ОРДЕНА ЛЕНИНА
ИНСТИТУТ
ПРОБЛЕМ
УПРАВЛЕНИЯ

В.Л. КИСТЛЕРОВ

ПРИНЦИПЫ ПОСТРОЕНИЯ ЯЗЫКА АЛГЕБРАИЧЕСКИХ ВЫЧИСЛЕНИЙ FLAC

ПРЕПРИНТ

МОСКВА 1987

УДК 681.3.06

Кистлеров В.Л. Принципы построения языка алгебраических вычислений *FLAC*. - М., 1987 (Препринт/Институт проблем управления)

Предложены принципы построения языка алгебраических вычислений, предназначенного для реализации на ЭВМ алгоритмов аналитических преобразований. В основе языка лежит концепция задержанных вычислений, позволяющая при интенсиональном подходе к вычислениям конструировать любые типы данных, возникающих в алгебре. Введенный в язык аппарат синтаксического отождествления позволяет производить анализ сконструированных алгебраических структур с любой степенью детализации, а аппарат управления задержкой - интерпретировать основные управляющие конструкции традиционных языков программирования.

Рецензент к.т.н. И.Б.Мучник

Текст препринта воспроизводится в том виде, в котором представлен автором.

Утверждено к печати Редакционным советом Института

ВВЕДЕНИЕ

С середины шестидесятых годов идут интенсивные исследования механизма алгебраических вычислений с целью создания языка, позволяющего описывать алгоритмы таких вычислений с последующим их выполнением на ЭВМ. Примерами реализованных универсальных систем, преследующих эту цель, являются язык АНАЛИТИК для ЭВМ МИР-2 [1], системы АЛЬКОР [2], SANTRA-BASIC [3], MACSYMA [4], REDUCE [5] и др. Эти и множество других систем подобного рода различными способами выявляют все то разнообразие проблем, которое возникает при попытке описания процессов алгебраических вычислений в виде алгоритмов.

Основные проблемы, общие для всех этих систем, связаны, на наш взгляд, с отсутствием адекватного аппарата конструирования и анализа всего того многообразия типов значений, которое появляется в процессе вычислений. Попыткой решения этих проблем в большинстве систем является введение фиксированного набора встроенных типов значений с соответствующим аппаратом их анализа. Так, в системе REDUCE предложено несколько встроенных типов данных, среди которых - скалярные выражения, массивы, матрицы. Предоставляется также встроенный аппарат частичного анализа структуры значений этих типов. Однако аппарат этот столь недостаточен, что руководства по использованию системы об этом сразу предупреждают, и, в случае необходимости проведения более сложного анализа структуры значений, рекомендуется переходить на язык реализации системы, - в данном случае LISP. К тому же, недостаточность адекватных средств конструирования типов данных ведет к попыткам интерпретации новых типов имеющимися средствами, что, в свою очередь, приводит к громоздкости, неэффективности и, во многих случаях, к невозможности продолжения работы.

Пытаясь по-своему решить эти проблемы, мы предварительно проведем анализ и построим модель функционирования предметной области, называемой "алгебраические

вычисления". Затем предложим язык, в котором будет обеспечена возможность конструирования и анализа всех тех типов данных, которые появятся в нашей модели и будут достаточны для представления любых алгебраических об'ектов.

I. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

Основными понятиями, используемыми при описании процесса вычислений в математике, являются понятия множества и функции. При этом подразумевается, что с каждым из этих понятий связываются некоторые идеальные об'екты, имеющие заданную структуру, и обладающие определенными свойствами. Говоря о языке, описывающем различные виды этих понятий, мы будем иметь в виду некоторый специальный случай знакового моделирования. При этом модель должна адекватно отражать как структуру моделируемых об'ектов, так и их свойства. Уверенность в возможности этого проистекает из убеждения, что механизм алгебраических вычислений построен по правилам, природу которых можно смоделировать, основываясь на относительно небольшом числе понятий и принципов. Мотивы, по которым мы выберем в дальнейшем необходимые нам базовые понятия, поясним при обсуждении нижеследующих примеров работы с предложенными моделями вычислений.

В качестве примера рассмотрим один из способов моделирования натуральных чисел и построение системы рациональных чисел как результат деятельности над ними.

Вероятно, один из самых простых способов знакового моделирования понятия количества это представление его в виде строки из однородной последовательности одинаковых знаков, - например, знаков I. Тогда количество "три" будет представлено как III, количество "пять" - IIIII, нулевое количество будем пока изображать пустой

строкой. Такую знаковую модель количества будем называть натуральными числами в единичной системе счисления.

Выбор описанного способа моделирования количества представляется вполне адекватным по следующим соображениям. Определим сумму двух натуральных чисел как конкатенацию соответствующих им строк. Тогда тот факт, что сумма двух количеств есть снова количество, соответствует свойству строк символов при конкатенации образовывать строку символов. Коммутативности и ассоциативности сложения количеств соответствуют те же свойства конкатенации.

Теперь пусть у нас кроме функции сложения определена еще и рекурсивная функция вычитания. Алгоритм вычитания определим как процесс отщепления у каждого из аргументов слева по одному символу с применением к оставшимся частям аргументов снова функции вычитания. Работа алгоритма будет продолжаться до тех пор, пока правый аргумент не станет пустым. После чего вычисления прекращаются а левый аргумент выдается в качестве результата.

Мы построили знаковую модель множества натуральных чисел, где каждый элемент множества изображается строкой символов.

Для изображения нашего намерения вычислить функцию F в заданной точке x множества X мы воспользуемся традиционным обозначением F(x), где сначала идет строка символов, обозначающая имя функции, а затем строка аргумента, заключенная в скобки. Если функция определена на декартовом произведении множеств, то в соответствии с математической традицией, тот же факт будем изображать в виде P(x, y, ... , z). Если значение функции не зависит от аргументов, то можно писать только имя функции. Такие об'екты мы будем называть вызовами функций. Процесс вычисления будет заключаться в замене такого вызова функции на ее значение в заданной точке. Тогда для описанной выше функции сложения ADD и

рекурсивной функции вычитания SUB будем иметь:

ADD(III, II) -> IIIII
SUB(IIII,III) -> I

В дальнейшем, для простоты изложения, при необходимости, будем переходить к десятичной записи чисел и к инфиксной форме записи арифметических операций.

Введенные функции сложения и вычитания уже позволяют нам вычислять значения некоторых выражений, представленных в виде суперпозиции этих функций, и принимающих значения на множестве натуральных чисел. Например:

((3 + 5) - 8) + 2 -> 2

Однако, пытаясь тем же способом вычислить значение выражения:

(3 - 8) + (3 + 4)

мы сталкиваемся с тем, что наша функция SUB является частичной, и не определена при обращении SUB(0,5). Эту ситуацию нам необходимо как-то интерпретировать. Традиционный способ заключается в том, что такая ситуация об'является аварийной и вычисления прекращаются.

Неудовлетворительность подобного подхода проистекает прежде всего из того, что в данном случае значением всего выражения, является натуральное число "два". А довести вычисления до конца не удалось только потому, что порядок вычислений был задан "неудачно". При этом возможно такое эквивалентное преобразование данного алгоритма, учитывающее ассоциативность и коммутативность сложения, которое позволит довести вычисления до конца.

