объекта.

```
PREFIX      = "f" | "e" | "a" | "p"
```

Здесь `"f"` обозначает плавающую гипотезу, `"e"` — существенную гипотезу, `"a"` — аксиоматическое утверждение, `"p"` — доказуемое утверждение. Интерпретацией префиксов будем считать элементы зафиксированного множества $\mathrm{Px} \rightleftharpoons \{\mathtt{e}, \mathtt{f}, \mathtt{a}, \mathtt{p}\}$.

3.11. *Доказательство*

В отличие от языка Russell синтаксическое определение доказательства в smm достаточно просто. Это непустая последовательность глобальных индексов с префиксами для утверждений и гипотез того утверждения, заключение которого доказывается.

```
PROOF = PREFIX index {PREFIX index} "$."
```

Семантика проверки доказательства smm по сути идентична семантике верификации доказуемых утверждений Russell, но гораздо более подробна, поскольку доказательство в smm содержит полную информацию о выводе всех выражений по правилам грамматики выражений, а также полное описание подстановок логических переходов.

Для определения процесса верификации доказательств необходимо понятие стека выражений. Стек выражений — это конечная последовательность выражений, с которой можно производить две операции: pop (извлечение верхушки стека) и push (помещение в верхушку стека). Тогда доказательство smm будет интерпретироваться как RPN программа вычисления символьного выражения на стековом «калькуляторе», оперирующем символическими выражениями на стеке. Соответственно процесс верификации доказательства некоторого утверждения t состоит в вычислении выражения согласно доказательству t, затем на стеке должно остаться ровно одно выражение, и оно должно посимвольно совпадать с выражением заключения утверждения t. Если

все эти условия выполнены, то t считается верифицированным. Более подробно семантика доказательства в языке smm будет изложена позже. Пример доказательства: `e0 f0 f1 a2 f0 f1 f0 a1 a5 $.` Интерпретацией доказательства p будем считать последовательность пар $\boldsymbol{\nu}(p) = \langle (p_0, l_0), \ldots, (p_N, l_N) \rangle$, где p_i — это соответствующие префиксы, l_i — индексы, строго меньшие индекса доказуемого заключения.

3.12. *Утверждение*

Утверждение — это основная синтаксическая единица в языке smm. При трансляции из Russell в smm в утверждения переходят практически все основные синтаксические структуры языка Russell (кроме декларации констант): правила, аксиомы, определения, теоремы, отношения супертип-подтип между типами. Правило РБНФ для утверждений в языке smm:

```
ASSERTION = "${" {VARIABLE} {DISJOINTED} {ESSENTIAL}
            {FLOATING} (AXIOMATIC | PROVABLE) "$}"
```

Здесь `VARIABLE` — декларации всех переменных, входящих в выражения утверждения, `DISJOINTED` — декларации всех дизъюнктных ограничений в данном утверждении, `FLOATING` — список плавающих гипотез утверждения, `ESSENTIAL` — список существенных гипотез утверждения, `AXIOMATIC` или `PROVABLE` — заключение данного утверждения. Границы утверждения задаются токенами `$\{` — начало, и `$}` — конец.

Для описания семантики утверждения a определим математическую интерпретацию a как четверку $\boldsymbol{\nu}(a) = (D, F, E, p)$ где D — множество дизъюнктных ограничений, $F = \langle f_0, \ldots, f_{n-1} \rangle$ — последовательность интерпретаций плавающих гипотез (гипотез о типах), $E = \langle e_0, \ldots, e_{m-1} \rangle$ — последовательность интерпретаций существенных гипотез, p — интерпретация заключения утверждения a. Далее будет приведен ряд примеров утверждений.

3.12.1. *Пример правила грамматики*

Следующие два аксиоматических утверждения фактически представляют собой то, во что превращаются правила грамматики выражений языка Russell при трансляции. Отрицание (`-.`) формулы — это формула

```
${
   $v ph $.
   f0 $f wff ph $.
   a0 $a wff -. ph $.
$}
```

Импликация (`->`) двух формул — это формула

```
${
   $v ph ps $.
   f0 $f wff ph $.
   f1 $f wff ps $.
```

```
   a1 $a wff ( ph -> ps ) $.
$}
```

3.12.2. *Пример аксиомы*

Следующие два аксиоматических утверждения — это две аксиомы пропозициональной логики. В подобные утверждения транслируются соответствующие аксиомы из языка Russell. Аксиома без посылок (т. е. аксиома в обычном понимании):

```
${
   $v ph ps $.
   f0 $f wff ph $.
   f1 $f wff ps $.
   a2 $a ( ph -> ( ps -> ph ) ) $.
$}
```

Аксиома с двумя посылками (правило вывода modus ponens):

```
${
   $v ph ps $.
   e0 $e ph $.
   e1 $e ( ph -> ps ) $.
   f0 $f wff ph $.
   f1 $f wff ps $.
   a5 $a ps $.
$}
```

3.12.3. *Пример определения*

В языке smm нет специальной конструкции для определений, вместо этого определения даются через аксиомы. Следующее аксиоматическое утверждение получается из определений в языке Russell при трансляции в smm.

