УДК 519.683.8

**А.В. Козлов**

# Упрощение арифметических выражений

В статье рассматривается методика упрощения арифметических выражений.
**Ключевые слова:** парсер, алгебраический тип, Haskell, упрощение выражений, терм, выражение, фактор.

**Введение**

Многие экономические и технологические процессы можно описать с помощью математических формул, но вид этих формул зачастую приобретает сложные и громоздкие выражения. С определенной долей погрешности сложные выражения можно преобразовать в более простые выражения, применив различные способы упрощения.

Существует много методов упрощения, описанных в математической и технической литературе [1, 2], но возникают все новые задачи, решение которых просто осуществить с помощью компьютерных программ. Для эффективного решения задачи упрощения выражений надо написать программу с «искусственным интеллектом», которая бы упрощала выражение наиболее оптимально. Самый простой способ – перебирать все варианты упрощенных выражений и выбрать вариант с наименьшим количеством операций.

Написание программы по этому пути требует много вычислительных ресурсов, ограничение связано только с ресурсами компьютера.

В настоящее время существуют различные математические пакеты программ, которые позволяют реализовать упрощение арифметических выражений. Все эти пакеты представляют собой сложные вычислительные системы, содержащие в себе сотни математических формул. Например, функции упрощения представлены в пакетах: Matlab, Mathcad, Maple. Эти функции обеспечивают упрощение математических выражений, выполняя следующие типовые действия (для простоты обозначим их как →):

- комбинируя цифровые подвыражения (3*x*5→15*x, 10*x/5→2*x);
- приводя подобные множители в произведениях (x^3*a*x→a*x^4);
- приводя подобные члены в суммах (5*x+2+3*x→8*x+2);
- используя тождества, содержащие ноль (a+0→a, x–0→x);
- используя тождества, содержащие единицу (1*x→x);
- распределяя целочисленные показатели степени в произведениях ((3*x*y^3)^2 →9*x^2*y^6);
- сокращая expr на наибольший общий полиномиальный или иной множитель;
- понижая степень полиномов там, где это возможно;
- используя преобразования, способные упростить выражения.

Эти пакеты представляют собой лицензионные программные продукты, которые дорого стоят. Также они требуют от пользователя знания, как самого пакета, так и умения хорошо программировать.

И для использования этих математических пакетов необходимо сделать запись в реестре Windows.

Целью работы является создание программы, позволяющей наиболее эффективно провести упрощение арифметических выражений, и являющейся полностью портативным исполняемым приложением, не требующей установки. Также реализованная программа может использовать для автономных вычислений (т.к. программа помимо упрощения, позволяет также вычислять выражения).

С такой программой может работать пользователь, незнакомый со сложными математическими пакетами и языками программирования. И самое главное, работа с этой программой не требует лицензионного соглашения.

Программа упрощения арифметических выражений написана на языке функционального программирования Haskell [3]. Отличие функциональных языков программирования от

объектно-ориентированных заключается в том, что вся программа строится через функции, нет ни циклов, ни операторов переприсваивания, ни условных операторов. Один из плюсов Haskell`я (рис. 1) то, что он использует ленивое вычисление, т.е. он не вычисляет аргумент, пока он не потребуется, что приводит к быстрому решению, и нет зависания программы при бесконечном цикле.

Математические задачи всегда будут существовать, а вместе с ними и задача упрощения выражений.

```
C:\>C:\main.exe
(2+5+6)*(4+6+7+2-6-3-5)
Result: 65
C:\>C:\main.exe
2-a*b-c*(a+b*(3))+3*a*c-b
Result: -1*b+(2*a*c+(-3*c*b+(-1*a*b+2)))
C:\>C:\main.exe
(a+b+c)-3*c+4*a-a*b+b+4*a*b
Result: 2*b+(5*a+(-2*c+3*a*b))
C:\>C:\main.exe
a*b*(e+2*(2+2))-a*b+a*b*e+10
Result: 7*a*b+(2*a*b*e+10)
C:\>C:\main.exe
a-b+b-a
Result: 0
C:\>_
```

Рис. 1. Выполнение программы

**Анализ и решение задачи**

Исходное выражение представляется в виде строки, которую нужно перевести в алгебраический тип. Это делается с помощью парсеров или комбинаций парсеров, которые должны распознать число или переменную, символы скобок и символы операций. Если есть другие символы, то должно появиться сообщение об ошибке.