Другой подход, по-существу моделирующий логику построения существующего языка алгебры, заключается в следующем. Если необходимо вычислить значение функции

в точке, находящейся вне области определения, то в качестве значения функции в этой точке порождается новый синтаксический об'ект, отражающий факт вызова функции в заданной точке. Этот об'ект мы определим как строку, полностью совпадающую с текстом вызова. Такой способ доопределения функций будем называть *задержкой вычислений функции*, или просто *задержкой функции*. Значение, получающееся в результате такой задержки, будем называть *интенсионалом функции* или, для большей краткости, просто *интенсионалом*.

В нашем примере, вычитая восемь из трех, мы получим в качестве значения строку SUB(0,5). После чего вычисления будут продолжаться, но при этом множество значений пополнится порожденным интенсионалом функции SUB.

Необходимость задержки мотивируется потребностью дальнейших вычислений наружных вызовов функций, которые продолжаются, но уже на расширенной области определения. Ясно, что для продолжения вычислений необходимо доопределить какие-то функции в точках расширения. Простейший способ заключается в том, что всякая функция, значение которой вычисляется в точке из расширенной части области, в свою очередь задерживается на таком аргументе. Если же точка принадлежит ранее допустимой части области определения, то вычисления продолжаются как и прежде. Например, выражение:

```
(3 - 8) + (3 + 4)
```

вычисленное по предложенным правилам, будет приведено к виду:

```
ADD(SUB(0,5),7)
```

Здесь произошла задержка как функции SUB, так и функции ADD. При этом мы, в некотором смысле, продвинулись к

цели. Так что, хотя вычисления с нашей точки зрения и не завершены, но подвыражение 3 + 4, составляющее часть всего выражения, вычислилось. Такие вычисления, когда единственным способом автоматического доопределения функции является задержка, мы будем называть **частичными вычислениями**.

Доопределяя все функции только таким однообразным способом как задержка, мы, по-существу, рассматриваем интенсионал, на котором производим задержку в процессе частичных вычислений, как об'ект, о котором нам совершенно ничего не известно. Но ведь интуитивно ясно, что в данном примере, когда мы задерживаем сложение на интенсионале ранее задержанного вычитания, у нас были основания рассчитывать на более успешное продвижение вычислений. Ибо задерживая в последнем примере вычитание и порождая интенсионал SUB(0,5), мы исходили из того, что может быть нам и не нужно было вычитать пять из нуля, но зато в дальнейшем, возможно, нам надо будет складывать этот интенсионал с каким-либо числом, например, семь. Тогда в результате анализа структуры интенсионала, поданного в качестве аргумента функции ADD, будет учтено, что содержательно интенсионал SUB(0,5) это незавершенное вычитание, и функция ADD будет доопределена так, чтобы вычисления могли быть корректно продолжены. Один из вариантов такого доопределения может быть следующим. Сначала следует поместить 7 в качестве левого аргумента функции SUB, а потом продолжить вычисление функции SUB при новых значениях аргументов. После чего мы, наконец, получаем искомый результат 2.

Таким образом, мы имеем новый тип данных в виде интенсионалов, полученных в результате задержки вычитания на паре натуральных чисел. Мы склонны трактовать такие интенсионалы как отрицательные числа. Аналогично мы можем интерпретировать рациональные числа как интенсионалы, полученные в результате задержки частичной

функции деления на паре целых чисел. Так, пытаясь разделить I на 2 мы получаем интенсионал DIV(I,2), или в инфиксной записи I/2. Определение еще одной обратной функции - квадратного корня SQRT, дает пример другой частичной функции, порождающей такие, например, интенсионалы как SQRT(2) или SQRT(-I).

До сих пор рассматривались только арифметические функции и интенсионалы, полученные их задержкой. Алгебраические же выражения содержат, кроме того, об'екты, называемые переменными и вызовами функций. Мы будем моделировать их интенсионалами функций без аргументов, и с аргументами соответственно. Если внешние функции не будут специально доопределяться на этих интенсионалах, то частичные вычисления все равно будут произведены автоматически. Так что, если необходимо вычислить выражение x+I, то функция сложения просто задерживается, и значением выражения будет ADD(x,I). Аналогично случаю арифметических интенсионалов можно явно доопределить функции на интенсионалах переменных и вызовов функций какими-то соответствующими значениями.

Таким образом, в нашей модели выделены следующие элементы механизма алгебраических вычислений:

задержка вычислений - порождение интенсионалов;

частичные вычисления на расширенной области определения;

анализ структуры интенсионалов для явного доопределения частичных функций на них.

В языке алгебраических вычислений должен быть аппарат для адекватного выражения всех этих действий.

2. ПОСТРОЕНИЕ ЭЛЕМЕНТОВ ЯЗЫКА

Рассмотрим функции F: X -> Y и G: Y -> Z. Каждая точка множества X отличается для нас от другой точки только своим уникальным именем. Имена элементов множества будем изображать строкой символов. Например, x, xI, Z28A – описание трех различных элементов. Семантически каждое из этих имен представляется нам целостным, неделимым об'ектом и, теоретически, не требует анализа его синтаксической структуры. В дальнейшем такие имена, являющиеся идентификаторами, будем называть простыми об'ектными термами.

Элементы множества Y первоначально могли обозначаться только простыми об'ектными термами, но в дальнейшем, согласно нашей концепции задержанных вычислений, множество значений может быть пополнено элементами, обозначаемыми строкой вида F(x) или F(xI,..., xN). Такие интенсионалы получились в результате задержки частичной функции F, так что мы можем рассматривать их как имена новых точек из расширенного множества Y. Аналогично множество Z будет пополнено элементами с именами вида G(yI), G(F(xI,x2),yI) и т.п. Об'екты с подобной синтаксической структурой будем называть в дальнейшем аппликативными об'ектными термами. Существенным свойством таких имен является то, что в них зафиксирована их собственная "история" появления. Анализ этой истории нам может быть нужен для того, чтобы явно доопределять на таких элементах функцию G. Для этого в языке необходимо средство синтаксического анализа подобных структур.

Один из простейших способов описания функции заключается в задании ее в виде таблицы, каждая строка которой задает пару: аргумент – значение. Например:

F(0) = I;
F(I) = 3;

Но такой способ описания не устраивает нас, когда множество элементов X бесконечно или слишком велико. Мы должны выбрать способ описания функции, определенной на бесконечном множестве, конечным числом предложений. Для этого необходимо, чтобы одним предложением можно было описывать значение функции сразу на целом классе точек из X с одинаковой, на данном шаге процесса вычислений, вычислительной "судьбой". При этом описание конечным числом предложений возможно только тогда, когда существует конечное покрытие множества X семейством таких классов. Поскольку точки этого множества представляются нам только своими именами, то каждому вышеупомянутому семантическому классу должен соответствовать синтаксический класс имен, описывающийся одной строкой на нашем языке.

