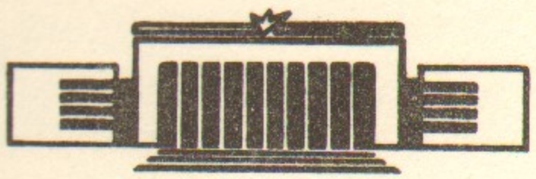
ИНСТИТУТ ЯДЕРНОЙ ФИЗИКИ СО АН СССР

А.Г.Грозин

DIRAC—
МАШИННО-НЕЗАВИСИМАЯ ПРОГРАММА ДЛЯ АЛГЕБРАИЧЕСКИХ ВЫЧИСЛЕНИЙ С ПОЛИНОМАМИ И ТЕНЗОРАМИ

ПРЕПРИНТ 83—117



НОВОСИБИРСК

Аннотация

Описана программа на языке PASCAL, производящая алгебраические операции с полиномами и тензорами. Полиномы имеют рациональные (в частности, целые) коэффициенты и могут зависеть от нескольких скаляров. Скалярам можно приписать порядки малости, и дать указание программе отбрасывать члены, суммарный порядок малости которых больше некоторого. Полиномы можно складывать и вычитать, умножать, возводить в целую степень, дифференцировать и интегрировать, подставлять полином вместо произведения степеней скаляров. Тензоры состоят из тензорных структур с полиномиальными коэффициентами. Тензорные структуры строятся из векторов с индексами, метрического тензора и единичного антисимметричного тензора. Размерность пространства может быть любой, в частности заданной аналитическим выражением. Все квадраты и скалярные произведения векторов должны быть выражены через скаляры. Тензоры можно складывать и вычитать, умножать со сверткой по повторяющимся индексам, дифференцировать по вектору. Можно подставлять тензор вместо тензорной структуры, умноженной на произведение степеней скаляров (часть индексов могут рассматриваться как формальные). Можно также дифференцировать и интегрировать тензор по скалярам, и подставлять в нем полином вместо произведения степеней скаляров. Программа доступна на Одре-1305, ЕС ЭВМ, Электронике 100-25 и совместимых с ними машинах, и легко может быть перенесена на другие типы ЭВМ.

DIRAC—
COMPUTER-INDEPENDENT PROGRAM FOR ALGEBRAIC CALCULATIONS WITH POLYNOMIALS AND TENSORS

A.G.Grozin
Institute of Nuclear Physics, Novosibirsk 630090, USSR

Abstract

A PASCAL program for algebraic operations with polynomials and tensors is described. Polynomials have rational (in particular, integer) coefficients and may depend on several scalars. Orders of smallness may be assigned to scalars, and the program will drop out terms wich have total order of smallness greater than a given maximum order. Polynomials may be added, subtracted, multiplied, raised to an integer power, differentiated and integrated. A polynomial may be substituted for a product of powers of scalars. Tensors consist of tensor structures with polynomial coefficients. Tensor structures are made of vectors with indices, metric tensor, and unit antisymmetric tensor. Space dimension may be arbitrary, in particular it may by given by an analytic expression. All squares and scalar products of vectors must be expressed through scalars. Tensors may be added, subtracted, multiplied with the contraction over repeated indices, differentiated with respect to a vector. A tensor may be substituted for a tensor structure multiplied by a product of powers of scalars (some indices may be regarded formal). Tensors may be also dfferentiated and integrated with respect to scalars. In a given tensor, a polynomial may be substituted for a product of powers of scalars. The program is available on computers compatible with ICL-1900, IBM-360, 370, PDP-11, and can be easely adapted to other types of computers.

1. Введение.

Системы аналитических вычислений на ЭВМ все более широко применяются в теоретической физике [1-2], что позволяет передать машине рутинную работу и заняться более интересными вещами. Используются такие мощные универсальные системы, как, например, REDUCE [3]. Однако при их применении возникает ряд трудностей, главной из которых является нехватка оперативной памяти. Сами эти системы чрезвычайно велики, и представление данных не является максимально компактным в силу их общности. Кроме того, эти системы доступны не на всех ЭВМ, и их использование в диалоговом режиме часто затруднительно. Между тем во многих задачах требуются вычисления с очень простыми типами выражений, однако чрезвычайно большой длины. В квантовой теории поля это в первую очередь тензоры и выражения с матрицами Дирака. Даже в случае относительно коротких вычислений возможность провести их быстро в диалоге с легко доступной ЭВМ позволяет физику-теоретику значительно эффективнее использовать свое время.

В связи с этим было решено написать простую программу для такого рода вычислений, предельно экономящую память (даже в ущерб общности), которую можно было бы использовать на любой доступной ЭВМ, в частности, в диалоговом режиме. Некоторые идеи при ее написании были заимствованы из известных систем REDUCE [3], SCHOONSCHIP [4] и CAMAL [5]. В качестве языка реализации был выбран PASCAL [6] вследствии своей широкой распространенности, простоты, эффективности, развитых структур данных и наличия встроенного механизма динамического распределения и освобождения памяти.

Программа была названа DIRAC. Краткое сообщение о ней содержится в [7]. В настоящей работе описывается ее версия 2.1. Она обрабатывает 2 типа данных: полиномы и тензоры. Полиномы имеют рациональные (в частности, целые) коэффициенты и могут зависеть от нескольких скаляров. Скалярам можно приписать порядки малости—целые неотрицательные числа, и дать указание программе отбрасывать члены, суммарный порядок малости которых больше некоторого. Тензоры состоят из тензорных структур с полиномиальными коэффициентами. Тензорные структуры строятся из векторов с индексами, метрического тензора и единичного антисимметричного тензора. Размерность пространства может

быть любой, в частности заданной аналитическим выражением. Все квадраты и скалярные произведения векторов должны быть выражены через скаляры. Количество скаляров, индексов и векторов ограничено, но эти границы легко сменить при перекомпиляции программы.

Программа организована в виде набора независимых алгебраических процедур и интерпретатора, производящего ввод-вывод и вызывающего эти процедуры. Поэтому добавление новых операций над существующими типами данных производится просто: добавляется соответствующая алгебраическая процедура, и в интерпретатор вставляется вызывающая последовательность для нее.

В следующую версию программы будет включена обработка выражений с матрицами Дирака. Обработку выражений более общего вида (рациональных, содержащих функции и т.д.) реализовать затруднительно в силу ограниченности выбранного представления данных, и такое расширение не планируется.

Для взаимодействия с пользователем разработан интерпретатор с простого входного языка, описанного в п.4. Он обеспечивает полиномиальные и тензорные переменные, их ввод и вывод, алгебраические действия над ними и ряд служебных операций. Синтаксис входного языка приведен в Приложении 1, сообщения об ошибках—в Приложении 2, особенности реализации программы на ЭВМ различных типов—в Приложении 3, и пример работы с программой—в Приложении 4.

Выделение и освобождение памяти являются дорогими операциями, и не должны производиться без крайней необходимости. Поэтому в программе DIRAC переменная может использоваться в алгебраических операциях в двух режимах: с сохранением значения и с его уничтожением. В последнем случае процедура, реализующая алгебраическую операцию, старается использовать члены своего аргумента прямо на их местах, модифицируя их содержимое и строя из них результат. Интерпретатор следит за тем, чтобы переменные, используемые в алгебраических операциях, имели значение. Чтобы не допустить генерации мусора, интерпретатор следит также за тем, чтобы значения присваивались только таким переменным, которые его не имели. Если это необходимо, значение переменной можно явно уничтожить.

2. Полиномы.

Полином представляет собой список мономов, лексикографически упорядоченных по возрастанию степеней скаляров:

```
type poly=↑monom;
     monom=record num,den:integer; next:poly;
                  power:packed array [scalar] of char
            end;
```

Набор степеней скаляров представлен упакованным литерным массивом, что ограничивает максимальную степень, но приводит к значительной экономии памяти при сохранении машинной независимости и позволяет использовать встроенные операции сравнения строк. Целые неограниченной длины не предусмотрены.

К почленным операциям над полиномами относятся копирование, копирование с обратным знаком, умножение на моном и дифференцирование. Процедура дифференцирования за один просмотр полинома берет производные заданных порядков по всем скалярам (если порядок отрицательный, это означает интеграл, взятый соответствующее число раз). В режиме уничтожения аргумента все мономы модифицируются на месте, в режиме сохранения—строится новый список результата. Некоторые мономы могут превращаться в 0—из-за превышения максимального порядка малости или потому, что порядок производной по скаляру больше его степени в данном мономе. В режиме уничтожения такие мономы исключаются из списка и освобождаются, а в режиме сохранения не переносятся в новый список.

Сложение полиномов производится по известному алгоритму слияния двух упорядоченных списков в один. А именно, вводятся 2 указателя, в начальный момент устанавливаемые на начала складываемых полиномов. Если один из них пуст, другой копируется в ответ (как описано выше). В противном случае сравниваются степени мономов, на которые они указывают. Если одна из них меньше другой, этот моном передается в результат, и соответствующий указатель продвигается. При этом, если соответствующий полином уничтожается, моном передается в результат, оставаясь на своем месте в памяти, в противном случае копируется. Если степени равны, то это—подобные члены, и их необходимо привести; при этом продвигаются оба указателя. Если сумма коэффициентов отлична от 0, в результат выдается соответствующий моном. Он строится так: если хоть один из аргументов уничтожается, новый коэффициент записывается в моном из него, и этот моном включается в результат; если уничтожается и второй аргу-

мент, моном из него освобождается; если сохраняются оба аргумента, строится новый моном. В том случае, когда сумма коэффициентов подобных членов равна 0, в результат ничего не передается; мономы из уничтожаемых аргументов освобождаются.