Парсер (parser) – функция-анализатор. Парсинг – процесс синтаксического (грамматического) анализа строки. При парсинге исходный текст преобразуется в структуру данных с помощью парсера, обычно – в дерево, которое отражает синтаксическую структуру входной последовательности и хорошо подходит для дальнейшей обработки [3].

Алгебраический тип данных – в теории программирования любой тип, значения которого являются значениями некоторых иных типов, «обёрнутых» конструкторами алгебраического типа. Другими словами, алгебраический тип данных имеет набор конструкторов типа, каждый из которых принимает на входе значения определённых типов и возвращает значение конструируемого типа. Важное отличие конструктора типа от функции заключается в том, что конструктор не исполняется, но единственная его задача – создание значения своего типа на основе входных значений. Для работы с такими значениями используется механизм сопоставления с образцами, как наиболее эффективный способ разбора значений (но это не означает, что иные механизмы работы со значениями не применимы к алгебраическим типам данных) [4].

После перевода строки в алгебраический тип нужно с помощью специальных функций упростить выражение. А после упрощения нужно написать функцию, которая переводит полученное выражение в алгебраическом типе в строку.

Перевод строки в алгебраический тип можно с разными структурами и, следовательно, упрощать придется по-разному.

Есть много вариантов упрощения арифметических выражений: последовательно проходя по выражению, выполнять операции, вызывать последовательно функции упрощения (складывать/вычитать числа, складывать/вычитать переменные или совокупности переменных и чисел, умножать и делить числа и переменные, раскрывать скобки) и т.д.

При реализации были использованы уже имеющиеся парсеры и структура алгебраического типа.

*data Expression = Trm Term | AddOp Term Expression | SubOp Term Expression,*

где выражение может быть либо Терм, либо сложение/вычитание из Терма какого-либо выражения. Например, *a+1* – Выражение.

*data Term = Ftr Factor | MultOp Factor Term | DivOp Factor Term,*

где Терм, в свою очередь может быть либо Фактором, либо умножением/делением Фактора на Терм. Например, *a*b* – Терм.

*data Factor = Number Int | Identifier String | Expr Expression,*

где Фактор – число, переменная, либо Выражение, взятое в скобки: (1+8).

Для работы программы были взяты две функции (readExpt, showExpr), которые переводят строку в алгебраический тип и наоборот.

Пример того, как арифметическое выражение выглядит в алгебраическом типе:

*readExpr «(2+6)*(4–5)»*

*Trm (MultOp (Expr (AddOp (Ftr (Number 2)) (Trm (Ftr (Number 6))))))*
*(Ftr (Expr (SubOp (Ftr (Number 4)) (Trm (Ftr (Number 5)))))))*

Алгоритм программы:

Программа в цикле вызывает каждый раз «основную функцию» до тех пор, пока было зафиксировано какое-либо упрощение функцией-анализатором. Основная функция по порядку вызывает 5 функций упрощения.

*> glav ::Expression -> Expression*

*> glav e|(prask (rask3 e))||(prs (rask (rask3 e)))||(pskob1*
*(sclprm (rs (rask (rask3 e)))))||(prask3 e)||(psclad1 (zm (rs (rask (rask3 e))))) =*
*= glav (glav (skob1 (sclprm (rs(rask (rask3 e))))))*

*> glav e= skob1 (sclprm (rs(rask (rask3 e))))*

Алгоритм каждой функции упрощения основан на применении своих правил упрощения в зависимости от ситуации.

**Функции упрощения выражения**

В программе реализовано 5 функций упрощения выражения:

1. Инициализация выражений в скобках;
2. Упрощение операций умножения и деления;
3. Упрощение операций сложения и вычитания;
4. Сложение и вычитание переменных;
5. Упрощение операций со скобками.