Для описания синтаксической структуры интенсионалов нам понадобятся два типа переменных. Один тип называется п е р е м е н н о й т е р м а, и значением переменной такого типа может быть только строка, являющаяся простым или аппликативным термом. Для того, чтобы в тексте описания строки можно было отличить обозначение переменной от текста об'ектного терма, перед именами переменных будем помещать специальные символы, которые одновременно будут играть роль синтаксических указателей типа переменных. Обозначение переменной терма представляет собой сим ол "&", непосредственно за которым следует имя этой переменной, являющееся идентификатором. Например, &X, &YI – две переменных терма. Нам понадобится еще один тип переменных: это п е р е м е н н а я с п и с к а т е р м о в. Ее обозначение представляет собой символ "#", непосредственно за которым следует имя этой переменной, являющееся идентификатором. Значением такой переменной может быть список об'ектных термов, представляющих собой последовательность об'ектных термов, разделенных запятыми, либо один об'ектный терм. Теперь у нас есть достаточно выразительных средств для

демонстрации описания классов интенсионалов. Например:

```
F(&X, A)
G(&X, A, #Y)
H(&Y, F(&Y), &C(#A))
```

Элементы каждого из этих классов получаются подстановкой вместо переменных каждого типа значений, допустимых для данного типа переменной. Такое описание, содержащее об'ектные термы и переменные, называется **списком типовых термов**.

Описание функции теперь можно представить в виде последовательности предложений, каждое из которых имеет следующий вид:

<ЛЕВАЯ ЧАСТЬ> = <ПРАВАЯ ЧАСТЬ>;.

Здесь левая часть описывает класс точек из области определения функции, а правая часть – способ вычисления значения функции для точек этого класса.

Вычисления значения функции в заданной точке начинаются с определения того, к какому классу принадлежит точка. Имя точки задается в виде списка об'ектных термов. Будем говорить, что список об'ектных термов L0 отождествляется с данным списком типовых термов LT, если существует такой набор значений для переменных из LT, что после подстановки этих значений вместо соответствующих переменных в LT, с учетом выше сформулированных условий, результат совпадает со строкой L0. Имя функции, за которым в скобках следует список об'ектных термов, описывающих точку вызова, будем называть интенсионалом функции. Если вызов функции не содержит аргументов, то так будем называть простой об'ектный терм, являющийся именем функции. Принадлежность точки какому-то классу, означает отождествимость вызова функции с левой частью соответствующего предложения. При выбранном нами способе описания классы

могут пересекаться. Так что вызов функции может отождествиться с левыми частями нескольких предложений. Для того, чтобы ввести однозначность в выборе предложения, правая часть которого будет определять способ вычисления значения, уточним, что из всех таких предложений следует выбрать первое в порядке следования в тексте описания функции.

Если вызов функции не отождествился ни с одной левой частью предложений, то значением функции в данном случае будет интенсионал, представляющий собой об'ектный терм, синтаксически совпадающий с текстом этого вызова.

Если все же вызов функции отождествился с левой частью какого-то предложения, то это значит, что для всех переменных, из этой левой части, нашелся подходящий набор значений. Правая часть предложения представляет собой список типовых термов, содержащих только те переменные, которые использовались в левой части. Подставив вместо этих переменных соответствующие значения, полученные на этапе отождествления, мы снова получим список об'ектных термов. После чего любой из об'ектных термов этого списка можно снова рассматривается как вызов функции и его значение вычисляется вышеописанным способом. Список об'ектных термов естественно считать окончательным значением, если ни один из его термов не отождествляется ни с одной левой частью предложений программы. Следует заметить, что окончательное значение составлено только из интенсионалов функций. Другого "строительного материала" мы не использовали.

3. ОПИСАНИЕ ЯЗЫКА

Программа на языке FLAC есть последовательность предложений, каждое из которых представляет собой описание редукции над множеством допустимых строк -

списками об'ектных термов. Синтаксис термов определим при помощи формы Бэкуса – Наура (БНФ).

⟨ОБ'ЕКТНЫЙ ТЕРМ⟩ ::= ⟨ПРОСТОЙ ОБ'ЕКТНЫЙ ТЕРМ⟩ |
⟨АППЛИКАТИВНЫЙ ОБ'ЕКТНЫЙ ТЕРМ⟩

⟨ПРОСТОЙ ОБ'ЕКТНЫЙ ТЕРМ⟩ ::= ⟨ИДЕНТИФИКАТОР⟩

⟨АППЛИКАТИВНЫЙ ОБ'ЕКТНЫЙ ТЕРМ⟩ ::=
⟨ИМЯ ТЕРМА⟩ (⟨СПИСОК ТЕРМОВ⟩)

⟨СПИСОК ТЕРМОВ⟩ ::= ⟨ТЕРМ⟩ | ⟨ТЕРМ⟩ , ⟨СПИСОК ТЕРМОВ⟩

⟨ТЕРМ⟩ ::= ⟨ОБ'ЕКТНЫЙ ТЕРМ⟩

⟨ИМЯ ТЕРМА⟩ ::= ⟨ИДЕНТИФИКАТОР⟩

⟨ИДЕНТИФИКАТОР⟩ ::= ⟨БУКВА⟩ |
⟨ИДЕНТИФИКАТОР⟩ ⟨БУКВА⟩ |
⟨ИДЕНТИФИКАТОР⟩ ⟨ЦИФРА⟩

Все вычисления будем моделировать преобразованием термов описанного вида. Преобразования описываются программой, синтаксис которой опишем также в виде БНФ.

⟨ПРОГРАММА⟩ ::= ⟨ПОСЛЕДОВАТЕЛЬНОСТЬ ПРЕДЛОЖЕНИЙ⟩

⟨ПОСЛЕДОВАТЕЛЬНОСТЬ ПРЕДЛОЖЕНИЙ⟩ ::= ⟨ПРЕДЛОЖЕНИЕ⟩ ; | .
⟨ПРЕДЛОЖЕНИЕ⟩ ; ⟨ПОСЛЕДОВАТЕЛЬНОСТЬ ПРЕДЛОЖЕНИЙ⟩

⟨ПРЕДЛОЖЕНИЕ⟩ ::= ⟨ЛЕВАЯ ЧАСТЬ⟩ = ⟨ПРАВАЯ ЧАСТЬ⟩ |
⟨ПРЕДЛОЖЕНИЕ ЗАДЕРЖКИ⟩

⟨ПРЕДЛОЖЕНИЕ ЗАДЕРЖКИ⟩ ::= ⟨ЛЕВАЯ ЧАСТЬ⟩ #

⟨ЛЕВАЯ ЧАСТЬ⟩ ::= ⟨ПРОСТОЙ ОБ'ЕКТНЫЙ ТЕРМ⟩ |
⟨ИМЯ ТЕРМА⟩ (⟨ СПИСОК ТИПОВЫХ ТЕРМОВ⟩)