При умножении полиномов сначала проверяются вырожденные случаи. Если хоть один из перемножаемых полиномов равен 0, выдается нулевой результат, и второй аргумент, если надо, уничтожается. Если хоть один из полиномов состоит из одного монома, вызывается описанная выше процедура умножения полинома на моном с режимом, соответствующим другому из аргументов, и затем, если надо, уничтожается одночленный полином.

В общем случае умножение производится путем слияния упорядоченных потоков, получаемых умножением каждого монома первого полинома на весь второй. Эти потоки представлены списком пар указателей. В начале работы он выстраивается параллельно первому полиному, так что первый указатель указывает на соответствующий моном первого полинома, а второй на первый моном второго полинома. При этом первые члены сливаемых потоков упорядочены по неубыванию степеней. Поэтому сначала выдается произведение мономов, даваемое первой парой указателей (если только не превышен максимальный порядок малости). Второй указатель этой пары продвигается. Если он стал пустым, значит, этот поток иссяк, и пара исключается из списка; если при этом список потоков стал пустым, процесс окончен. Если же это не так, первая пара указателей описывает какой-то новый член, вообще говоря, не находящийся в правильном порядке по отношению к первым членам остальных потоков. Он всплывает по списку потоков, пока не займет надлежащего места, после чего все повторяется сначала.

Этот алгоритм генерирует члены результата с неубывающими степенями, однако необходимо позаботиться о приведении подобных. Для этого заводится специальный буфер для одного монома, в начале заполняемый мономом с нулевым коэффициентом и всеми степенями, равными 0. Когда алгоритм слияния генерирует очередной моном, его степень сравнивается со степенью монома, хранимого в буфере. Если она больше, значит, мономы с предыдущей степенью уже кончились. Тогда моном из буфера помещается в результат, если только у него ненулевой коэффициент, а новый моном занимает место в буфере. Если же они равны, коэффициент нового монома просто прибавляется к коэффициенту того, который лежит в буфере.

При возведении полинома в степень также сначала проверяются вырожденные случаи. Если аргумент равен 0, выдается нулевой результат (в частности, программа считает, что 0 в степени 0 равен 0). Если степень равна 0, выдается единичный полином, и аргумент, если это требуется, уничтожается. Если степень равна 1, то аргумент копируется (в нужном режиме) в результат. Если аргумент состоит из одного монома, он возводится в степень очевидным образом и, если надо, уничтожается.

В общем случае необходимо вычислить сумму по всем способам распределения степени между мономами от произведений мономов в этих степенях, умноженных на мультиномиальные коэффициенты. Это делается при помощи рекурсивной процедуры, которой поручается распределить суммарную степень по всему полиному. Если полином содержит 2 монома, генерируется упорядоченная последовательность, начиная с максимальной степени первого монома и кончая максимальной степенью второго. Если распределяемя степень равна 1, также генерируется упорядоченная последовательность. В общем случае первому моному поочередно выделяется степень от максимальной до 0, и рекурсивно вызывается процедура с заданием распределить остаток степени по остатку полинома. Если происходит превышение максимального порядка малости, соответствующие ветви перебора отрубаются.

Мультиномиальный коэффициент для распределения степени n на $n_1,n_2,...n_k$ равен $n!/n_1!\ n_2!\ ...\ n_k!$. Для его вычисления рекурсивная процедура распределения степеней имеет дополнительный аргумент, равный мультиномиальному коэффициенту в предположении, что вся переданная этой процедуре степень распределена в один моном. Когда дело доходит до выдачи готового члена в ответ, это и есть полный мультиномиальный коэффициент. В противном случае, для самого первого варианта распределения (вся степень в текущий моном) этот коэффициент именно тот, что нужен, а затем на каждом обороте цикла он пересчитывается для подправления двух факториалов. Упорядоченные последовательности мономов, полученные при работе процедуры распределения, складываются, давая результат.

Подстановка служит для замены произведения степеней скаляров (в частности, одного скаляра) на полином. Для этого просматриваются все мономы исходного полинома, и для каждого из них определяется максимальная степень этого произведения скаляров, на которую он делится, и остаток. Эта степень удаляется, и заменяется на подставляемый полином в такой степени.

Те мономы, к которым подстановка неприменима, переходят в результат без изменений. Они уже упорядочены по степеням, поэтому для повышения эффективности запоминается указатель на последний вставленный без изменений моном, и поиск места для следующего такого монома начинается с этого указателя. В начале работы этот указатель пуст, что означает, что поиск надо начинать с начала списка.

При рассмотрении очередного монома исходного полинома в первую очередь определяется максимальная степень данного произведения скаляров, на которую этот моном делится. Дальше действия различны в зависимости от того, равна эта степень 0 или нет. Если она равна 0, то этот моном надо передать в результат без изменений. Если аргумент уничтожается, моном передается на своем месте в памяти, в противном случае копируется. Поиск места для его вставления в список результата начинается с указателя на предыдущий переданный без изменения моном или с начала списка, если этот указатель пуст. Если в списке результата не найден член с теми же степенями скаляров, что вставляемый, он просто вставляется в нужное место списка, и указатель на последний переданный без изменения моном устанавливается на него. Если такой член в списке найден, к его коэффициенту прибавляется коэффициент вставляемого монома, а сам он освобождается. Если получившийся коэффициент ненулевой, указатель на последний переданный моном устанавливается на этот моном. Если же он нулевой, то этот моном исключается из списка результата, а указатель устанавливается на предыдущий моном (или, если предыдущего нет, делается пустым). Если степень получилась ненулевая, то к такому моному применяется подстановка. Для этого в первую очередь определяется моном—остаток рассматриваемого монома после удаления этой степени левой части подстановки. Если аргумент уничтожается, то исходный моном освобождается. Далее вычисляется нужная степень правой части подстановки (с сохранением аргумента), и умножается на моном-остаток (на своем месте в памяти). Построенный список сливается со списком-результатом, начиная с его начала, как при сложении полиномов с уничтожением обоих аргументов. При этом, однако, надо соблюдать осторожность: если моном из списка-результата, на который указывает указатель на последний переданный моном, сокращается, то этот указатель нужно переставить на предыдущий моном (или, если предыдущего нет, сделать пустым). В конце работы полином—правая часть подстановки, если нужно, уничтожается.

3. Тензоры

Рассматриваются тензоры в евклидовом или псевдоевклидовом пространстве произвольной размерности (она задается переменной dim: poly). Скалярное произведение двух векторов понимается обычным образом, вектора на индекс—как вектор с этим индексом, а двух индексов—как метрический тензор с этими индексами. Все скалярные произведения векторов должны быть выражены как полиномы от скаляров, для чего используется массив scapr: array [vector,vector] of poly; его элементы scapr[u,v] и scapr[v,u] оба указывают на один и тот же список-полином, представляющий скалярное произведение векторов u и v.

Тензор представляет собой лексикографически упорядоченный список тензорных структур с полиномиальными коэффициентами:

```
type tensor = ↑tensbody;
  tensbody = record polycoef:poly; next:tensor;
               tenstruct:packed array [index] of char
               end;
```

Тензорная структура представляется упакованным массивом литер из соображений экономии памяти. В позиции, соответствующей некоторому индексу, стоит 0, если этот индекс отсутствует в этой тензорной структуре, а иначе номер того индекса или вектора, на который скалярно умножен этот индекс. Таким образом, метрический тензор представляется мостиком из двух позиций в тензорной структуре, указывающих друг на друга. Первые позиции тензорной структуры отведены для единичного антисимметричного тензора. Число его индексов (eps-размерность) является параметром и может меняться во время работы. В позициях, отведенных для eps-тензора, находятся номера его индексов или векторов, с которыми он свернут; в случае индекса в соответствующей ему позиции также записан номер позиции, отведенной для eps-тензора, так что образуется мостик. Номера, записанные в этих первых позициях, упорядочены по возрастанию. Если eps-размерность равна 0, работа с единичным антисимметричным тензором невозможна.

Почленные операции над тензорами (т.е. операции, действующие на каждый полиномиальный коэффициент по отдельности), выполняются по следующей схеме. Просматривается исходный тензор. В очередном члене сначала обрабатывается полиномиальный коэффициент (в том же режиме уничтожения или сохранения, что и тензор). Далее, если аргумент уничтожается, и получившийся полином ненулевой, тензорная структура передается в

список-результат ча своем месте, и к ней подвешивается получившийся полиномиальный коэффициент (он занял место старого коэффициента, которого к этому моменту уже не существует). Если же получился нулевой полином, тензорная структура освобождается (опять, старого полинома уже нет). В том случае, когда аргумент операции сохраняется, если получился ненулевой полином, то тензорная структура копируется, к ней подвешивается этот получившийся полином, и она пристраивается к списку-результату. Если получился нулевой полином, то к результату ничего не добавляется. В обоих случаях исходный полином, подвешенный к исходной тензорной структуре, остается неизменным. Таким образом реализованы копирование тензора и копирование с обратным знаком, дифференцирование и интегрирование тензора по скалярам, полиномиальная подстановка в тензоре.

Сложение тензоров производится по тому же алгоритму двухпотокового слияния, что и полиномов. Для сложения полиномиальных коэффициентов в подобных членах вызывается процедура сложения полиномов с тем же режимом уничтожения-сохранения аргументов, и дальнейшие действия производятся в зависимости от того, нулевой или ненулевой полином получился.