Рассмотрим работу алгоритма на примере:

*a+b+c+(r+a–c)*4+7–9*r–(a+b)*(c+d)+(1+(3+a))*

Результаты вызова основной функции по шагам:

1) 1. *a+b+c+(r+a–c)*4+7–9*r–(a+b)*(c+d)+(4+a)*
(Результат после вызова «Основной функции» в выражениях в скобках)

2. *a+b+c+(r+a–c)*4+7–9*r–(a+b)*(c+d)+(4+a)*
(Результат после функции упрощения операций умножения и деления)

3. *a+b+c+(r+a–c)*4+7–9*r–(a+b)*(c+d)+(4+a)*
(Результат после функции упрощения сложения и вычитания)

4. *a+b+c+(r+a–c)*4+7–9*r–(a+b)*(c+d)+(4+a)*
(Результат после функции сложения и вычитания всех переменных)

5. *a+b+c+4*r+4*a–4*c+7–9*r–a*c–a*d–b*c–b*d+4+a*
(Результат после функции упрощения выражений со скобкой)

*2)* 1. *a+b+c+4*r+4*a−4*c+7−9*r−a*c−a*d−b*c−b*d+4+a*
2. *a+b+c+4*r+4*a−4*c+7−9*r−a*c−a*d−b*c−b*d+4+a*
3. *a+b+c+4*r+4*a−4*c+7−9*r−a*c−a*d−b*c−b*d+4+a*
4. *a+b+c+4*r+4*a−4*c+11−9*r−a*c−a*d−b*c−b*d+a*
5. *6*a+b−3*c−5*r+11−a*c−a*d−b*c−b*d*
6. *6*a+b−3*c−5*r+11−a*c−a*d−b*c−b*d*

**Описание функций упрощения**

Алгоритм работы функции по инициализации выражений в скобках: рассматривая Выражение, берутся все Термы. Далее рассматривается каждый Терм, и если находится в нем Фактор, равный Выражению, взятому в скобки, то вызывается Основная функция для Выражения, находящегося в скобках.

```
> rask3 :: Expression -> Expression
> rask3 (Trm t) = (Trm (rsk t))
> rask3 (AddOp t e) = (AddOp (rsk t) (rask3 e))
> rask3 (SubOp t e) = (SubOp (rsk t) (rask3 e))
> rask3 e = e
> rsk :: Term -> Term
> rsk (MultOp (Number n) t) = (MultOp (Number n) (rsk t))
> rsk (MultOp (Identifier a) t) = (MultOp (Identifier a) (rsk t))
```

Эта функция применяет такое правило упрощения – найти все Термы и проверить их на следующие правила:

1) Если Терм оказался операцией умножения или деления какой-либо переменной или числа на новый Терм, то вызываем операцию рекурсивно от него. А если была операция умножения или деления с Выражением в скобках, то еще и вызываем Основную функцию от этого параметра.
2) Если Терм оказался Выражением в скобках, то вызываем Основную функцию от него.

Функция «Упрощение операций умножения и деления» – упрощает выражения, связанные с умножением и делением (такие как умножение или деление на единицу, умножение на ноль, умножение или деление двух чисел) и сокращением переменных или выражений, находящихся в скобках при делении. Так же при умножении нескольких переменных и чисел эта функция переносит получившееся число в начало выражения. (Например, *a*c*7 => 7*a*c*)

```
> rask :: Expression -> Expression
> rask (Trm (MultOp f t)) = (Trm ( uprMul1(uprMul1 (MultOp f (uprMul1 t)))))
> rask (Trm (DivOp f t)) = (Trm ( uprMul1(uprMul1 (DivOp f (t)))))
> rask (AddOp t e) = (AddOp (uprMul1 t) (rask e))
> rask (SubOp t e) = (SubOp (uprMul1 t) (rask e))
> rask e = e
> uprMul1 :: Term->Term
> uprMul1 (DivOp f (Ftr(Number 1))) = ( Ftr f )
> uprMul1 (DivOp f ( DivOp (Number 1) t)) = ( uprMul1 ( DivOp f t ) )
```

Также как и предыдущая, эта функция находит все термы и применяет к ним правила упрощения. Вот некоторые из них:

1) Если Терм – операция умножения на ноль, то убираем этот Терм;
2) Если Терм – операция умножения на 1, то эта операция и 1 убирается;
3) Если Терм – операция деления двух одинаковых Факторов, то эта операция вместе с параметрами убирается;

4) Если Терм – операция умножения или деления двух чисел, то выполняется эта операция.

Функция «Упрощение операций сложения и вычитания» – упрощает сложение и вычитание чисел, путем складывания и вычитания всех чисел в выражении, и убирает все операции сложения и вычитания нулей.

Эта функция работает так.