<ПРАВАЯ ЧАСТЬ> ::= <СПИСОК ТИПОВЫХ ТЕРМОВ>

<СПИСОК ТИПОВЫХ ТЕРМОВ> ::= <ТИПОВОЙ ТЕРМ> /
<ТИПОВОЙ ТЕРМ> , <СПИСОК ТИПОВЫХ ТЕРМОВ

<ТИПОВОЙ ТЕРМ> ::= <ПРОСТОЙ ТИПОВОЙ ТЕРМ> /
<АППЛИКАТИВНЫЙ ТИПОВОЙ ТЕРМ>

<ПРОСТОЙ ТИПОВОЙ ТЕРМ> ::= <ПРОСТОЙ ТЕРМ> /
<ПЕРЕМЕННАЯ>

<ПЕРЕМЕННАЯ> ::= <ПЕРЕМЕННАЯ ТЕРМА> !
<ПЕРЕМЕННАЯ СПИСКА ТЕРМОВ>

<ПЕРЕМЕННАЯ ТЕРМА> ::= &<ИМЯ ПЕРЕМЕННОЙ>

<ПЕРЕМЕННАЯ СПИСКА ТЕРМОВ> ::= #<ИМЯ ПЕРЕМЕННОЙ>

<ИМЯ ПЕРЕМЕННОЙ> ::= <ИДЕНТИФИКАТОР>

<АППЛИКАТИВНЫЙ ТИПОВОЙ ТЕРМ> ::=
<УКАЗАТЕЛЬ ИМЕНИ ТЕРМА> (<СПИСОК ТИПОВЫХ ТЕРМОВ>)

<УКАЗАТЕЛЬ ИМЕНИ ТЕРМА> ::= <ИМЯ ТЕРМА> !
<ПЕРЕМЕННАЯ ТЕРМА>

В правой части предложения могут использоваться только те переменные, имена которых встречались в левой части. Действие имен переменных локализовано в пределах одного предложения. В левой части предложения может использоваться только одна переменная списка термов на одном уровне списка термов.

Т Е О Р Е М А 3.I. Если в списке об'ектных термов один из термов списка заменить на список об'ектных термов, то результатом будет снова список об'ектных термов.

С каждым типом переменных связывается множество допустимых значений. Допустимым значением для переменной терма является любой об'ектный терм; для перемечной списка термов – список об'ектных термов.

О П Р Е Д Е Л Е Н И Е 3.1. Подстановкой DELTA называется множество упорядоченных пар вида ⟨M, A⟩, где M – переменная, A – допустимое для данного типа переменнсй значение.

О П Р Е Д Е Л Е Н И Е 3.2. Функция S, отображающая декартово произведение множества списков типовых термов LT и множества подстановок DELTA в множество списков об'ектных термов LO, называется функцией подстановки. Значение функции подстановки на паре ⟨LT,DELTA⟩ вычисляется следующим образом. Для каждой переменной M, входящей в строку LT, находится значение A, описаннсе для этой переменной в подстановке DELTA, и это значение подставляется в строку LT вместо переменной M.

Функция подстановки, определенная выше, является частичной. Сужение этой функции называется правильной функцией подстановки, если ее область определения удовлетворяет следующим условиям.

1. В подстановке, на которой определена функция, имя каждой переменной из LT встречается ровно один раз.

2. Если в списке типовых термов LT имеется переменная терма M, являщаяся указателем имени аппликативного терма, то значением для этой переменной, указанным в подстановке DELTA, является простой об'ектный терм (условие аппликативности).

Т Е О Р Е М А 3.2. Правильная функция подстановки S: ⟨DELTA,LT⟩ – LO всюду определена.

В дальнейшем нас будет интересовать только

правильная функция подстановки, и мы будем называть ее просто функцией подстановки.

О П Р Е Д Е Л Е Н И Е 3.3. Функция R, обратная к функции подстановки S, и вычисляющая подстановку DELTA по заданной паре – строка типовых термов ⌊T – строка об'ектных термов ⌊O, называется функцией отождествления.

Функция отождествления также является частичной функцией, поэтому вне области определения доопределим ее специальным значением "отождествление невозможно". В этом случае мы также будем говорить, что строка ⌊O не отождествилась со строкой ⌊T.

Процесс вычисления заключается в редукции списка об'ектных термов к терминальному списку об'ектных термов по заданной программе.

О П Р Е Д Е Л Е Н И Е 3.4.

1. Простой об'ектный терм называется терминальным, если он не отождествляется ни с одной левой частью предложений программы, кроме предложений задержки.

2. Аппликативный об'ектный терм называется терминальным, если сам терм не отождествляется ни с одной левой частью предложений, кроме предложений задержки, а его внутренний список термов терминален.

3. Список об'ектных термов называется терминальным, если каждый терм этого списка терминален.

Терминальными об'ектными термами мы будем моделировать интенсионалы функций.

О П Р Е Д Е Л Е Н И Е 3.5. Правило конверсии об'ектного терма. Пусть задан об'ектный терм ⌊O и программа P. Если ⌊O есть аппликативный терм, то предварительно конвертируется его список термов по определению 3.7, и результат конверсии подставляется вместо

прежнего списка. Затем проверяется терминальность ⌊O в смысле определения 3.4. Возможны два варианта.

A) Если ⌊O терминален, то результатом конверсии является он сам.

B) Если ⌊O не терминален, то он редуцируется по правилу редукции из определения 3.6.

О П Р Е Д Е Л Е Н И Е 3.6. Правило редукции об'ектного терма. Рассмотрим первое предложение программы, левая часть которого TI отождествляется с заданным об'ектным термом ⌊O. Правая часть этого предложения ⌊T является списком типовых термов. Вычислим подстановку DELTA по следующей формуле:

DELTA = R(TI, ⌊O)

где R – функция отождествления из определения 3.3. Учитывая условия выбора предложения значение DELTA определено. Если пара ⟨⌊T, DELTA⟩ не удовлетворяет условию аппликативности, то все вычисления прекращаются, и фиксируется ситуация "нарушение условия аппликативности". В противном случае результат редукции ⌊R вычисляется как:

⌊R = S(⌊T, DELTA).

Из теоремы 3.2 следует, что ⌊R является списком об'ектных термов.

О П Р Е Д Е Л Е Н И Е 3.7. Правило конверсии списка об'ектных термов. Список об'ектных термов конвертируется по одному терму последовательно слева направо. Каждый терм списка ковертируется по правилу конверсии терма, и результат конверсии подставляется вместо него в список.

По теореме 3.1 результатом применения правила конверсии списка будет список об'ектных термов.

Работа программы начинается с конверсии некоторого заранее заданного об'ектного терма и продолжается до тех пор, пока в результате вычислений не получится список терминальных об'ектных термов.

Этим мы пока завершим описание основных понятий языка FLAC.

4. ПРОСТЕЙШИЕ ПРИМЕРЫ

Опишем на нашем языке функции арифметики над числами. Здесь мы будем демонстрировать технику работы с концепциями из раздела I.

Представим натуральные числа в виде аппликативных термов с именем N и со списком одинаковых простых термов с именем I. Например, число 4 будет представлено как N(I,I,I,I). Число ноль представим как простой терм N. Тогда функция сложения будет иметь следующий вид.

```
ADDN( N, N )       = N;
ADDN( N, N(#Y) )   = N(#Y);
ADDN( N(#X), N )   = N(#X);
ADDN( N(#X), N(#Y) ) = N(#X,#Y);
```

Здесь и далее мы предполагаем, что функции, о которых явно не оговорено противное, нигде более не определены. В данном случае это относится к функциям I, Z и N, которые должны быть заданы. Декартово произведение множества на себя будем называть декартовым квадратом. Видно, что функция ADDN определена всюду на декартовом квадрате множества натуральных чисел, с нулем. Теперь опишем функцию вычитания, имеющую ту же область определения.

```
SUBN( &X,&X )   = N;
SUBN( N(I,#X), N(I) ) = N(#X);
```

```
SUBN( N(I), N(I,#Y) )  =  SUBN( N, N(#Y) );
SUBN( N(I,#X), N(I,#Y) )  =  SUBN( N(#X), N(#Y) );
```

Здесь мы сталкиваемся с тем, что в процессе вычислений будут порождаться интенсионалы вида:

```
SUBN( N, N(#Y))
```

Такие интенсионалы мы будем интерпретировать как отрицательные числа.

