УДК 519.688

О КОМПИЛЯЦИИ ВЫРАЖЕНИЙ В ЯЗЫКЕ MATHPAR

© **И.А. Борисов**

Ключевые слова: синтаксический анализ; предметно-ориентированный язык программирования; язык Mathpar.

Мы приводим формальное описание конструкций языка Mathpar, которые используются для записи выражений в системе компьютерной математики Math Partner, а также алгоритмов, которые используются для компиляции их во внутреннее представление.

1 Введение

Одной из важных задач при построении компьютерной математической системы является выбор структур данных для хранения математических объектов и определение языка пользователя.

В статье приводится формальное описание конструкций языка Mathpar, которые используются для записи выражений в системе компьютерной математики Math Partner [1–4], а также алгоритмов, которые используются для компиляции этих выражений во внутреннее представление.

2 Структура для хранения функций

Все выражения языка Mathpar изначально представляются в виде функций.

Для хранения композиций функций в Mathpar определен программный класс F. Объект этого класса включает в себя целочисленный идентификатор, указывающий имя функции, и список аргументов, причем аргументами могут быть как другие функции, так и числа, полиномы, векторы, матрицы и т. д. Функцию можно представить в виде дерева. Пример дерева функции представлен на рис. 1.

3 Язык Mathpar

В Mathpar для ввода задач пользователем используется язык, приближенный к языку разметки LaTeX. Ниже приведена основная часть грамматики языка, записанная в нотации Бэкуса-Наура [5].

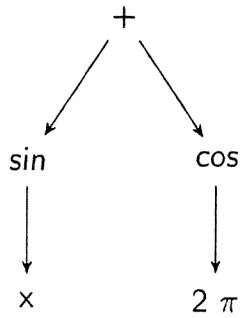

Рис. 1: Пример дерева функции $\sin(x) + \cos(2\pi)$

```
PROGRAM = EXPR+ ;
EXPR = [NAME "="] RHS ";" ;
NAME = ALPHABETIC (ALPHABETIC | DIGIT | UNDERSCORE | "{" | "}")* ;
RHS = FUNCTION | MATRIX | VECTOR | POLYNOMIAL | NUMBER ;
FUNCTION = "\" FUNCTION_NAME "(" [ARGUMENTS] ")" ;
FUNCTION_NAME = NAME;
ARGUMENTS = (ARGUMENT ",")* ARGUMENT ;
ARGUMENT = RHS;
MATRIX = "[" (VECTOR ",")* VECTOR "]" ;
VECTOR = "[" ((FUNCTION | POLYNOMIAL | NUMBER | NAME) ",")*
    (FUNCTION | POLYNOMIAL | NUMBER | NAME) "]";
POLYNOMIAL = POLTERM [("+" | "-") TERM] ;
POLTERM = NUMBER ["*"] VAR INTPOWER;
VAR = NAME;
NUMBER = INTEGER | REAL | COMPLEX;
COMPLEX =
      [(INTEGER | REAL) ("+" | "-")] (INTEGER | REAL) ["*"] "i"
    | (INTEGER | REAL) ["*"] "i" ;
INTPOWER = ("^" INTEGER) | ("^" "{" INTEGER "}") ;
INTEGER = ["-"] DIGIT+ ;
REAL = INTEGER "." DIGIT+ ;
ALPHABETIC = "A"| ... | "Z" | "a" | ... | "z" ;
DIIGT = "0" | ... | "9" ;
UNDERSCORE = "_";
```

Описываемый метод разбора применяется только к правой части выражений (RHS).

3.1 Примеры выражений на языке Mathpar

```
a = 10;
b = 2 * 2;
c = 5! 3;
d = \sin(x^2);
e = [1, 2, 3];
```

```
f = \int_{1}^{2} \sin(x) dx;
g = 'hello, world';
```

3.2 Алгоритм разбора функций, записанных на языке Mathpar

1. Выполнить лексический анализ.
2. Дополнительно подготовить список лексем для синтаксического анализа.
3. Провести синтаксический анализ и создать объект типа `F`.