Определение логической связки \/ (или) через импликацию:

```
${
   $v ph ps $.
   f0 $f wff ph $.
   f1 $f wff ps $.
   a225 $a ( ( ph \/ ps ) <-> ( -. ph -> ps ) ) $.
$}
```

3.12.4. *Пример теоремы*

Теорема syl (правило силлогизма) — транслированная из аналогичной теоремы на языке Russell:

```
${
   $v ph ps ch $.
   e0 $e ( ph -> ps ) $.
   e1 $e ( ps -> ch ) $.
   f0 $f wff ph $.
   f1 $f wff ps $.
   f2 $f wff ch $.
   a8 $p ( ph -> ch ) $=
      e0 e1 f1 f2 a1 f0 a6 f0 f1 f2 a7 f0 f1 a1 f0 f2 a1 a5 $.
$}
```

3.12.5. *Пример отношения подтип-супертип*

Типичным примером отношения между типами переменных можно считать отношение подтип-супертип (похожего на наследование в ООП) типа `set` (множество) от типа `class` (класс). При трансляции из Russell в smm данная пара подтип-супертип транслируется в следующее утверждение:

```
${
   $v x $.
   h0 $f set x $.
   a18 $a class x $.
$}
```

3.13. *Семантика верификации доказательства*

Зафиксируем некоторое утверждение t, для которого будет описываться семантика верификации. Для определения семантики доказательств t сначала определим множество выражений, допустимых в контексте данного утверждения. А именно, для этого достаточно определить два множества: C_t — множество констант, задекларированных до определения t, и V_t — множество переменных, задекларированных в утверждении t. Определим множество $E_t \rightleftharpoons \{\langle s_0, \ldots, s_{k-1}\rangle \mid s_i \in C_t \cup V_t,\ k \in \omega\}$ — это все выражения, допустимые в доказательстве t. Определим также множество $L_t = M \times \mathrm{Px}$, где M — это индекс утверждения t в исходном файле.

3.13.1. *Стек выражений*

Определим множество последовательностей допустимых выражений:
$S_t \rightleftharpoons \{\, \langle e_0, \ldots, e_{k-1}\rangle \mid e_i \in E_t,\ k \in \omega \,\}$.

Стек выражений (более точно: LIFO стек выражений) определяется двумя операциями:

- $push : S_t \times E_t \to S_t$ — операция размещения в стеке, определяется следующим образом: $push(\langle e_0, \ldots, e_{k-1}\rangle, e) = \langle e_0, \ldots, e_{k-1}, e\rangle$;
- $pop : S_t \to S_t \times E_t$ — операция выталкивания из стека, определяется следующим образом: $pop\,(\langle e_0, \ldots, e_{k-1}\rangle) = (\langle e_0, \ldots, e_{k-2}\rangle, e_{k-1})$.

3.13.2. *Доказательство как RPN-программа вычисления выражения*

Пусть интерпретация доказательства утверждения t — это последовательность $\pi = \langle (p_0, l_0), \ldots, (p_N, l_N) \rangle$, где p_i — префиксы, и l_i — индексы. Отметим, что $\alpha_i = (p_i, l_i) \in L_t$ для всех $i \leq N$.

π можно интерпретировать как программу для вычисления заключения утверждения t в обратной польской записи (RPN-программа). В соответствии с этим программа вычисления заключения исполняется последовательно, по шагам доказательства, совершая операции на стеке, который изначально (при начале доказательства) является пустым.

Для описания семантики работы стековой машины, требуется описать функцию перехода: $v : S_t \times L_t \to S_t$. Функция $v(s, \alpha)$, где $s \in S_t$ и $\alpha \in L_t$, определяется по-разному в зависимости от первой компоненты пары: $\alpha = (p, l)$.

3.13.3. *Случай индекса гипотезы*

Рассмотрим случай, когда $p \in \{\mathtt{e}, \mathtt{f}\}$. Тогда l должно быть индексом некоторой гипотезы $h = (l, x)$ типа p в утверждении t. Тогда положим $v(s, \alpha) \rightleftharpoons push(s, x)$, т. е. в данном случае функция перехода добавляет выражение гипотезы h на стек и не зависит от типа гипотезы.

3.13.4. *Случай индекса утверждения*

Теперь рассмотрим случай, когда когда $p \in \{\mathtt{a}, \mathtt{p}\}$, т. е. l является индексом некоторого предыдущего утверждения a. Распишем интерпретацию a. Пусть это четверка (D, F, E, p). Тогда вычисление $v(s, \alpha)$ распадается на три этапа.