Дифференцирование тензора по вектору с индексом производится почленно. Из каждого члена исходного тензора при этом получается упорядоченный список (возможно, пустой), и все они складываются, давая результат. В режиме уничтожения аргумента использованный член затем освобождается вместе со своим полиномиальным коэффициентом.

Если индекс вектора, по которому ведется дифференцирование, не участвует в рассматриваемой тензорной структуре, то просматриваются все индексные позиции в поисках нужного вектора. Каждый раз, когда он находится, создается новый элемент списка, в него копируется тензорная структура, только вместо найденного вектора с индексом в нее записывается метрический тензор, и к нему прицепляется копия (с сохранением) полиномиального коэффициента. Таким же образом, как и обычные индексные позиции, просматриваются eps-позиции, только при замене на метрический тензор индекс всплывает на свое место по eps-позициям. Если такой индекс уже есть у eps-тензора, член не генерируется. После нахождения нужного вектора в eps-позиции немедленно происходит выход из цикла, т.к. он там не может встретиться вторично.

Если индекс вектора, по которому ведется дифференцирование, свернут с другим индексом, этот метрический тензор удаляется из тензорной структуры, и заменяется индекс оператора дифференцирования, после чего все происходит точно так же.

Если индекс вектора, по которому ведется дифференцирование, свернут с вектором, все происходит подобным образом, только вместо метрического тензора в тензорную структуру генерируемого члена записываается этот вектор с индексом. При этом, в зависимости от того, меньше или больше номер этого вектора, чем того, по которому производится дифференцирование, список строится с начала или с конца, т.к. при этом он оказывается правильно упорядоченным.

Если индекс вектора, по которому ведется дифференцирование, свернут с этим же вектором, тензорная структура копируется, в эту индексную позицию записывается 0, и к ней подвешивается полиномиальный коэффициент, равный произведению исходного полиномиального коэффициента (с сохранением) на сумму размерности пространства (с сохранением) и числа позиций, в которых находится вектор, по которому производится дифференцирование.

Наконец, если индекс вектора, по которому ведется дифференцирование, свернут с eps-тензором, то просматриваются индексные позиции. Каждый раз, когда находится нужный вектор, генерируется новый элемент, в него копируется тензорная структура, в соответствующую eps-позицию записывается индекс и всплывает на свое место (если у eps-тензора уже есть такой индекс, новый элемент не генерируется), а в индексную позицию записывается 0. К созданному элементу прицепляется копия (с сохранением) полиномиального коэффициента.

При умножении тензоров, если хоть один из аргументов равен 0, выдается нулевой результат, и второй тензор, если надо, уничтожается. В общем случае, произведения членов тензоров по очереди строятся в двух вложенных циклах (внешний по первому тензору, внутренний—по второму), и все они складываются, давая результат. В конце каждого оборота внешнего цикла, если первый тензор уничтожается, освобождается использованный из него член. Аналогично, в конце каждого оборота внутреннего цикла, если необработанная часть первого тензора содержит 1 член, а второй тензор уничтожается, освобождается использованный член второго тензора, т.к. в этом случае он больше не понадобится.

Произведение одного члена из первого тензора на один член второго тензора строится так. Их индексные позиции просматриваются слева направо, пока не найдется такая, которая используется хоть в одной из тензорных структур. Далее из такой позиции мы начинаем двигаться, чередуя переходы по записанному в данной позиции указателю с перепрыгиванием на другую тензорную структуру. Этот путь может кончиться по трем причинам: либо мы придем туда, откуда вышли, либо выйдем за пределы индексных позиций, встретив не индекс, а вектор или указатель в единичный антисимметричный тензор, либо при прыжке обнаружим, что попали в пустую позицию. В первом случае встретилась замкнутая петля из метрических тензоров, и счетчик замкнутых петель увеличивается на 1. Иначе надо от исходной точки пойти в другую сторону и найти другой конец этой цепочки. Производя описанные перемещения, мы заметаем за собой следы, т.е. записываем нули в пройденные позиции, чтобы не наткнуться вторично на ту же цепочку при дальнейшем просмотре.

Если оба конца цепочки находятся в одном и том же антисимметричном тензоре, произведение равно 0, а если в разных—счетчик числа индексов, по которым свернуты эти тензоры, увеличивается на 1, и в них вместо этих индексов заносятся нули (на свое место). Если один конец цепочки находится в антисимметричном тензоре, а на другом висит индекс или вектор, он вставляется на свое место в этот тензор, а если там уже был такой же—произведение равно 0. Если на обоих концах висят индексы, в результирующую тензорную структуру входит метрический тензор, и в ней необходимо построить соответствующий мостик. Если на одном конце висит индекс, а на другом вектор, в нее входит вектор с индексом, т.е. в позицию этого индекса нужно записать этот вектор. Если на обоих концах висят векторы, получившееся скалярное произведение запоминается в специальном списке. Если оно равно 0, выдается нулевой результат.

Вычисление полиномиального коэффициента начинается с возведения размерности пространства в степень, равную числу встреченных петель, затем эта степень домножается на запомненные скалярные произведения (их список при этом уничтожается) и на полиномиальные коэффициенты от исходных тензорных структур. В тех случаях, про которые говорилось, что выдается нулевой результат, никакие действия с полиномами не производятся, список запомненных скалярных произведений уничтожается, и происходит выход на конец процедуры по goto.

Если исходные тензорные структуры не содержали единичного антисимметричного тензора, то просто создается один новый член с готовой тензорной структурой и полиномиальным коэффициентом. Если его содержала одна из исходных структур, он предварительно записывается в результирующую структуру (в него уже вставлены на нужные места индексы и векторы с концов цепочек, которые в нем начинались).

Если же его содержали обе исходные структуры, их произведение раскрывается через скалярные произведения входящих в них индексов и векторов, и результат представляет собой упорядоченный список членов. Для его построения используется рекурсивная процедура, которая свертывает наименьший из имеющихся индексов или векторов по очереди со всеми из второго антисимметричного тензора. Если наименьшим является индекс, уровень рекурсии считается индексным, а если вектор—векторным. Самый внешний уровень, с которого вызывается процедура, считается индексным, а самый внутренний, когда не осталось неиспользованных позиций—векторным. При переходе с индексного уровня на векторный происходит подготовка к накоплению полиномиального коэффициента, а при обратном переходе—накопленный полиномиальный коэффициент умножается на коэффициенты от исходных структур, и если полученный коэффициент ненулевой, из него и текущей тензорной структуры строится новый член. На индексном уровне при переборе индексов и векторов дополняется тензорная структура, и каждый раз рекурсивно вызывается процедура для дальнейшего перебора. На векторном уровне при переборе векторов каждый раз рекурсивно вызывается процедура для дальнейшего перебора с полиномиальным аргументом, домноженным (с сохранением) на текущее скалярное произведение; после окончания перебора на векторном уровне полиномиальный аргумент уничтожается. Когда не остается неиспользованных позиций, текущий полиномиальный аргумент прибавляется к накапливаемому полиномиальному коэффициенту (с нужным знаком).

Тензорная подстановка служит для замены произведения тензорной структуры и степеней скаляров на произвольный тензор. Часть индексов в левой части подстановки могут рассматриваться как формальные, и подстановка будет производиться при любых их фактических значениях (разумеется, те же фактические значения будут подставлены вместо формальных в правую часть). Если в левой части подстановки стоит единичный антисимметричный

тензор, порядок индексов и векторов в нем безразличен—соответствие образцу будет обнаружено в любом случае, с учетом замен формальных индексов на фактические. В единичном антисимметричном тензоре на месте формального индекса может стоять вектор. В отличие от полиномиальной подстановки, тензорная может применяться к данному члену не более одного раза, т.к. тензорная структура не может встречаться дважды.

Члены исходного тензора просматриваются по очереди. Для каждого из них определяется, содержит ли его тензорная структура подструктуру из левой части подстановки (с возможными заменами формальных индексов). Если да, то его полиномиальный коэффициент расщепляется (в том же режиме уничтожения-сохранения, что и тензор) на 2 полинома, мономы которых соответственно содержат и не содержат произведение степеней скаляров из левой части подстановки (из первого из них удаляется это произведение). Если полином, содержащий его, ненулевой, то подстановка применяется, т.е. строится тензор, равный правой части подстановки (с возможными заменами формальных индексов), домноженной на остаток тензорной структуры и на этот полином. Этот тензор сливается с результирующим тензором, начиная с его начала, подобно тому, как это было описано для полиномиальной подстановки. Второй из продуктов расщепления полинома вместе с исходной тензорной структурой образуют член, к которому подстановка неприменима. Если же с самого начала не найдена тензорная подструктура из левой части подстановки, то таким членом будет весь член исходного тензора. В любом случае, этот член (если только у него не нулевой полиномиальный коэффициент) вставляется в результат, начиная с указателя на предыдущую тензорную структуру, переданную без изменений, аналогично случаю полиномиальной подстановки.

Сопоставление старой тензорной структуры с образцом—левой частью подстановки производится так. Просмативаются все позиции структуры. Каждая из них либо связывается с подходящей позицией образца, либо нет. В первом случае делается пометка, что эта позиция в новой тензорной структуре будет заполняться из правой части подстановки, а во втором—из старой структуры. Такое связывание может произойти не только с той же позицией образца, но и с какой-либо позицией, соответствующей формальному индексу. В этом случае в таблице перекодировки отмечается, какому фактическому индексу он соответствует.