4 Лексический анализ

Лексический анализатор работает со строкой, содержащей выражение на языке Mathpar. Результатом его работы является список *токенов* (или *лексем*) — отдельных значащих подстрок.

Лексема определяется ее строковым представлением и типом. Некоторые токены, такие как числа и полиномы, дополнительно имеют связанные с их значениями данные.

Разбор входной строки осуществляется с помощью списка регулярных выражений [6], соответствующих различным типам токенов по следующему алгоритму [7–8].

Пока не достигнут конец строки, выполняется поиск по заданным регулярным выражениям. На основе подстроки с совпадающей последовательностью символов создается токен соответствующего типа. Поиск следующего токена продолжается с конца предыдущего. Если не было найдено совпадений по заданным регулярным выражениям, начало поиска сдвигается вперед на один символ до тех пор, пока снова не встретится подстрока с предопределенным выражением. Нераспознанная подстрока, находящаяся между двумя соседними распознанными подстроками, заносится в токен специального типа `SYMBOLIC` для дальнейшего анализа интерпретатором. Токен `SYMBOLIC` создается при обнаружении строкового литерала — любой последовательности символов, окруженных апострофами.

Постоянные регулярные выражения:

1)числа (как целые, так и вещественные);

2)ключевые слова, встроенные константы (например, греческие буквы, знак бесконечности и т. п.);

3) имена функций;

4) инфиксные операции;

5) постфиксные операции;

6) разделители аргументов функций;

7) скобки;

8) пробельные символы.

Зависящие от контекста регулярные выражения:

1) переменные полиномов;

2) функции, определенные пользователем.

Перед началом лексического анализа выполняются следующие подготовительные шаги:

1) инициализация счетчиков, определяющих текущую позицию внутри исходной строки;

2) создание и компиляция регулярных выражений, зависящих от контекста, а также вставка их в общий список регулярных выражений.

5 Синтаксический анализ

В основе синтаксического анализатора лежит обратная польская нотация (ОПН). ОПН — форма записи математических выражений, в которой операнды расположены перед знаками операций.

Например, выражение $(3+5)\times 2$ в ОПН записывается как $2\ 3\ 5+\times$ или $3\ 5+2\times$.

Синтаксический анализ происходит в 3 этапа:

1) подготовительный этап. Происходит изменение списка токенов для создания ОПН;

2) преобразование списка токенов в ОПН;

3) создание объекта типа `F` из списка токенов в ОПН.

На подготовительном этапе список токенов, полученный после лексического анализа, перестраивается таким образом, чтобы все выражения были приведены к стандартной префиксной форме записи функций. Примеры выражений, нуждающихся в предварительной обработке:

- пределы. $\lim\limits_{x\to 0}\frac{1}{x}$: `\lim_{x \to 0} 1/x` → `\lim(1/x, x, 0);`
- интегралы. $\int \sin(x)dx$: `\int \sin(x) dx` → `\int(\sin(x), x);`

  $\int\limits_a^b \sin(x)dx$: `\int_{a}^{b} sin(x) dx` → `\int(\sin(x), x, a, b).`
- суммы. $\sum\limits_{i=1}^{n}\sin(\pi n)$: `\sum_{i = 1}^{n} \sin(\pi n)` →

  `\sum(\sin(\pi n), i, 1, n);`
- логарифмы. $\log_a b$: `\log_{a} b` → `\log(a, b);`
- векторы. $[1,2,3]$: `[1, 2, 3]` → `\vector(1, 2, 3);`

Для преобразования из инфиксной нотации в ОПН используется алгоритм «сортировочной станции», предложенный Э. Дейкстрой.

Входными данными для алгоритма является список токенов, полученный после подготовительного этапа; выходными — список токенов, представляющий ОПН входной цепочки токенов. Для хранения промежуточных данных используется стек.