Теперь для того, чтобы можно было продолжить вычисления, нужно доопределить наши функции на декартовом квадрате натуральных чисел с нулем, пополненных множеством интенсионалов функции SUBN. Для этого добавим к описанию функции ADDN следующие предложения.

```
ADDN( SUBN(N, N(#Y) ), N(#X))  =
                              SUBN( N(#X), N(#Y) );
ADDN( N(#X), SUBN(N, N(#Y)) )  =
                              SUBN( N(#X), N(#Y) );
ADDN( SUBN(N, &X), SUBN(N, &Y) )  =
                        SUBN( N, ADDN(&X, &Y) );
```

понятно, что мы не доопределяем функцию ADDN на интенсионалах функции ADDN, поскольку она нигде не задерживается внутри интересущей нас области. Подобным же образом доопределим функцию SUBN.

```
SUBN( &X, SUBN(N, &Y) )  =  ADDN( &X, &Y );
SUBN( SUBN(N, &X), &Y )  =
                        SUBN( N, ADDN(&X, &Y) );
```

Аналогично описываются функции умножения MULN и деления DIVN. Поскольку функция DIVN является частичной, и снова будут порождаться интенсионалы, то теперь все функции следует доопределить не только на интенсионалах функции SUBN, но и на интенсионалах функции DIVN.

Последние интенсионалы мы будем интерпретировать как рациональные числа.

Введенных изобразительных средств уже достаточно, чтобы описывать функции над множеством целых или рациональных чисел. Для примера возьмем функцию факториал, которая для общности демонстрации будет описываться и над множеством рациональных чисел.

```
FACT( N )   =  N(I);
FACT( &N )  =  MULN( &N, FACT( SUBN( &N, N(I) ) ) );
```

Из приведенных примеров видно, что тексты описаний функций тем более громоздки, чем больше в них глубина вложенности суперпозиций функций основных арифметических операций, наиболее часто встречающихся в алгебраических выражениях. В традиционной записи математических текстов эта проблема решается путем введения инфиксной формы записи для операций, наиболее распространенных бинарных функций. Так, вместо термовой записи выражения

```
ADD(ADD(I, Y), X)
```

в инфиксной форме будет:

```
I + Y + X
```

Последняя форма записи удобнее для визуального восприятия, но пока не изобразима на нашем языке. Поэтому мы воспользуемся тем фактом, что выражение с инфиксной формой записи операций имеет эквивалентное термовое представление, и введем в наш язык следующие расширения.

Всюду, где раньше мог быть только типовой терм, теперь может быть любое правильное алгебраическое выражение, в котором терминальными операндами являются типовые термы. Такое выражение преобразуется в эквивалентную термовую форму с учетом общепринятого

старшинства четырех арифметических операций и порядка выполнения одинаковых операций слева направо; унарного минуса, возведения в степень и с учетом порядка вычислений, указанного скобками. При этом преобразовании бинарным инфиксным операциям "+", "-", "*", "/", "**" - ставятся в соответствие бинарные функции с именами ADD, SUB, MUL, DIV, POWER соответственно, а унарной операции "-" - унарная функция с именем MINUS. Так, выражение

```
-A*&X + (Y**&N - &B)
```

будет преобразовано в термовый вид:

```
ADD( MINUS(MUL(A, &X)), SUB(POWER(Y, &N), &B)) ).
```

Обратное отображение может быть запрограммировано на самом REL AC'е при преобразовании в символьный вид для визуализации результатов вычислений.

Теперь функцию факториала можно записать следующим образом:

```
FACT( N )    =  N(I):
FACT( &N )      = &N * FACT( &N - N(I) ):

&X - &Y   =  SUBN(&X, &Y):
&X * &Y   =  MULN(&X, &Y):
```

Последние два предложения доопределяют функции SUB и MUL, которые, в противном случае, просто бы задержались.

Ранее мы уже говорили, что числа мы представляем в виде интенсионалов функций I, N, SUB и DIV. Для удобства введем новый тип данных "рациональное число". Тип будем интерпретировать при помощи аппликативного терма, где имя терма будет играть роль имени типа, а список термов этого аппликативного терма - роль значения. Так, имя типа значения "рациональное число" будет RNUM,

а списком термов будут интенсионалы вышеописанной структуры.

Теперь расширим наш язык так, что всюду, где раньше мог быть типовой терм, теперь может быть традиционная запись рациональных чисел в десятичной системе счисления. Подобно случаю с инфиксными операциями они будут преобразовываться к термовой форме. Например:

```
-3/2  ->  RNUM( DIVN( SUBN( N, N(I,I,I) ), N(I,I) ) )
```

Теперь еще раз продемонстрируем описание функции факториала, но теперь уже с учетом последних расширений языка.

```
FACT(0)  =  I;
FACT(&N) =  &N * FACT(&N - I);

RNUM(&X) - RNUM(&Y)  =  RNUM( SUBN( &X, &Y ) );
RNUM(&X) * RNUM(&Y)  =  RNUM( MULN( &X, &Y ) );
```

Последние два предложения как и прежде доопределяют функции SUB и MUL, но уже на новом типе данных – "рациональное число".

5. АЛГОРИТМЫ РЕДУКЦИИ

Рассмотрим алгоритм редукции об'ектного терма ⌊O, описанный в определении 3.6. Первый этап работы алгоритма заключается в нахождении соответствующего предложения программы, с левой частью TI которого отождествляется ⌊O, и вычислении подстановки DELTA. На втором этапе вычисляется список об'ектных термов ⌊R, как значение функции подстановки на правой части ⌊T соответствующего предложения и подстановки DELTA. После чего ⌊R опять должен конвертироваться по правилу 3.5.

Например, пусть некоторый фрагмент программы имеет вид:

P = F(A):

F(&X) = &X, &X:

тогда, если вычисления начать с об'ектного терма P, то на втором шаге конверсии он будет приведен к списку об'ектных термов:

A, A

который снова должен конвертироваться.

Но ведь это совершенно излишне!

Все об'ектные термы, составляющие этот список заведомо являются терминальными. Это следует из следующей теоремы.

Т Е О Р Е М А 5.I. О терминальности значений переменных. Значения, принимаемые переменными в процессе отождествления на каждом шаге конверсии, являются терминальными списками термов.

Опираясь на результат теоремы 5.I определим новый алгоритм вычисления таким образом, чтобы не приходилось конвертировать заведомо терминальные термы, являющиеся значениями переменных.

О П Р Е Д Е Л Е Н И Е 5.I. Правило конверсии об'ектного терма. Пусть задан об'ектный терм ⌊0 и программа P. Если ⌊0 есть аппликативный терм, то предварительно конвертируется его список термов по определению 3.7, и результат конверсии подставляется вместо прежнего списка. После чего проверяется терминальность самого терма ⌊0 в смысле приведенного определения 3.4.