Если eps-размерность положительна, сначала сравниваются eps-позиции. Если в образце они пусты, они будут переноситься из старой структуры; если в образце они не пусты, а в старой структуре пусты, сопоставление не удалось. Если они не пусты в обоих структурах, они будут переноситься из правой части. В этом случае просматриваются eps-позиции старой структуры. Заводятся 2 указателя, один из которых продвигается по eps-позициям образца, связанным с формальными индексами, а второй—по остальным. Если позиции второго типа исчерпаны, либо в первой из них содержится вектор или индекс, больший, чем в текущей позиции старой структуры, то формальный индекс насыщается вектором или индексом из текущей позиции, и первый указатель продвигается. В противном случае в первой позиции второго типа должен находиться тот же вектор или индекс, что и в текущей позиции старой структуры, и продвигается второй указатель.

Далее просматриваются индексные позиции старой структуры. Если в текущей позиции 0, позиция образца не должна требовать соответствия, т.е. должна содержать 0 или быть позицией формального индекса. Если в текущей позиции вектор, а позиция образца не требует соответствия, делается попытка насытить этой позицией ненасыщенный формальный индекс с тем же вектором. При удаче он связывается с текущим индексом, а при неудаче текущая позиция будет переноситься в новую структуру из старой. Если позиция образца требует соответствия, оно должно быть для успеха сопоставления, а текущая позиция будет переноситься из правой части.

Наконец, если в текущей позиции находится больший индекс, ищется соответствие для метрического тензора (в случае меньшего индекса оно уже проверено ранее). Если обе позиции образца не требуют соответствия, делается попытка насытить 2 ненасыщенных формальных индекса, образующих метрический тензор. При удаче они связываются с двумя рассматриваемыми индексами, а при неудаче эти две позиции будут переноситься в новую структуру из старой. Если одна из позиций образца требует соответствия, а другая нет, то первая из них должна указывать на формальный индекс, который и насыщается. Если обе позиции образца требуют соответствия, оно должно быть для успеха сопоставления, а две рассматриваемые позиции будут переноситься в новую структуру из старой.

После окончания просмотра сопоставление является удачным, если не осталось ненасыщенных формальных индексов.

Применение подстановки производится так. Для каждого члена из правой части подстановки производится построение новой тензорной структуры путем перекодировки формальных индексов в фактические. При этом eps-позиции упорядочиваются. Формальному индексу может соответствовать фактический вектор (при сопоставлении eps-позиций). Поэтому при перекодировке могут возникать скалярные произведения двух векторов, на которые домножается полиномиальный коэффициент. Те индексные позиции, которые должны заполняться из старой структуры, переносятся без изменений. Полиномиальный коэффициент текущего члена правой части домножается на тот из продуктов расщепления, к которому применена подстановка (с сохранением). Полученный таким образом тензорный член в общем случае встраивается в список, начиная с его начала; в частном случае, если перекодировки не было (например, подстановка без формальных индексов), члены генерируются сразу в правильном порядке. После окончания просмотра правой части продукт расщепления уничтожается.

4. Входной язык

Программа на входном языке состоит из идентификаторов, чисел и специальных символов

\+ — * / ↑ = : , ; . ? @ ( ) [ ]

которые обобщенно будем называть лексемами. Идентификатор начинается с буквы и состоит из букв и цифр. Слишком длинные идентификаторы обрезаются до некоторой длины (которая является машинно-зависимой). Идентификаторы служат именами команд, функций, флагов, скаляров, индексов, векторов, полиномиальных и тензорных переменных. Число представляет собой последовательность цифр. Если число превышает максимально допустимое в данной машине целое, при его вводе произойдет переполнение. Кроме лексем, в программу могут входить тексты—конструкции ⟨ограничитель⟩ ⟨последовательность символов⟩ ⟨ограничитель⟩ где ограничитель—любой символ, отличный от пробела (в частности, переход на новую строку), а последовательность символов не содержит ограничителя. Если ограничитель не является переходом на новую строку, текст может располагаться на нескольких строках, а если является—занимает ровно одну строку. В текстах и в качестве ограничителей можно использовать любые доступные символы.

Между лексемами можно вставлять любое количество пробелов и переходов на новую строку, что не меняет смысла программы. Между соседними идентификаторами, числами или идентификатором и числом обязательно должен быть хоть один пробел или переход на новую строку. Они не могут располагаться внутри идентификатора или числа, а внутри текста или в качестве ограничителя являются значащими символами.

Программа представляет собой последовательность команд, которые вводятся и выполняются в порядке поступления. Какие-либо специальные символы, заканчивающие или разделяющие команды, отсутствуют, команды просто записываются одна за другой. Команды делятся на описания, команды установки параметров, команды-действия и присваивания.

Команды описания scalar, index, vector, poly, tensor служат для описания идентификаторов скаляров, индексов, векторов, полиномиальных переменных и тензорных переменных соответственно. Идентификаторы скаляров, индексов и векторов должны быть все отличными друг от друга. Идентификаторы полиномиальных и тензорных переменных должны быть отличными друг от друга и от идентификаторов команд и функций. Команда scalar служит также для задания порядков малости скаляров.

Максимальное количество идентификаторов каждого типа фиксировано в течение сеанса работы с программой. Фактически для каждого типа имеется таблица идентификаторов, состояние которой меняется соответствующей командой описания. В начале работы с программой таблицы пусты.

За идентификатором команды описания следует список элементов описания через ',' (возможно, пустой), оканчивающийся ';' или '?'. В последнем случае после окончания работы команды выводится список всех имеющихся идентификаторов данного типа (для скаляров—с порядками малости) в формате команды описания, оканчивающейся ';'.

Элементы описания действуют на последовательные элементы таблицы идентификаторов, начиная с первого. Элемент описания может начинаться с конструкции

/ ⟨стар.имя⟩ /

что означает требование сохранить в прежнем виде все элементы таблицы идентификаторов от текущего до того, который совпадает с заданным идентификатором. Далее в элементе описания может

следовать ⟨нов.имя⟩, которое в этом случае записывается в текущий элемент таблицы идентификаторов. В случае описания scalar далее может следовать конструкция

: ⟨число⟩

В этом случае порядок малости скаляра, соответствующего текущему элементу таблицы скаляров, устанавливается равным заданному числу. Элемент описания может быть, в частности, пустым, что означает пропуск текущего элемента таблицы идентификаторов.

Рассмотрим ряд примеров.

Описать скаляры x, y (с порядком малости 1) и z:

```
scalar x,y:1,z;
```

Описать полиномиальные переменные a, в, c:

```
poly a,в,c;
```

Узнать, какие скаляры известны программе:

```
scalar ?
```

Изменить порядок малости скаляра z, сделав его равным 2:

```
scalar /z/:2;
```

Вставить полиномиальную переменную d после c:

```
poly /c/,d;
```

Переименовать полиномиальную переменную c в c1:

```
poly /c/c1;
```

Мономы служат для построения полиномов и тензоров, а также используются в функциях dif и sub. Моном представляет собой последовательность (1 или более) факторов, разделенных символом '*'. После фактора могут следовать (0 или более) делителей вида

/ ⟨число⟩

Существуют несколько типов факторов: число, степень скаляра, скалярное произведение и конструкция ⟨eps⟩. Степень скаляра—это скаляр, за которым, возможно, следует символ '↑' и число, перед которым может стоять знак. Скалярное произведение—это вектор или индекс, символ '.' и еще один вектор или индекс. Конструкция ⟨eps⟩—это список векторов или индексов через запятую, заключенный в квадратные скобки.

Полином представляет собой последовательность мономов, разделенных знаками '+' или '-'. Перед первым мономом также может стоять знак. Мономы в полиноме не могут содержать скалярных произведений и конструкций ⟨eps⟩. Тензор представляет собой последовательность тензорных членов, разделенных знаками '+' или '-'. Перед первым тензорным членом также может стоять

знак. Тензорный член—это моном, в котором вместо первого фактора может стоять полином в скобках. В принципе, разные члены в одном тензоре могут иметь разный набор индексов, но использование таких тензоров не рекомендуется.

Делители должны быть отличны от 0. Если некоторый скаляр входит в моном несколько раз, его степени складываются. Суммарная степень каждого скаляра должна быть неотрицательной (за исключением мономов в функции dif). Примеры мономов, которые могут входить в полиномы: x/3, 2/3*x↑2*y, 5/6*x*y↑2*2*7*x↑-2*z↑3/4/5*x↑+4.

Для тензоров, скалярное произведение вектора на вектор интерпретируется обычным образом, вектора на индекс (и наоборот)—как вектор с индексом, а индекса на индекс—как метрический тензор с индексами. Конструкция ⟨eps⟩ со всеми индексами представляет единичный антисимметричный тензор; если вместо некоторых индексов стоят векторы, они считаются свернутыми по соответствующим индексам с этим тензором. В случае псевдоевклидова пространства с отрицательной сигнатурой ⟨eps⟩ фактически представляет i* единичный антисимметричный тензор.

Скалярные произведения в тензорном мономе должны содержать хоть один индекс. Каждый индекс может встречаться не более одного раза. Конструкция ⟨eps⟩ разрешена только в том случае, если командой eps была установлена ненулевая eps-размерность. Количество векторов и индексов в ⟨eps⟩ должно быть равно этой размерности; эта конструкция может встречаться в мономе не более одного раза. Примеры тензорных мономов: p.m*n.l, x*p.m/3*q.n*5*y↑2, 2/3*x↑2*m.n*[p,q,r,s]*q.l .

Мономы в функциях dif и sub не могут содержать чисел в числителе и знаменателе. В случае функции dif и sub без списка индексов они не могут также содержать скалярных произведений и конструкций ⟨eps⟩. В единственном случае функции dif суммарные степени скаляров в мономе могут быть отрицательными.