В ходе преобразования в ОПН осуществляется проход по всему списку токенов, и в зависимости от типа текущего токена выполняются следующие действия:

- если токен является полиномом, числом или строкой (тип SYMBOLIC), он сразу добавляется в выходной список токенов;
- функция или открывающая скобка помещается в стек;

- разделитель аргументов функции — элементы из стека выталкиваются в выходной список, пока не встретится открывающая скобка;

- арифметическая операция или функция — элементы из стека выталкиваются в выходной список, пока приоритет текущей операции меньше или равен приоритету операции, находящейся на вершине стека;

- закрывающая скобка — элементы из стека выталкиваются в выходной список, пока не встретится открывающая скобка. Открывающая скобка также удаляется из стека, но не помещается в выходной список. Если открывающая скобка, соответствующая текущей закрывающей, стояла сразу после токена функции, после удаления открывающей скобки на вершине стека останется токен функции, который нужно вытолкнуть из стека в выходной список.

После завершения прохода по списку токенов все элементы, оставшиеся в стеке, выталкиваются в выходной список.

Завершающий этап синтаксического анализа — конструирование объектов типа F из списка токенов в ОПН. Для этого выполняется проход по списку токенов в виде ОПН, и в зависимости от типа текущего токена выполняются следующие действия:

- строка (SYMBOLIC), полином или число — создается объект F, содержащий соответствующее значение. Он является листовой вершиной дерева;

- функция или арифметическая операция — по строковому имени функции или операции определяется целочисленная константа, соответствующая имени. Определяется количество аргументов. Из стека выталкиваются аргументы данной функции и создается объект F с этими аргументами.

Все создаваемые объекты помещаются во временный стек. После завершения компиляции выражения построенная функция, являющаяся вершиной дерева, находится на вершине стека.

6 Заключение

Была приведена формальная грамматика языка Mathpar и описан подход для разбора выражений на этом языке с применением регулярных выражений и обратной польской нотации.

ЛИТЕРАТУРА

1. *Malaschonok G.I.* Tropical computations in Mathpar. Tropical and Idempotent Mathematics // G.L. Litvinov, V.P. Maslov, A.G. Kushner, S.N.Sergeev (Eds.) Institute for Information Transmission Problems of RAS. Moscow, 2012. P. 143-148.

2. *Malaschonok G.I.* Project of Parallel Computer Algebra // Tambov University Reports. Series: Natural and Technical Sciences. Tambov, 2010. V. 15. Issue 6. P. 1724-1729.

3. *Малашонок Г.И.* Компьютерная математика для вычислительной сети // Вестник Тамбовского университета. Серия Естественные и технические науки. Тамбов, 2010. Т. 15. Вып. 1. С. 322-327.

4. *Малашонок Г.И.* О проекте параллельной компьютерной алгебры // Вестник Тамбовского университета. Серия Естественные и технические науки. Тамбов, 2009. Т. 14. Вып. 4. С. 744-748.

5. *Фридл Дж.* Регулярные выражения. СПб.: Символ-Плюс, 2008.
6. *Вирт Н.* Построение компиляторов / пер. с англ. Е.В. Борисов, Л.Н. Чернышов. М.: ДМК, 2010.
7. *Шилдт Г.* Java: методики программирования Шилдта. М.: Вильямс, 2008.
8. *Шилдт Г.* Искусство программирования на Java. М.: Вильямс, 2005.

БЛАГОДАРНОСТИ: Работа выполнена при поддержке гранта РФФИ № 12-07-00755-а.

Поступила в редакцию 20 декабря 2012 г.

Borisov I.A. ABOUT COMPILATION OF EXPRESSIONS IN MATHPAR LANGUAGE.

We describe syntax of Mathpar language and propose method to parse expressions in this language.

Key words: syntax analysis, domain-specific languages, computer mathematical system Mathpar.