При этом возможны два существенно различных варианта.

А) Если ⌊O терминален, то результатом конверсии является он сам.

В) Если ⌊O не является терминальным, то он редуцируется по правилу редукции, описанному ниже в определении 5.4.

О П Р Е Д Е Л Е Н И Е 5.2. Правило конверсии типового терма. Пусть заданы типовой терм TP и подстановка DELTA.

I. Если TP является об'ектным термом, то его конверсия происходит по правилу конверсии об'ектного терма, из определения 5.I.

2. Если TP является переменной, то вместо нее подставляется соответствующее значение из подстановки DELTA.

3. Если TP является аппликативным типовым термом, то сначала конвертируется по определению 5.3 его список типовых термов, а затем редуцируется аппликативный типовой терм TP', получившийся в результате подстановки в терм TP, вместо самого списка его типовых термов, значение конверсии этого списка. Причем конверсия TP' происходит следующим образом:

а) если указатель имени TP' есть идентификатор, то TP' является об'ектным термом, и его конверсия происходит по правилу из определения 5.I;

В) если указатель имени TP' есть переменная терма, то вместо нее подставляется соответствующее значение из подстановки DELTA. После чего получившийся аппликативный терм, являющийся об'ектным термом, конвертируется по правилу конверсии из определения 5.I.

О П Р Е Д Е Л Е Н И Е 5.3. Правило конверсии списка типовых термов. Если ⌊T является списком типовых термов, то он конвертируется по одному терму слева направо.

О П Р Е Д Е Л Е Н И Е 5.4. Правило редукции об'ектного терма. Рассмотрим первое предложение программы, левая часть которого TI отождествляется с заданным об'ектным термом L0. Правая часть этого предложения LT является списком типовых термов. Вычислим подстановку DELTA по следующей формуле:

DELTA = R(TI, L0)

Где R – функция отождествления из определения 3.3. Учитывая условия выбора предложения значение DELTA определено. Если пара <LT, DELTA> не удовлетворяет условию аппликативности, то все вычисления прекращаются, и фиксируется ситуация "нарушение условия аппликативности". В противном случае, на основании вычисленной подстановки DELTA, конвертируется LT по правилу конверсии списка типовых термов из определения 5.3.

Т Е О Р Е М А 5.2. Если вместо переменной, входящей в типовой терм TP, подставить соответствующее ей значение из подстановки DELTA, то в результате снова получится список типовых термов.

Т Е О Р Е М А 5.3. Результатом конверсии типового терма TP по правилу из определения 5.2 является список об'ектных термов.

Т Е О Р Е М А 5.4. Результатом конверсии списка типовых термов LT по правилу из определения 5.3 является список об'ектных термов.

Т Е О Р Е М А 5.5. Для любого об'ектного терма L0 и программы P результаты конверсий, проведенных по правилам из определений 3.5 и 5.1, совпадают.

Правила конверсии списка термов из определения 5.3

можно модифицировать не влияя на окончательные результаты конверсии. Например, порядок, в котором конвертируются элементы списка термов, совершенно не существенен. Возможна также одновременная потенциальным конверсия всех или части элементов списка, что является источником возможности автоматического распараллеливания вычислений, в случае реализации алгоритмов на различных многопроцессорных системах.

6. УПРАВЛЕНИЕ ЗАДЕРЖКОЙ

В разделе 5 был предложен алгоритм редукции, позволяющий избегать ненужные конверсии заведомо терминальных термов, являющихся значениями переменных. Это не единственный случай, когда в процессе вычислений производится заведомо излишняя конверсия. Оптимизация вычислений в таких случаях является достаточно важным делом, чтобы уделять этому самое пристальное внимание, – вплоть до введения в язык новых изобразительных средств.

Рассмотрим следующий фрагмент программы:

```
F(&X)  = G(&X, H(&X), K(&X,&X));
G(T, #L)  = T:
G(A, &H, &K)  = &H;
```

Здесь предполагается, что функции K и H еще где-то определены. Тогда при конверсии, например, терма F(A) сначала будут конвертироваться термы H(A) и K(A,A), а уже потом будет вычисляться функция G от результирующего списка.

Но алгоритм вычисления функции G устроен так, что вначале анализируется только первый терм аргумента, и лишь затем, в зависимости от его значения, используется та или иная часть всего остального аргумента. Так, в

первом предложении не используется значение ни второго, ни третьего термов, а во втором предложении значение третьего терма. Следовательно, конверсия этих термов, в каждом из рассматриваемых случаев, была излишней и приводила только к ненужным вычислениям. Потеря эффективности здесь вызвана неизбежностью автоматической конверсии всех термов аргумента вызова функции G. Хотя тут стоило бы сначала конвертировать только первый терм, и в дальнейшем, в зависимости от его значения, конвертировать или не конвертировать какие-то из последующих термов. При этом, вызываемая функция G должна сама управлять конверсией тех термов, значения которых будут необходимы в том или ином случае.

Введем в язык новые средства, позволяющие при описании алгоритмов управлять конверсией термов в правых частях предложений.

В тех случаях, когда нужно предотвратить автоматическую конверсию какого-либо списка типовых термов из правой части предложения, надо заключить этот список в квадратные скобки. При выполнении соответствующего шага редукции из определения 5.4 каждый типовой терм задерживаемого списка, также, как и каждый его подтерм, за исключением переменных, сначала будет заключен в аппликативный терм с именем HOLD, а затем вместо переменных будут подставлены соответствующие значения. Ни один из термов, находящихся внутри квадратных скобок не будет конвертироваться.

Так, если в нашем примере вместо прежнего описания функции F поместить следующее:

$$F(\&X) = G(\&X, [H(\&X), K(\&X,\&X)]);$$

то в результате конверсии терма F(A), будем иметь HOLD(H(A)). Здесь видно каким образом происходит задержка конверсии. Нарочный аппликативный терм с именем HOLD нужен для того, чтобы при дальнейшей работе с интенсионалами можно было отличить термы, задержанные

явно в правых частях предложений и задержанные в результате имплицированной задержки, описанной ранее.

Для того, чтобы явно задержанные интенсионалы можно было затем, в случае необходимости, конвертировать, в язык введена метафункция CONVERT. В результате обращения к ней конвертируется об'ектный терм, являющийся ее аргументом.

Заменим прежнее описание функции G в нашем примере на новое:

```
G(T, #L)  =  T;
G(A, HOLD(&H), &K)  =  CONVERT(&H);
```

Тогда в результате конверсии терма F(A) мы получим тот же результат, что и в самом первом варианте нашего примера.

Предложенный аппарат явной задержки позволяет, кроме всего прочего, эффективно описывать интерпретирующие функции, - например, основных управляющих конструкций традиционных языков программирования. Продемонстрируем это на примере интерпретирующей функции условного выражения.

Для большей выразительности расширим еще раз наш входной язык. Разрешим опускать запятые в записи списка термов, заменяя их пробелами в тех случаях, когда запятые однозначно восстанавливаются. Разрешим также в правой части предложения, при записи типового аппликативного терма, после имени терма вместо круглых скобок заключать список термов в квадратные скобки. Это в точности эквивалентно записи после имени терма круглых скобок, внутри которых весь список термов заключен в квадратные.