Команда

order ⟨число⟩

устанавливает максимальный порядок малости членов, удерживаемых при вычислениях, равным указанному числу. Команда

order ?

выводит текущий порядок малости в формате команды order ⟨число⟩. Начальное значение максимального порядка малости

равно 0, и если есть скаляры с ненулевым порядком малости, все содержащие их члены будут выбрасываться. Поэтому для работы с малыми скалярами эту команду обязательно нужно выполнить в начале работы с программой.

Команда

dim (⟨полином⟩)

устанавливает размерность пространства равной указанному полиному (старое значение размерности предварительно уничтожается). Команда

dim ?

выводит текущую размерность пространства в формате команды dim (⟨полином⟩). Начальное значение размерности равно 0, что приводит к несколько неожиданным последствиям при вычислениях с тензорами (например, произведение любых двух тензоров в этом случае равно 0). Поэтому для работы с тензорами эту команду обязательно нужно выполнить в начале работы с программой.

Команда

eps ⟨число⟩

устанавливает eps-размерность равной указанному числу. Команда

eps ?

выводит текущую eps-размерность в формате команды eps ⟨число⟩. eps-размерность—это число индексов единичного антисимметричного тензора. Вообще говоря, она не обязана совпадать с обычной размерностью. Обычная размерность пространства может быть нецелой, в частности задаваться полиномом, а eps-размерность—целое неотрицательное число. Начальное значение eps-размерности равно 0, при этом работа с единичным антисимметричным тензором невозможна. В силу используемого представления тензоров, первые элементы таблицы индексов в количестве, равном eps-размерности, не могут использоваться как индексы (количество доступных индексов при этом, очевидно, уменьшается). Поэтому, когда единичный антисимметричный тензор не используется, лучше работать в состоянии eps 0. Кроме того, команду eps лучше выполнять до описания index, т.к. если ее выполнить после, несколько первых описанных индексов исчезнут (или, если eps-размерность уменьшилась, перед ними появится несколько неописанных индексов).

Команда

(⟨скал.произ.⟩ = ⟨полином⟩)

устанавливает скалярное произведение векторов (в частности, квадрат вектора) равным заданному полиному (старое значение скалярного произведения предварительно уничтожается). Команда

(⟨скал.произ.⟩)

выводит текущее значение скалярного произведения в формате команды (⟨скал.произ.⟩ = ⟨полином⟩). В скалярное произведение не должны входить индексы. Начальные значения всех скалярных произведений (включая квадраты) равны 0, поэтому для работы с тензорами необходимо задать все ненулевые скалярные произведения в начале работы с программой.

Работа программы зависит от состояния флагов, которые могут быть включены или выключены.

Флаг line (по умолчанию включен) требует начинать каждую команду с новой строки. Текст от конца команды до ближайшего конца строки пропускается. Если этот флаг выключен, можно размещать по нескольку команд в строке.

Флаг echo (по умолчанию выключен) вызывает копирование вводимой программы на вывод с нумерацией строк. Если он включен, команда text рассматривается как com.

Флаг write (по умолчанию выключен) вызывает печать вычисленного значения после выполнения каждого присваивания. Печать производится в формате присваивания с явным значением в правой части.

Флаг dialog (состояние по умолчанию машинно-зависимо) влияет на обработку ошибок, см. Приложение 2.

Флаг можно включить командой

\+ ⟨флаг⟩

и выключить командой

— ⟨флаг⟩

Программа нумерует вводимые строки, начиная с 1. Номера строк используются в распечатке программы при + echo, а также в сообщениях об ошибке. Команда

⟨число⟩

устанавливает номер строки, на которой она появилась, равным этому числу.

Команда

end

служит для нормального окончания работы программы.

Команда

halt

вызывает прерывание работы программы и выход на уровень операционной системы с сохранением возможности продолжить счет с того места, в котором произошло прерывание. Точный ее эффект является машинно-зависимым. Она может быть не реализована на некоторых машинах. Основное назначение команды halt—дать возможность изменить направление ввода или вывода средствами операционной системы во время работы программы.

Команда

text ⟨текст⟩

производит вывод текста (без окружающих его ограничителей). Она особенно полезна при выводе в файл, а также при выполнении программы из файла с выводом на терминал.

Команда

com ⟨текст⟩

не производит никаких действий и служит для помещения комментария. Она особенно полезна в программах, записанных в файл для многоразового использования.

В алгебраических операциях, а также командах delete и write, используются аргументы. Аргумент—это полиномиальная или тензорная переменная, перед которой, возможно, стоит символ '@'. Если он присутствует, значение аргумента сохраяется, в противном случае уничтожается.

Команда

delete ⟨арг.⟩

уничтожает значение полиномиальной или тензорной переменной, если только не задан режим сохранения аргумента (в последнем случае она ничего не делает).

Команда

write ⟨арг.⟩

выводит значение полиномиальной или тензорной переменной. Значение сохраняется, если задан знак '@', и уничтожается в противном случае. Вывод производится в формате, допускающем последующий ввод.

Существуют несколько типов присваиваний: числа, явного выражения, операции, функции. Присваивание вида

⟨переменная⟩ = ⟨число⟩

в случае полиномиальной переменной присваивает постоянный полином, равный этому числу, а в случае тензорной переменной—тензор без индексов, равный этому числу. Такая форма присваивания особо рекомендуется для присваивания нулевого значения. Следует подчеркнуть, что присваивание a=1 разрешено, а a=1/2—нет, и при -line последнее присваивание даст переменной a значение 1 без выдачи сообщения об ошибке, т.к. /2 будет рассматриваться как комментарий.

Присваивание явного значения—это

⟨переменная⟩ = (⟨явное значение⟩),

где явное значение—это полином или тензор. Значение должно согласовываться по типу с переменной в левой части присваивания (напомним, что допустимы тензоры без индексов, которые могут выглядеть так же, как полиномы).

Присваивание с унарной операцией—это

⟨переменная⟩ = ⟨ун.оп.⟩ ⟨арг.⟩,

где ⟨ун.оп.⟩ означает одну из двух унарных операций '+' и '—'. Они служат для копирования (возможно, с изменением знака) полиномов и тензоров. Если аргумент уничтожается, фактически копирования значения в памяти не происходит. Аргумент в правой части может быть любого типа. Присваивание тензорной переменной полиномиального значения преобразует полином в тензор без индексов. Присваивание полиномиальной переменной тензорного значения работает так: если тензор равен 0, присваивается нулевой полином, а если нет—полиномиальный коэффициент при первой тензорной структуре. Эту возможность рекомендуется использовать в том случае, когда в результате тензорных вычислений получился тензор без индексов, и желательно скопировать его значение в полиномиальную переменную.

Присваивание с бинарной операцией—это

⟨переменная⟩ = ⟨арг.1⟩ ⟨бин.оп.⟩ ⟨арг.2⟩

где ⟨бин.оп.⟩ означает одну из двух бинарных операций '+' и '*'. Они служат для сложения и умножения полиномов и тензоров. Сложение производится эффективнее, если один или оба аргумента уничтожаются. Оба аргумента должны соответствовать по типу переменной в левой части присваивания.

Для полиномиальной переменной и аргумента имеется также операция возведения в целую степень:

⟨переменная⟩ = ⟨арг.⟩ ↑ ⟨число⟩

Присваивание с функцией дифференцирования—это

⟨переменная⟩ = dif ⟨моном⟩ : ⟨арг.⟩

или

⟨переменная⟩ = dif ⟨скал.произ.⟩ : ⟨арг.⟩

В первом случае переменная и аргумент могут быть оба полиномиальными или тензорными. Аргумент дифференцируется по скалярам, указанным в мономе, порядок производной по каждому скаляру равен его суммарной степени. Суммарная степень скаляра в этом случае может быть отрицательной, это означает неопределенный интеграл, взятый соответствующее число раз. Например,

a=dif x:a

дифференцирует a по x;

a=dif x*y↑2:a

берет первую производную по x и вторую по y;

a=dif x↑-1:a

интегрирует a по x.

Вторая форма функции дифференцирования требует, чтобы переменная и аргумент были тензорного типа. В скалярном произведении должен быть ровно 1 вектор и 1 индекс. Это дифференцирование тензора по вектору с индексом. Индекс может совпадать с одним из индексов, используемых в тензоре, в этом случае производится свертка.

Присваивание с функцией подстановки -это

⟨переменная⟩ = sub ⟨моном⟩ = ⟨арг.2⟩ : ⟨арг.1⟩

или

⟨переменная⟩ = sub ⟨спис.инд.⟩ : ⟨моном⟩
= ⟨арг.2⟩ : ⟨арг.1⟩

В первом случае аргумент 2 должен быть полиномиальной переменной; переменная и аргумент 1 могут быть оба полиномиального или тензорного типа. Это—полиномиальная подстановка: вместо произведения степеней скаляров в левой части подставляется значение полиномиальной переменной. Заменяется максимальная степень левой части, которую удается выделить в члене аргумента подстановки.

Во втором случае переменная и оба аргумента должны быть тензорного типа. Список индексов (через ',', может быть пустым) перечисляет формальные индексы. Вместо тензорной структуры, умноженной на произведение степеней скаляров, которая задана в левой части (с учетом возможного переименования формальных индексов) подставляется значение тензорной переменной (в котором произведено такое же переименование). Если в тензорной левой части есть конструкция eps с произвольными индексами, на их месте могут быть не только индексы, но и векторы.

Все индексы, перечисленные в списке формальных индексов, должны присутствовать в левой части, в противном случае подстановка не применяется. В нормальном случае набор индексов в правой и левой части подстановки должен совпадать. Если в правой части отсутствует часть индексов, имеющихся в левой, в результате может получиться тензор, разные члены которого имеют разный набор индексов. Если в правой части имеются индексы, отсутствующие в левой, это может привести к ошибке.