Анализируя выражение, она при встрече двух чисел подряд или хотя бы через один Терм, сразу выполняет операции сложения или вычитания. Но если встретилось число, а рядом другого числа нет, то она переносит это число в конец текущего Выражения.

```
> rs ::Expression -> Expression
> rs e = rask1 (tochka e)
> rask1 :: Expression -> Expression
> rask1 (SubOp t e) = (uprAdd1 (SubOp t e))
> rask1 (AddOp t e) = (uprAdd1 (AddOp t e))
> rask1 e = e
> uprAdd1 :: Expression->Expression
> uprAdd1 ( AddOp (Ftr (Number n)) (Trm (Ftr (Number 0)))) = ( Trm ( Ftr (Number n) ))
> uprAdd1 ( AddOp (Ftr (Number 0)) (Trm (Ftr (Number n1)))) = ( Trm ( Ftr (Number n1) ))
> uprAdd1 ( AddOp (Ftr (Number 0)) (AddOp t e) ) = ( uprAdd1 (AddOp t (uprAdd1 e)))
> uprAdd1 ( AddOp t (Trm (Ftr (Number 0))) ) = (Trm t)
> uprAdd1 ( SubOp t (Trm (Ftr (Number 0))) ) = (Trm t)
```

Функция «Сложение и вычитание переменных» – складывает и вычитает переменные по алгоритму:

1. Заменяет вычитание переменных на их сложение. Происходит это следующим образом: все переменные, которые идут в операциях вычитания в роли вычитаемого, умножаются на коэффициент «–1», и операция вычитания заменяется сложением.
2. Из всех Термов извлекаются переменные и помещаются в список в соответствующем порядке, причем переменные, на которые делят, идут со специальным знаком «~».
3. Все списки между собой сравниваются и, если попадутся два одинаковых, то их коэффициенты складываются. Если окажется, что коэффициент равен нулю, то оба выражения удаляются.

```
> sclprm :: Expression -> Expression
> sclprm e = sclad1 (zm e)
> zm:: Expression -> Expression
> zm (AddOp t e) = (AddOp t (zm e))
> zm (SubOp t (Trm (Ftr (Number n)))) = (SubOp t (Trm (Ftr (Number n))))
> zm (SubOp t (Trm t1)) = (AddOp t (Trm (MultOp (Number (−1)) t1)))
> zm (SubOp t (AddOp (Ftr (Number n)) e)) = (SubOp t (AddOp (Ftr (Number n)) (zm e)))
> zm (SubOp t (SubOp (Ftr (Number n)) e)) = (SubOp t (zm (SubOp (Ftr (Number n)) e)))
> zm (SubOp t (AddOp t1 e)) = (AddOp t (AddOp (MultOp (Number (−1)) t1) (zm e)))
> zm (SubOp t (SubOp t1 e)) = (AddOp t (zm (SubOp (MultOp (Number (−1)) t1) e)))
> zm e = e
> sclad1 :: Expression->Expression
> sclad1 (AddOp (Ftr (Number n) ) e ) = (AddOp (Ftr (Number n)) (sclad1 e))
> sclad1 (AddOp t e) = (pol (AddOp t (sclad1 e)))
> sclad1 (SubOp (Ftr (Number n) ) e ) = (SubOp (Ftr (Number n)) (sclad1 e))
> sclad1 (SubOp t e) = (pol (SubOp t e))
> sclad1 e = e
```

*> pol :: Expression->Expression*

*> pol (AddOp (MultOp (Number n) t) (Trm (MultOp (Number n1) t1)))|proverka (razbor t1) (razbor t) = (Trm (MultOp (Number (n+n1)) t) )*

*> pol (AddOp (MultOp (Number n) t) (Trm t1))|proverka (razbor t1) (razbor t) = (Trm (MultOp (Number (n+1)) t) )*

Функция «Упрощение операций со скобками» – упрощает выражения, где встречаются скобки. Так как задача упрощения арифметического выражения неоднозначна, то есть, нет единого критерия упрощенности выражения, эта функция раскрывает скобки при умножении выражения в скобках на что-либо только тогда, когда внутри находится не более трех операций сложения и/или вычитания. Также она перемножает два выражения в скобках, раскрывает скобки и убирает лишние скобки.

Некоторые правила работы данной функции:

1) Если производится сложение или вычитания двух Выражений в скобках и в одно из выражений есть операции сложения и вычитания, то Терм, к которому складывают или вычитают, выносится за скобки. И Выражение заново анализируется.
2) Если в Выражении в скобках нет операций сложения или вычитания, то скобки убираются.
3) Если в Выражении в скобках производится операция вычитания или сложения двух термов, и при этом Выражение умножается на Идентификатор или число, то скобки раскрываются, а Термы умножаются на этот Идентификатор или число.