Этап I. Пусть $F = \langle f_0, \ldots, f_{n-1} \rangle$. Тогда определим последовательность S_F пар выражений и стеков (s_i, u_i), $i < n$, получаемую n-кратным применением операции *pop* к s: $(s_{n-1}, u_{n-1}) = pop(s)$ и $(s_{i-1}, u_{i-1}) \rightleftharpoons pop(s_i)$ (внимание: индексы в обратном порядке!). При этом для каждого $i < n$ проверяется условие, что первый символ выражения u_i и выражения x_i, где $f_i = (l_i, x_i)$, совпадают. Если это условие не выполняется, то верификация заканчивается с отрицательным результатом. Последовательности F и S_F однозначно определяют некоторую подстановку θ следующим образом. Область определения θ — это множество переменных, участвующих в плавающих гипотезах из F: $dom(\theta) = \{v_i | f_i = (l_i, \langle t_i, v_i \rangle), i < n\}$. При этом значение θ на переменных определяется так: $\theta(v_i) = \langle s_1, \ldots, s_{k_i} \rangle$, где символы s_i определяются по выражению $u_i = \langle s_0, \ldots, s_{k_i} \rangle$. Иным значением θ на переменной v_i является выражение u_i без первого символа (первый символ имеет значение типа выражения, поэтому убирается). В результате этого этапа получаются стек s' и подстановка θ.

Этап II. Пусть $E = \langle e_0, \ldots, e_{m-1} \rangle$. Тогда определим последовательность S_E пар выражений и стеков (t_j, w_j), $j < m$, получаемую m-кратным применением операции *pop* к s': $(t_{m-1}, w_{m-1}) = pop(s')$ и $(t_{i-1}, w_{i-1}) \rightleftharpoons pop(t_i)$ (снова индексы в обратном порядке). При этом для каждого $j < m$ проверяется условие, что $w_j = \theta(e_j)$. Если это условие нарушено, то верификация заканчивается с отрицательным результатом. После выполнения этого этапа мы получаем стек s''.

Этап III. После прохождения предыдущих двух этапов у нас есть стек s'' и подстановка θ. На этом этапе требуется проверка еще двух условий, связанных с дизъюнктными ограничениями. Во-первых, для любого множества дизъюнктных переменных $d \in D$ и любой пары различных переменных $x, y \in d$ требуется, чтобы множества переменных, входящих в выражения $\theta(x)$ и $\theta(y)$ были дизъюнктными, т. е. не пересекались. Во-вторых, для каждой такой пары $x, y \in d$ требуется, чтобы каждая пара переменных $u \in \theta(x)$ и $w \in \theta(y)$ входила бы в некоторое множество дизъюнктных переменных для доказуемого утверждения t (это условие наследственности дизъюнктных ограничений). После проверки этих условий можно считать, что данный шаг доказательства верифицирован. Тогда положим $v(s, \alpha) \rightleftharpoons push(s'', \theta(p))$.

3.13.5. *Терминальная проверка заключения*

После того как пройдены все шаги доказательства, т. е. построена последовательность стеков $\langle s_0, \ldots, s_N \rangle$, где s_0 — пустой стек, и $s_{i+1} = v(s_i, \alpha_{i+1})$ для всех $0 < N-1$, получим итоговый стек s_N. Он должен состоять из единственного выражения, и заключение доказываемого утверждения p должно совпадать с этим выражением. Фактически эти условия можно записать таким образом: $pop(S_N) = (\emptyset, p)$.

Список литературы

1. *Boyer R. et al.* The QED Manifesto // Automated Deduction — CADE 12 / Ed. by A. Bundy. Springer-Verlag, 1994. Vol. 814 of LNAI. P. 238–251. URL: `http://www.cs.ru.nl/~freek/qed/qed.ps.gz`.

2. *Barendregt H., Wiedijk F.* The Challenge of Computer Mathematics // Transactions A of the Royal Society. 2005. Vol. 363. No. 1835. P. 2351–2375.

3. *Megill N. D.* Metamath: A Computer Language for Pure Mathematics. Morrisville: Lulu press, 1997. URL: `http://us.metamath.org/downloads/metamath.pdf`.

4. *Власов Д. Ю.* Язык формальной математики Russell // Вестн. Новосиб. гос. ун-та. Серия: Математика, механика, информатика. 2011. Т. 11, № 2. С. 27–50.

Материал поступил в редколлегию 26.03.2010

Адрес автора

ВЛАСОВ Дмитрий Юрьевич

Новосибирский государственный университет

ул. Пирогова, 2, Новосибирск, 630090, Россия

Институт математики им. С. Л. Соболева СО РАН

пр. Акад. Коптюга, 4, Новосибирск, 630090, Россия

e-mail: vlasov@academ.org