Длина строки вывода ограничена, границу легко сменить при перекомпиляции программы. Программа начинает свою работу с того, что представляется: выводит строку

dirac version 2.1

Далее программа может выводить информацию несколких типов: сообщения об ошибках; вывод команды text; вывод команды write; вывод копий входных строк при включенном флаге echo; вывод результатов присваиваний при включеном флаге write; и вывод информационных команд

order ? dim ? eps ? (⟨скал.произ.⟩)

и описаний, оканчивающихся символом '?'. Вывод команды write производится в соответствии с синтаксисом полиномов и тензоров.

Перед началом диалоговой работы с программой бывает полезно ввести файл, в котором заданы описания и команды установки для дальнейшей работы. Во время работы может потребоваться ввести файл, имеющий смысл подпрограммы. Передача информации подпрограмме и возврат результатов ее работы производятся в полиномиальных и тензорных переменных. Каждый такой внешний файл должен заканчиваться командой halt, чтобы после него изменить источник ввода. Ввод файла с подпрограммой производится следующим образом:

```
halt
сообщение ОС о прерывании
команда ОС: подключение ввода к файлу с подпрограммой
команда ОС: продолжение счета
сообщение ОС о прерывании
команда ОС: подключение ввода к терминалу
команда ОС: продолжение счета
com 'продолжение диалоговой работы с программой'
```

Файл с подпрограммой или командами начальной установки может быть создан стандартными средствами ОС, однако удобно создать его при работе с программой. Для этого служат информационные команды

scalar ? poly ? index ? vector ? tensor ?
order ? dim ? eps ? (⟨вектор⟩.⟨вектор⟩)

и присваивания в режиме +write. Файл начальной загрузки может быть создан так:

```
halt
сообщение ОС о прерывании
команда ОС: подключение вывода к файлу
команда ОС: продолжение счета
text ''
scalar ?
poly ?
order ?
vector ?
dim ?
eps ?
index ?
tensor ?
com 'скалярные произведения векторов p, q...'
(p.p)
(p.q)
(q.q)
com 'и т.д.'
+write
com 'сохранить полиномиальные переменные a,в...
и тензорные переменные c,d...'
a=+a
b=+b
```



```
com 'и т.д.'
c=+c
d=+d
com 'и т.д.'
text 'halt'
halt
сообщение ОС о прерывании
команда ОС: подключение вывода к терминалу
команда ОС: продолжение счета
com 'продолжение диалоговой работы с программой'
```

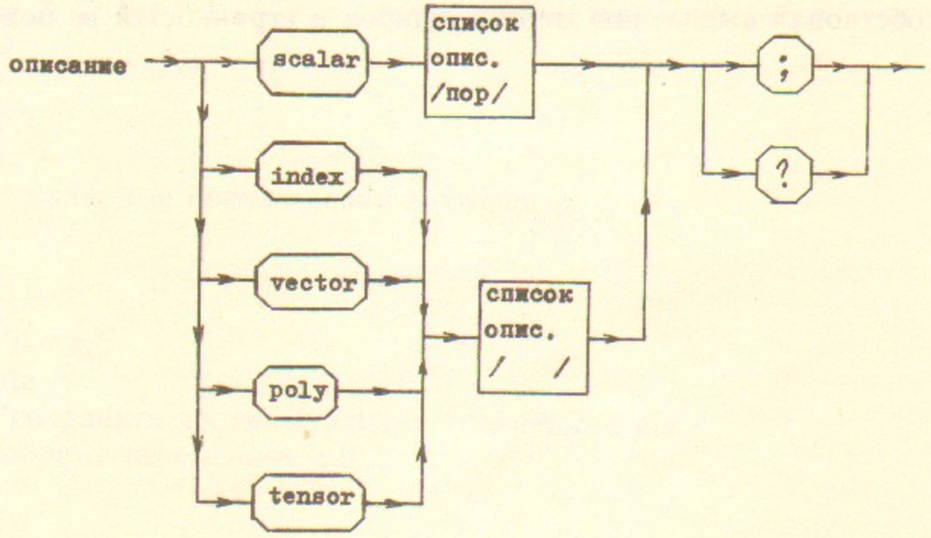
Таким же образом можно сохранить промежуточные результаты для дальнейшего использования. При этом не обязательно полностью сохранять контекст описаний и установок, достаточно выполнить при +write присваивания типа a=+a и т.д.

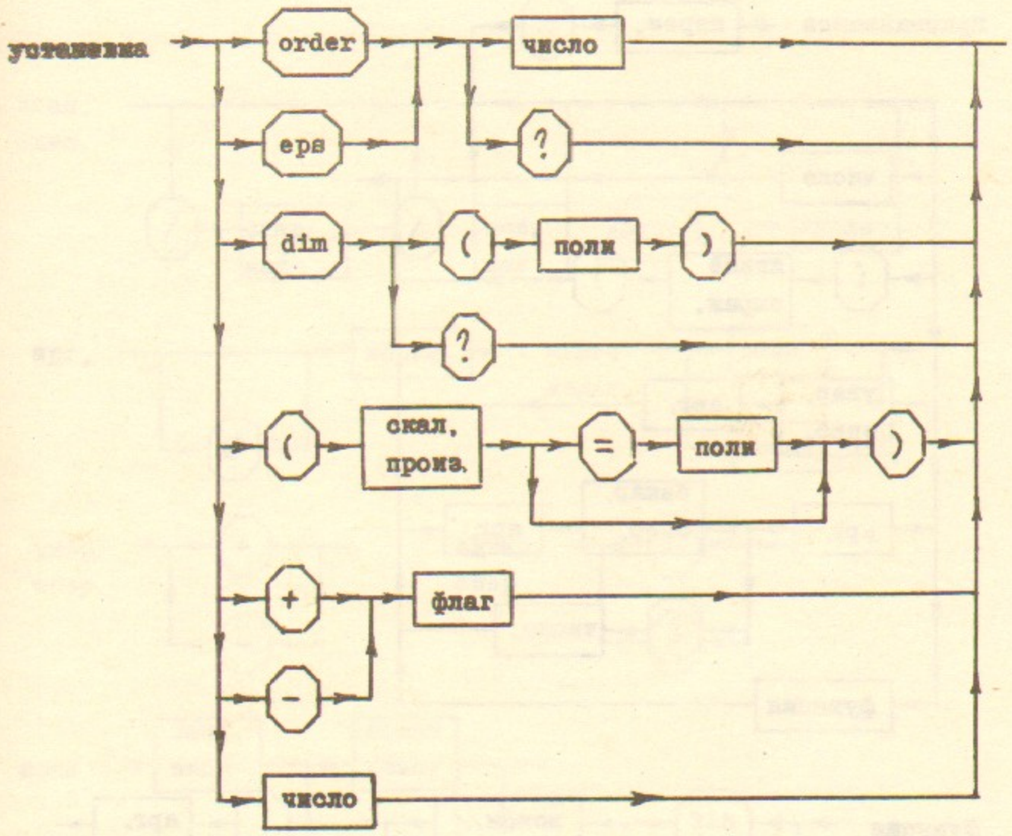
5. Заключение

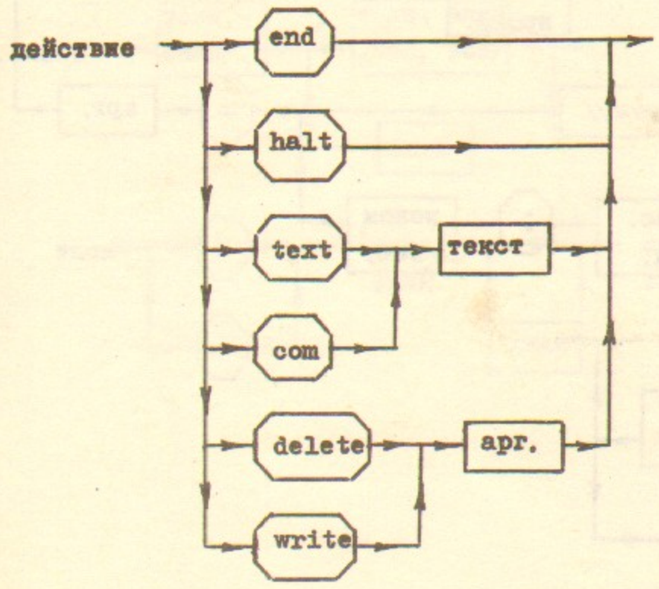
Мне приятно поблагодарить С.Д.Белова за полезные консультации в период отладки программы на Одре; Н.С.Дворникова и Б.Л.Сысолетина—за работу по постановке компилятора PASCAL на ЕС ЭВМ и консультации при переносе; П.Л.Храпкина и М.В.Ясенева—за аналогичную работу на Электронике 100-25 и СМ-4. Я благодарен также Ю.Н.Кафиеву—первому (кроме автора) пользователю программы, который, не имея опыта работы с ЭВМ, использовал программу для решения физической задачи, и способствовал выявлению многих ошибок и странностей ее поведения.