*> skob1 :: Expression->Expression*

*> skob1 (Trm t) = umnog1 (Trm t)*

*> skob1 (AddOp (Ftr f) e) = umnog1 (AddOp (Ftr f) (skob1 e))*

*> skob1 (AddOp t (Trm (Ftr f))) = (AddOp t (Trm (Ftr f)))*

*> skob1 (AddOp t e) = umnog1 (AddOp t e)*

*> skob1 (SubOp (Ftr f) e) = umnog1 (SubOp ((Ftr f)) (skob1 e))*

*> skob1 (SubOp t (Trm (Ftr f))) = (SubOp t (Trm (Ftr f)))*

*> skob1 (SubOp t e) = umnog1 (SubOp t e)*

*> skob1 e = e*

*> umnog1 :: Expression->Expression*

*> umnog1 (AddOp (Ftr (Expr (AddOp t e))) e1) = skob1 (AddOp t (AddOp (Ftr (Expr e)) e1))*

*> umnog1 (AddOp (Ftr (Expr (SubOp t e))) e1) = skob1 (SubOp t (AddOp (Ftr (Expr e)) e1))*

*> umnog1 (AddOp (Ftr (Expr (Trm t))) e1) = skob1 (AddOp t e1)*

*> umnog1 (SubOp (Ftr (Expr (AddOp t e))) e1) = skob1 (AddOp t (SubOp (Ftr (Expr e)) e1))*

*> umnog1 (SubOp (Ftr (Expr (SubOp t e))) e1) = skob1 (SubOp t (SubOp (Ftr (Expr e)) e1))*

**Тестирование алгоритма упрощения**

Тест1: *main.exe <<"(2+5+6)*(4+6+7+2–6–3–5)"*
*Result:65*

Тест2: *main.exe <<"2–a*b–c*(a+b*(3))+3*a*c–b"*
*Result:–1*b+(2*a*c+(–3*c*b+(–1*a*b+2)))*

Тест3: : *main.exe <<«(a+b+c)–3*c+4*a–a*b+b+4*a*b»*
*Result:2*b+(5*a+(–2*c+3*a*b))*

Тест4: : *main.exe <<«a*b*(e+2*(2+2))–a*b+a*b*e+10»*
*Result:7*a*b+(2*a*b*e+10)*

Тест5: : *main.exe <<«* `a-b+b-a`*»*
*Result:0*

В первом тесте приведена ситуация умножения скобки на скобку. В этом примере нет переменных, а значит, требуется просто вычислить это выражение и, следовательно, полученный результат должно быть число.

Второй тест уже намного сложнее. В нем требуется: убрать скобки при 3, раскрыть скобки и сложить полученные совокупности.

В третьем тесте нужно просто убрать скобки и полученные выражения сложить/ вычесть из тех, которые были за скобкой.

Четвертый тест похож на второй, но в нем нужно вычислить вложенную скобку и умножить две переменных на скобку.

В пятом тесте просто приведена ситуация, когда все переменные сокращаются.

**Заключение**

Упрощение арифметических выражений очень неоднозначная задача. И поэтому очень трудно определить критерий упрощенности. В статье автором был описан один из вариантов реализации алгоритма по решению такой задачи.

По результатам многочисленных тестов, можно сделать вывод, что в данной программе реализован алгоритм, который эффективно упрощает арифметические выражения.

**Литература**

1. Акритас А. Основы компьютерной алгебры с приложениями. – М. : Мир, 1994. – 544 с.

2. Бухбергер Б. Компьютерная алгебра. Символьные и алгебраические вычисления. – М. : Мир, 1986. – 392 с.

3. Зюзьков В.М. Ленивое функциональное программирование : учебное пособие. – 2-е изд., перераб. и доп. – Томск : ТУСУР, 2007. – 294 с.

4. Словари и энциклопедии на Академике // [Электронный ресурс]. – 2009. – Режим доступа : [http://dic.academic.ru/dic.nsf/ruwiki/211401/].

---

**Козлов Александр Владимирович**
Студент ТУСУРа
Эл. почта: skat2005@mail.ru

A.V. Kozlov
**Simplification of arithmetic expressions**
In clause the technique of simplification of arithmetic expressions is considered.
**Keywords:** parser, algebraic type, Haskell, simplification of expressions, term, expression, factor.

---