Теперь интерпретирующую функцию IF условного выражения можно описать следующим образом.

```
IF(&P, #L)  =  IFSEL(EVAL(&P),#L);

IFSEL(T, &THEN, &ELSE)  =  EVLIST(&THEN);
IFSEL(F, &THEN, &ELSE)  =  EVLIST(&ELSE);

EVLIST(HOLD( &THENELSE(#L) ))  =  EVAL(#L);

EVAL(HOLD( &F(#L) ))  =  &F(EVAL(#L));
EVAL(HOLD( &X ))  =  CONVEPT(&X):
EVAL(&X,#L)  =  EVAL(&X), EVAL(#L);
EVAL(&X)  =  &X;
```

Приведем описание простейшего предиката – предиката равенства.

```
EQ(&X, &X)  =  T;
EQ(&X, &Y)  =  F;
```

для демонстрации использования приведенной интерпретирующей функции IF опишем еще раз функцию факториала.

```
FACT(&N,  =  IF[EQ(&N,0)  THEN(I)
               ELSE(&N * FACT(&N - I))] ;
```

Если необходимо получить интенсиональ функции, порожденный в результате явной задержки, без об'емлющего аппликативного терма с именем HOLD, то следует к явно задержанному списку термов применить следующую функцию INTENT:

```
INTENT(HOLD( &F(#L) ))  =  INTAP(&F, INTENT(#L));
INTENT(HOLD(&X))  =  &X;
INTENT(&X,#L)  =  INTENT(&X), INTENT(#L);
INTENT(&X)  =  &X;

INTAP(&F, #L)  =  INTDLTHOLD[&F(#L)] ;
INTDLTHOLD(HOLD( &X ))  =  &X;
```

7. СКАЛЯРНЫЕ ВЫРАЖЕНИЯ

Традиционную запись произвольного алгебраического выражения, содержащего простые переменные и обращения к функциям, также будем также представлять в термовом виде. Пусть, например, дано выражение

```
x + SIN(y) - EXP(x+y) + 2*x**2
```

Мы будем интерпретировать его следующим образом. Все простые переменные есть простые об'ектные термы – интенсионалы функций без аргуметов. Все обращения к функциям вида SIN(Y) есть аппликативные об'ектные термы – интенсионалы функций с аргументами. Например:

```
x + SIN(y)  -> ADD( x, SIN(y) )
```

В некоторых случаях такое непосредственное представление может быть приемлемым. Но если, например, известно, что интенсионалы образованы в результате задержки функций, принимающих значения на множестве действительных чисел, то тогда возникают проблемы тождества выражений. Например, рассмотрим два интенсионала:

```
ADD(x, I)    и   ADD(I, x)
```

Интенсионально эти два выражения не равны, но если, например, x есть интенсионал действительной функции, то экстенсионально они равны. В процессе вычислений существенную роль играет экстенсиональное равенство выражений, и для того чтобы наши вычисления могли успешно продвигаться, необходимо как можно в большем числе случаев уметь определять экстенсиональное равенство выражений. Одним из путей решения этой проблемы является приведение всякого выражения данного типа к некоторой нормальной форме (если она существует).

Основным свойством нормальной формы является то, что интенсиональное равенство выражений является необходимым и достаточным условием их экстенсионального равенства.

Важную роль при работе алгоритма построения нормальной формы выражений играет бинарная функция "COMPARE", определенная на декартовом квадрате множества об'ектных термов и принимающая значения "меньше", "равно", "больше". Как именно устроено это отношение линейного порядка сейчас для нас не важно.

Выделим в качестве отдельного типа данных тип "скалярное выражение". К нему будем приводить выражения с интенсионалами функций, принимающих значение из некоторого кольца. Имя этого типа данных будет SCAL.

Пусть выражение P принадлежит кольцу полиномов от N переменных над полем рациональных чисел. Пользуясь функцией COMPARE выберем из множества переменных самую старшую - XI. Тогда для выражения P выберем следующую нормальную форму:

$$P = \sum_{i=1}^{n} A_i * X_1^i$$

где A_i принадлежит кольцу полиномов от оставшихся N-I переменных. В термовом виде это будет представлено следующим образом:

SCAL(<ПЕР.>, <СТЕПЕНЬ-I>, AI , ... <СТЕПЕНЬ-N>, AN)

Здесь последовательность степеней расположена по возрастанию. Например, выражение

```
I + X**2 - 3*X
```

в термов вида будет представлено следующим образом:

SCAL(X, 0, I, I, -3, 2, I).

Для выражений, содержащих аппликативные термы, нормальная форма имеет несколько более сложный вид. Пусть кольцо KI получено путем кольцевого присоединения к полю рациональных чисел всех аппликативных термов с именем F, список аргументов которых также приведен к нормальной форме. Тогда для выражения P, содержащего такие аппликативные термы, выберем следующую нормальную форму:

$$P = \sum_{i=1}^{n} A_i * \prod_{j=1}^{m_i} F(L_{ij})^{K_{ij}}$$

где A_i принадлежит полю рациональных чисел, L_{ij} – для каждого i упорядочены вместе с j по возрастанию, и все произведения из M_i аппликативных термов упорядочены вместе с i по возрастанию.

Случай, когда выражение содержит как простые переменные, так и аппликативные термы, обобщается естественным образом. Термовое представление этой нормальной формы строится аналогично представлению нормальной формы полинома. Опишем термовое представление выбранной нами нормальной формы такого скалярного выражения в виде БНФ.

<СКАЛЯРНОЕ ВЫРАЖЕНИЕ> ::= SCAL (<НОРМ. Ф. СКАЛЯРА>)

<НОРМ. Ф. СКАЛЯРА> ::= <РАЦИОНАЛЬНОЕ ЧИСЛО> |
<ИМЯ ПЕРЕМЕННОЙ> , <ЧЛЕНЫ ПОЛИНОМА> |
<ИМЯ ФУНКЦИИ> , <СУММА ПРОИЗВЕДЕНИЙ Ф-ЦИИ>

⟨ЧЛЕНЫ ПОЛИНОМА⟩ ::= ⟨ЧЛЕН ПОЛИНОМА⟩ !
⟨ЧЛЕН ПОЛИНОМА⟩ , ⟨ЧЛЕНЫ ПОЛИНОМА⟩

⟨ЧЛЕН ПОЛИНОМА⟩ ::= ⟨СТЕПЕНЬ⟩ , ⟨СКАЛЯРНОЕ ВЫРАЖЕНИЕ⟩

⟨СТЕПЕНЬ⟩ ::= ⟨РАЦИОНАЛЬНОЕ ЧИСЛО⟩

⟨СУММА ПРОИЗВЕДЕНИЙ ф-ЦИИ⟩ ::= ⟨СЛАГАЕМОЕ⟩ !
⟨СЛАГАЕМОЕ⟩ , ⟨СУММА ПРОИЗВЕДЕНИЙ ф-ЦИИ⟩

⟨СЛАГАЕМОЕ⟩ ::= M (⟨ПРОИЗВ. АППЛ. ТЕРМОВ⟩) ,
⟨КОЭФФИЦИЕНТ⟩

⟨ПРОИЗВ. АППЛ. ТЕРМОВ⟩ ::= 0 ! ⟨МНОЖИТЕЛЬ⟩ !
⟨МНОЖИТЕЛЬ⟩ , ⟨ПРОИЗВ. АППЛ. ТЕРМОВ⟩

⟨МНОЖИТЕЛЬ⟩ ::= ⟨СТЕПЕНЬ⟩ , A (⟨СПИСОК АРГУМЕНТОВ⟩)