Приложение 1. Синтаксис входного языка

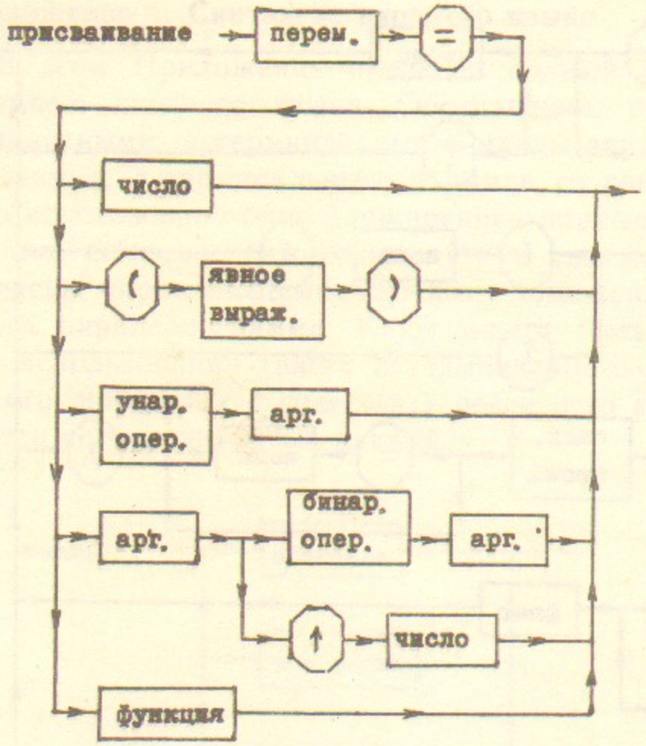
В этом Приложении приведен полный набор синтаксических диаграмм входного языка. Обозначения, в основном, являются стандартными: нетерминальные символы заключены в прямоугольные рамки, а терминальные—в рамки со срезанными углами. Однако использовано одно расширение—параметризованные нетерминальные символы. Некоторые пути в синтаксических диаграммах помечены управляющими ключами, помещенными в рамку в виде косого параллелограмма. Ключ может быть открыт или закрыт. При использовании такого нетерминального символа (или любого другого, который его содержит) после него в косых скобках помещается список открытых ключей.



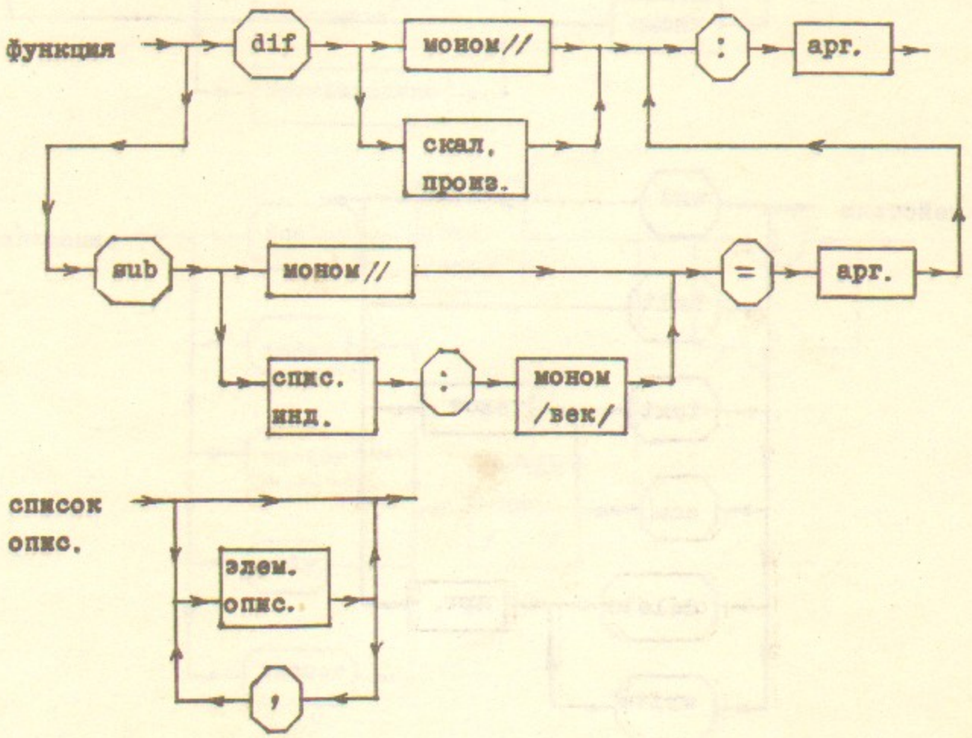




присваивание

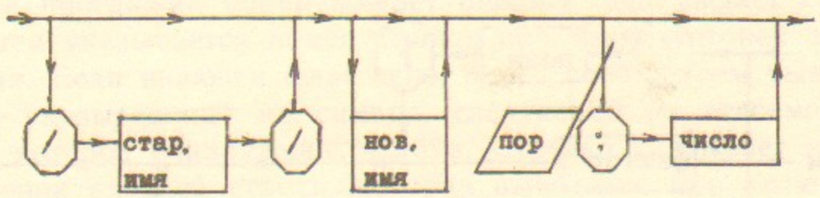


функция

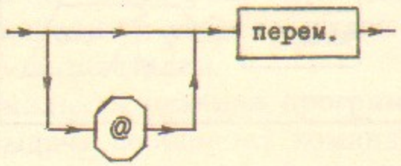


список
опис.

элем.
опис.

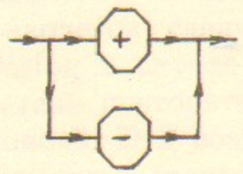


арг.

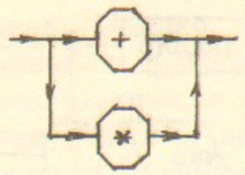


явное
выраж.

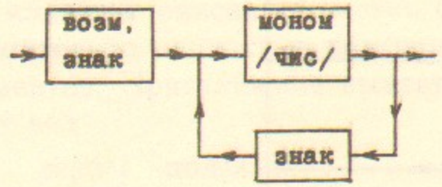
унар.
опер.



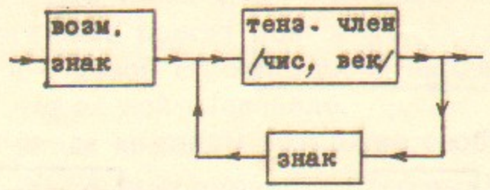
бинар.
опер.



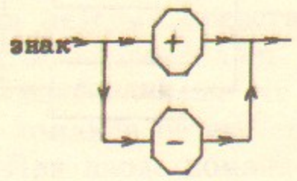
поли



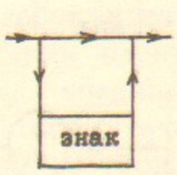
тензор



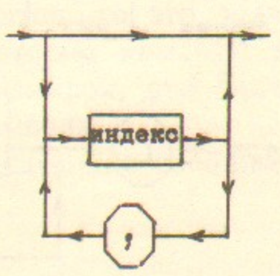
знак

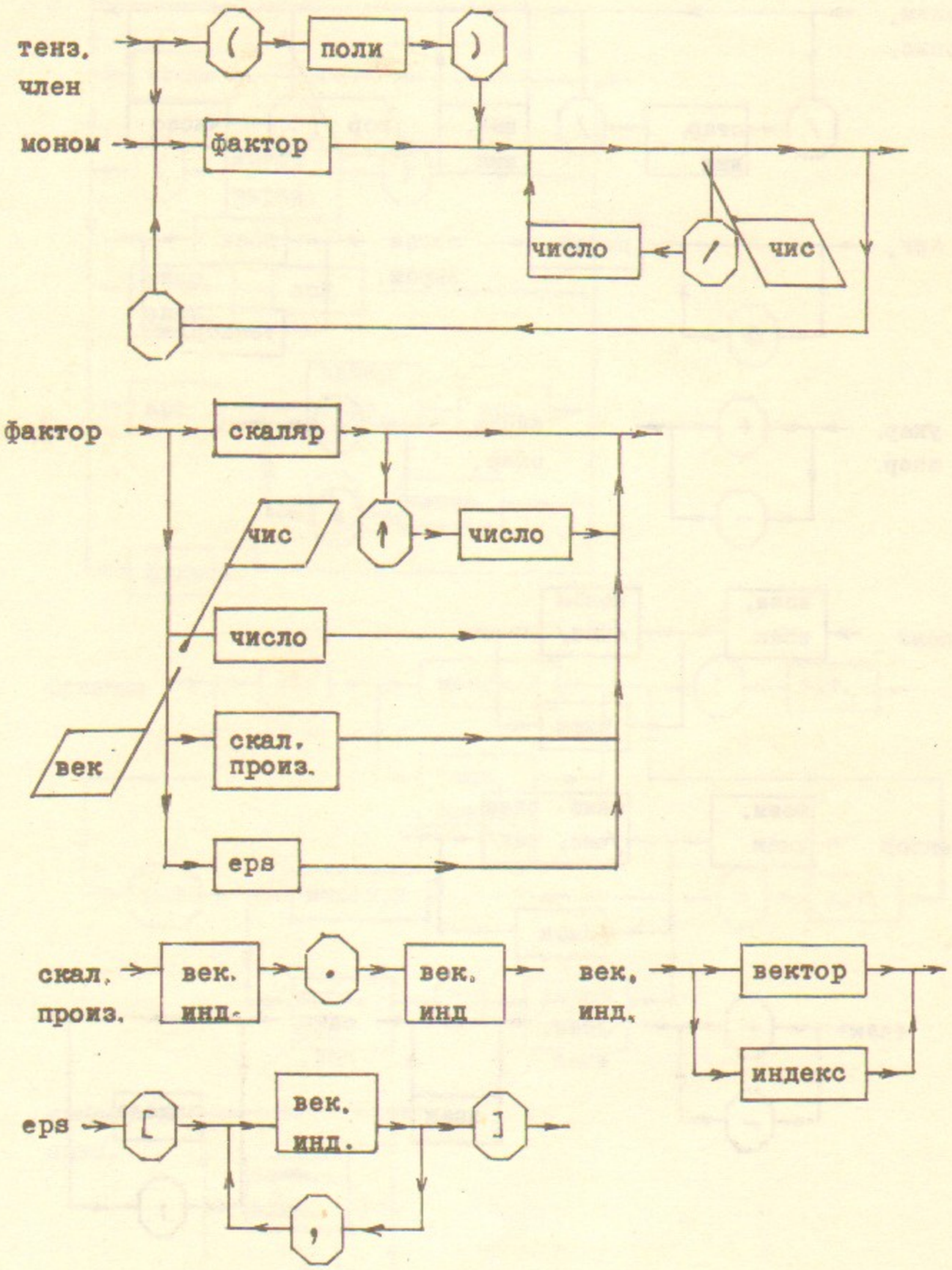


возм.
знак



спис.
инд.





