УДК 519.685

# К ВОПРОСУ ОБ ИСПОЛЬЗОВАНИИ ПОНЯТИЯ СОПРЯЖЕННОГО ПО ОТНОШЕНИЮ К БЕССКОБОЧНЫМ ВЫРАЖЕНИЯМ

*Теодор Заркуа*

Грузинский университет им. Святого Андрея Первозванного Патриаршества Грузии, Грузия

**Аннотация**

*Данная работа повящена вопросам программной реализации расширения алгоритмических языков путем привнесения в них средств для автоматизации обработки функциональных выражений, заданных в естественном виде. Указанные средства существенно используют свойства бесскобочных (т.н. польских) записей функциональных (алгебраических) выражений, основывающихся на понятии сопряженного, исследованных в цикле работ автора, посященных этому вопросу.*

*Предлагается внутренний язык для описания функциональных преобразований олгебраических выражений, а также структура данных для представления этих выражений, позволяющие, с одной стороны, существенно облегчить программную реализацию, а с другой стороны, сделать ее легко модифицируемой. Конкретные вопросы раскрываются применительно к языку C++, однако подход имеет общий характер и его основные положения легко могут быть перенесены на другие обьектно-ориентированные языки программирования.*

Общеизвестно, что для атоматизации обработки алгебраических выражений удобно использовать бесскобочные, т.н. польские, записи, названные так в честь их автора, польского математика Яна Лукасевича. Причем для вычисления значений удобнее использовать постфиксную форму, а для функциональных преобразований – префиксную [1]. В работах [2]-[5] были исследованы свойства бесскобочных выражений, основанные на понятии сопряженного, введенного автором. Основополагающий результат, полученный в этих работах кратко можно сформулировать следующим образом – имея префиксную форму алгебраического выражения нет необходимости преобразовывать ее к постфиксному виду, т.к. для вычисления значения выражения достаточно обработать префиксную форму справа налево алгоритмом, идентичным по своей сложности с классическим алгоритмом, применяемым для этой цели к постфиксной форме.

Совершенно очевидно, что этот результат позволяет акцентироваться на получении префиксной формы алгебраического выражения и реализации его функциональных преобразований, что весьма актуально для задач, при решении которых возникает необходимость в обработке фуникциональных выражений, заданных в естественном виде посредством строковых данных Так как в работе [5] довольно подробно изучен вопрос о том как удобнее получить префиксную форму алгебрического выражения, то остается решить вопросы, непосредственно связанные с реализацией функциональных преобразований на базе префиксной записи выражения.

Вначале заметим, что префиксное выражение можно представить в виде последовательнсти, каждый элемент которой является либо опреандом, либо знаком операции. Исходя из этого, тип элемента этой последовательности может быть описан так:

```
struct Telem
{char operand;
union {char nomer;
double value;
}
};
```

При этом, можем считать, что если operand равен 0, то это операция и нас интересует ее номер (nomer), а если operand не равен 0, то это операнд. Причем, если значение поля operand равно 1, то это переменная и в дальнейшем нас может интересовать ее номер, т.е. поле nomer (если переменных несколько) и значение (которое будет задаваться извне), а если operand равен 2, то это константа и тогда для нас представляет интерес поле value, содержащее соответствующее значение. Вполне естественно последовательность таких элементов представить в виде вектора.

А теперь зададимся вопросом – какие операции будут нам нужны по отношению к этому вектору, если мы хотим уметь выполнять, напрмер, дифференцирование выражения. Легко заметить, что нам будет необходимо уметь выполнять следующие действия

– выделение подвектора, представляющего собой первый операнд операции, выделение подвектора, представляющего собой второй операнд операции, конкатенация вектора с вектором и вектора с элементом, вставка в вектор вектора и вставка в вектор элемента. Таким образом, напрашивается решение сделать класс, соответствующий префиксному выражению потомком стандартного класса вектора и добавить туда перечисленные возможности.

Дальнейший шаг это разработка внутреннего языка для описания функциональных преобразований. Здесь предлагается язык, в котором преобразование описывается строкой, каждый элемент которой занимает одну позицию. В частности, константы представляют сами себя (при этом e обозначает число Непера), операция, для наглядности, обозначается соответствующим знаком, элементарная функция – соответствующей малой буквой латинского алфавита, начиная с буквы 'f', первый и второй операнды выражения – соответственно буквами 'a'и 'b' , производные этих операндов – соответственно, буквами 'A' и 'B'. В этих терминах, к примеру, некоторые формулы дифференцирования записываются следующим образом:

$$`x1``^{x}n**b^{a}-b1A``^{a}x**^{a}bqaA`+uv+AB`-uv-AB`*uv+*Ab*aB`/uv/-*Ab*aB*bb$$

$$`uv*AbB`Lnu*/1aA`Lgu*/reaA`Sinu*gaA`Cosu*-0faA$$

Для того, чтобы понять приведенное, необходимо учесть, что выражения приведены в префиксном виде, и что Sin обозначается буквой f, Cos - буквой g, Ln – буквой q, Lg - буквой r (полный список обозначений элементарных функций вряд ли представлzет интерес – ведь речь идет о внутреннем языке и суть важно знать его принципы). В частности, запись */reaA есть произведение, в котором первый операнд (/rea) является результатом деления Lge на первый операнд исходного выражения, а второй операнд является производной первого операнда исходного выражения.

Теперь для реализации остается записать в отдельный вектор строк все необходимые формулы на внутреннем языке и иметь метод, который обрабатывая префиксную запись, данную ему на входе просто применяет саоответствующие формулы. В таком случае, очевидно, добавление новых преобразований требует просто расширения вектора формул.

Реализация уже может реально работать, но в окончательном варианте следует предусмотреть сокращение полученных выражений за счет появления константы 0 в качестве сомножителя или слагаемого, константы 1 в качестве сомножителя или делителя, а также за счет сокращений.

Очевидно предлагаемый подход легко переносится на любые алгоритмические языки программирования, поддерживающие объектно-ориентированную парадигму.

**Список использованных источников:**


1. Грис Д. Конструирование компиляторов для цифровых вычислительных машин. - Москва, "Мир 1975.
2. Заркуа Т. К свойству перевернутой польской записи. Internet-Edication-Sciense-2008. New Informational and Computer Tecnologies in Education and Sciense. H. Intelligence Information Systems. 2008 year, p.543-544
3. Заркуа Т. Некоторые способы обработки функциональных выражений. Winter Programming School. Материалы зимней школы по программированию, Харьков, ХНУРЭ, 2009, p. 205-211
4. Zarkua T. An Approach to Functional Expressions Processing Automation. Billetin №2 of Saint Andrew the First-Called Georgian University of the Patriarchy of Georgia, Tbilisi, 2009, p. 50-61
5. Т.Заркуа.Автоматизация обработки функциональных выражений на уровне алгоритмических языков. Internet-Edication-Sciense-2010. New Informational and Computer Tecnologies in Education and Sciense.p.201-206.