⟨СПИСОК АРГУМЕНТОВ⟩ ::= ⟨СКАЛЯРНОЕ ВЫРАЖЕНИЕ⟩ !
⟨СКАЛЯРНОЕ ВЫРАЖЕНИЕ⟩ , ⟨СПИСОК АРГУМЕНТОВ⟩

⟨КОЭФФИЦИЕНТ⟩ ::= ⟨СКАЛЯРНОЕ ВЫРАЖЕНИЕ⟩

⟨ИМЯ ПЕРЕМЕННОЙ⟩ ::= ⟨ПРОСТОЙ ОБ'ЕКТНЫЙ ТЕРМ⟩

⟨ИМЯ ФУНКЦИИ⟩ ::= ⟨ПРОСТОЙ ОБ'ЕКТНЫЙ ТЕРМ⟩

"Степень", хотя и описывается как "рациональное число", принимает только целые положительные значения. Случай, когда произведение аппликативных термов обозначается нулем, соответствует равенству нулю степени у "множителя". Список аргументов при этом опускается, ибо он не существенен.

Все функции над скалярным типом данных должны сохранять эту нормальную форму. Поскольку мы смоделировали скалярные выражения используя только термы, то в нашем языке выразимы любые преобразования над ними.

Аналогично строятся модели для таких типов данных как дробно-рациональные выражения, матрицы, тензоры и т. П.

ЗАКЛЮЧЕНИЕ

В результате анализа предметной области "алгебраические вычисления" удалось предложить небольшой набор понятий и принципов, основываясь на которых оказалось возможным моделировать как алгебраические структуры, так и действия над ними. Разрабатывая язык мы использовали эти понятия в качестве базовых. Понятия функциональной зависимости, вызова функции и принцип задержки вычислений позволили обеспечить наличие в языке аппарата конструирования необходимых типов данных. Единственным "строительным материалом", используемым при их конструировании в нашей модели, являются интенсионалы функций. Они получаются в результате задержки вычислений функции в точке, находящейся вне области ее определения. Синтаксические об'екты, изображающие в нашем языке интенсионалы функций, - это простые и аппликативные об'ектные термы, - например, "x", "F(x, y)", "G(F(x,y),)" и т.п. Из них-то и конструируются все алгебраические структуры.

Если для конструирования таких об'ектов необходимо иметь в языке аппарат задержки вычислений, то для описания действий над ними, необходимо наличие в языке средства анализа синтаксической структуры порождаемых интенсионалов.

Термы, изображающие интенсионалы, имеют синтаксическую стуктуру деревьев, поэтому для их анализа нам достаточно уметь анализировать лишь такие структуры.

В языке имеется аппарат явной задержки, что позволяет эффективно описывать интерпретирующие функции

основных управляющих конструкций традиционных языков программирования. Их использование существенно повышает выразительность языка.

Существенной особенностью языка является то, что в нем не фиксированы никакие базовые алгебраические типы данных. Все они – даже числа и работа с ними, моделируются теми же средствами, что и при описании таких типов данных как скалярные выражения и их отношение, матрицы, кватернионы, тензоры и т. п. Никакие свойства никаких алгебраических об'ектов не являются предопределенными. Мы можем описывать, например, неассоциативные или антикоммутативные структуры и, при описании действий над ними, использовать "удобную инфиксную запись операций, делая эти операции "перегруженными". При этом анализ любых созданных отруктур может, естественно, быть сколь угодно детальным. Эти свойства языка позволяют решить задачи, сформулированные во введении.

Предлагаемый язык FLAC является функциональным языком с аппаратом рекурсивного описания функций. Ряд основных конструкций языка, – включая аппарат синтаксического отождествления, аналогичны соответствующим конструкциям языка рефал [6]. Так что, одно из направлений, развиваемое сегодня, – применение методов оптимизации программ на языке FLAC, основанных на принципах суперкомпиляции, изложенных в [7].

Первая реализация языка осуществлена на языке рефал. На ней успешно решались некоторые задачи математической физики, теории устойчивости и геометрической теории дифференциальных уравнений.

Изложенные здесь основные принципы построения языка FLAC не исчерпывают, конечно, всего круга идей и методов, связанных с использованием концепции задержанных вычислений для построения языка алгебраических вычислений. Продолжение работ естественным образом разделяется на исследования, связанные с дальнейшим развитием самого языка, и на создание на нем пакетов алго-

ритмов. Некоторые работы по второму направлению опубликованы в [8] и [9].

ЛИТЕРАТУРА

I. Глушков В.М. и др. АНАЛИТИК (алгоритмический язык для описания вычислительных процессов с использованием аналитических преобразований). - КИБЕРНЕТИКА, I97I, N3

2. Проворов Л.В., Штаркман Вс.С. АЛЬКОР: система аналитических вычислений. - М., I982, (Препринт / ИПМ АН СССР, NN 6I-62).

3. Щенков И.Б. Система SANTPA-BASIC для символьно - аналитических преобразований. В кн.: Всесоюзная конференция "Системы для аналитических преобразований в механике": тез. докл. Горький, I984., с. 47-49.

4. RAND R.N. Computer algebra in applied mathmatics: an introduction to MACSYMA. Pitman Advanced Publishing Program, Research Notes In Mathmatics, Boston, I984.

5. HEAPN A.C. REDUCE-2 USER'S MANUAL. COMPUTINC PHISICS CROUP, UNIVERSITY OF UTAH, I973.

6. Базисный рефал и его реализация на вычислительных машинах. - м., ЦНИПИАСС, I977, вып V-40.

7. TURCHIN,V.F. SUPERCOMPILEP SYSTEM BASED ON the LANGUACE REFAL. - SIGPLAN NOTICES, I979, VOL.I4, NO.2.

8. Кистлеров В.Л., Серебровский А.П. Использование языка алгебраических вычислений REDUCE в диалоговых САПР систем управления. В кн.: Диалоговые информационно-вычислительные системы: тез. докл. Иркутск, СИБИЗМИР СО АН СССР, 1986.

9. Григорьев Ф.Н., Кистлеров В.Л., Кузнецов Н.А. Применение методов аналитических вычислений на ЭВМ при автоматизации проектирования систем управления. В кн.: X всесоюзное совещание по проблемам управления: тез. докл. Т.I, М.,[ВИНИТИ], 1986, с. 277-278.

ОГЛАВЛЕНИЕ

В.Л.Кистлеров
ПРИНЦИПЫ ПОСТРОЕНИЯ ЯЗЫКА АЛГЕБРАИЧЕСКИХ ВЫЧИСЛЕНИЙ *FLAC*. Препринт

Т-04904. от 10.03.87г. Формат бумаги 60x84/16
Уч.-изд.л.2. Тираж 300. Заказ 104. Цена 20 коп.

Институт проблем управления
117806, Москва, ГСП-7, Профсоюзная, 65

Цена 20 коп.