Приложение 2. Обработка ошибок

Когда программа обнаруживает ошибку, она выдает сообщение. В нем указывается номер строки, при вводе которой она обнаружена. Если включен флаг echo, перед сообщением выводится стрелка, указывающая на символ, следующий за лексемой, при анализе которой обнаружена ошибка. Стрелка указывает на символ в копии входной строки, которая выводится при включенном флаге echo. Если флаг echo выключен, но включен флаг dialog, также выводится стрелка, указывающая на символ в строке, вводимой пользователем.

Нормальная реакция программы на ошибку состоит в том, что после выдачи сообщения команда, в которой обнаружена ошибка, не выполняется, остаток входной строки пропускается, и с новой строки вводится следующая команда. Однако при включенном флаге dialog в случае некоторых ошибок программа приглашает пользователя повторить ввод той конструкции, в которой обнаружена ошибка. Эта возможность предусмотрена потому, что некоторые команды (например, содержащие полином или тензор, а также команды описания) могут быть весьма длинными, и было бы утомительно полностью повторять их в случае ошибки в одном из элементов. Приглашение выдается после сообщения об ошибке, и имеет вид

repeat ⟨подсказка⟩

где подсказка показывает пользователю, какую именно конструкцию программа просит повторить. Остаток входной строки пропускается, и с новой строки заново вводится эта конструкция, а также все, что за ней следовало.

При вводе команды возможно сообщение

error in command : ⟨лексема⟩

Если лексема представляет собой идентификатор, то это значит, что он не совпадает ни с одним идентификатором команды или переменной. Если же лексема—специальный символ, это значит, что команда не может начинаться с такого символа.

При вводе команд описания возможны следующие сообщения. Сообщение

error in old name : ⟨лексема⟩

означает, что после символа '/' в начале элемента описания сле-

дует лексема, отличная от идентификатора, или идентификатор, не совпадающий ни с одним старым именем от текущей позиции до конца таблицы. Сообщение

error in declaration element : ⟨лексема⟩

означает, что после старого имени следует лексема, отличная от символа '/'. Сообщение

error in redefined name : ⟨идентификатор⟩

может выдаваться в двух случаях. Если оно появилось после старого имени, значит, при передаче старых имен до того, которое задано в конструкции ⟨стар.имя⟩, встретился идентификатор, совпадающий с одним из ранее описанных в этой же команде. Он и указывается в сообщении об ошибке. Если же оно появилось после нового имени, значит, оно совпадает с одним из имен, ранее описанных в этой же команде или переданных из старой таблицы. Сообщение

error in illegal name : ⟨идентификатор⟩

означает, что этот идентификатор совпадает с идентификатором другого типа, для которого такое совпадение запрещено. Так, при описании полиномиальных или тензорных переменных (команды poly и tensor) это сообщение означает, что описываемый идентификатор совпадает с идентификатором команды, функции или переменной другого типа. При описании скаляров, индексов или векторов (команды scalar, index и vector) это сообщение означает, что описываемый идентификатор совпадает с идентификатором одной из двух остальных групп. Сообщение

error in order : ⟨лексема⟩

возможно при вводе команды scalar, и означает, что после символа ':' следует лексема, отличная от числа. Сообщение

error in declaration list : ⟨лексема⟩

означает, что после элемента описания следует лексема, отличная от символов ',', ';' и '?'. Сообщение

error in long declaration : ,

означает, что делается попытка описать больше идентификаторов данного типа, чем это максимально возможно.

Во всех этих случаях, если включен флаг dialog, после сообщения об ошибке выдается приглашение повторить последний элемент описания. Подсказка начинается с '/old/new', что напомина-

ет о структуре элемента описания. Далее в случае команды scalar следует ':order'. Подсказка завершается ',...;', что напоминает, что после вводимого элемента описания могут следовать другие через запятую, а заканчивается список символом ';' (или '?').

Мономы непосредственно входят в присваивания с функциями dif и sub в правой части. Они также служат для построения полиномов и тензоров, которые входят в присваивания с явным выражением в правой части и команды установки размерности и скалярного произведения. Во всех этих случаях возможно появление сообщений об ошибках при вводе монома. Сообщение

error in power : ⟨лексема⟩

означает, что после символа '↑' следует лексема, отличная от числа. Сообщение

error in index : ⟨индекс⟩

означает, что индекс встретился в мономе не первый раз (оно может выдаваться при анализе первой или второй позиции в скалярном произведении или позиции в списке eps). Сообщение

error in scalar product : ⟨лексема⟩

означает, что после идентификатора первого вектора или индекса следует лексема, отличная от символа '.'. Сообщение

error in second vector : ⟨лексема⟩

означает, что после символа '.' следует лексема, отличная от идентификатора вектора или индекса. Сообщение

error in ⟨вектор⟩.⟨вектор⟩

означает, что встретилось скалярное произведение двух векторов. Сообщение

error in eps list : ⟨лексема⟩

означает, что после очередного вектора или индекса в списке eps следует лексема, отличная от ',', или после последнего—лексема, отличная от ']' (в частности, это может означать, что в списке eps неправильное количество индексов и векторов). Сообщение

error in eps vector : ⟨лексема⟩

означает, что в списке eps встретилась лексема, отличная от идентификатора вектора или индекса. Сообщение

error in factor : ⟨лексема⟩

означает, что фактор начинается с недопустимой лексемы. Возможно, фактор такого типа запрещен в мономе данного типа,

например, числа и делители запрещены в мономах в функциях dif и sub; скалярные произведения и конструкции eps запрещены в мономах, входящих в полином, функцию dif и sub без списка индексов; конструкция ⟨eps⟩ запрещена при нулевой eps-размерности или если она встречается в мономе не в первый раз. Сообщение

error in denominator : ⟨лексема⟩

означает, что после символа '/' следует лексема, отличная от числа, или 0. Сообщение

error in monom : ⟨лексема⟩

означает, что после фактора (или полинома в скобках в случае тензорного члена) и делителей следует лексема, не являющаяся ни символом '*' и ни одним из символов, которые могут следовать за мономом данного типа. В полиноме или тензоре за мономом может следовать знак '+' или '-', либо символ ')', заканчивающий полином или тензор; в функции dif за мономом должен следовать символ ':', а в функции sub—'='. Сообщение

error in negative power

означает, что суммарная степень одного или нескольких скаляров оказалась отрицательной после завершения ввода монома такого типа, для которого это запрещено (т.е. во всех случаях, кроме функции dif).

Во всех этих случаях, если включен флаг dialog, после сообщения об ошибке выдается приглашение повторить последний моном. Если подсказка начинается с 'n*', значит, разрешены числа в числителе и знаменателе, т.е. моном входит в полином или тензор. Если далее следует 's↑n', то отрицательные степени скаляров запрещены, а если 's↑-n'—разрешены (т.е. этот моном входит в функцию dif). Если далее следует '*v.i', то разрешены скалярные произведения, т.е. моном входит в тензор или функцию sub со списком индексов. При ненулевой eps-размерности в этом случае разрешена и конструкция ⟨eps⟩. Далее в подсказке следует '⋯', что напоминает, что факторы описанных типов могут повторяться неоднократно. Если далее следует '+⋯)', значит, моном входит в полином или тензор, и после него через знак '+' (или '—') могут следовать другие мономы, либо символ ')', завершающий полином или тензор. Если вместо этого стоит ':', моном входит в функцию dif, где после него должен стоять символ ':', а в случае '='—в функцию sub, где после него должен стоять символ '='.

При вводе команды order возможно сообщение

error in order : ⟨лексема⟩

означающее, что после идентификатора команды следует лексема, отличная от числа и символа '?'. Аналогично, при вводе команды eps возможно сообщение

error in eps : ⟨лексема⟩

означающее, что после идентификатора команды следует лексема, отличная от числа и символа '?', или число слишком велико—больше или равно максимальному числу индексов. При вводе команды dim возможно сообщение

error in dim : ⟨лексема⟩

означающее, что после идентификатора команды следует лексема, отличная от символов '(' и '?'. При вводе команды установки скалярного произведения возможен ряд сообщений об ошибках. Сообщение

error in first vector : ⟨лексема⟩

означает, что после символа '(' следует лексема, отличная от идентификатора вектора. Сообщение

error in scalar product : ⟨лексема⟩

означает, что после идентификатора первого вектора следует лексема, отличная от символа '.'. Сообщение

error in second vector : ⟨лексема⟩

означает, что после символа '.' следует лексема, отличная от идентификатора вектора. Сообщение

error in equality : ⟨лексема⟩

означает, что после скалярного произведения следует лексема, отличная от символов '=' и ')'. При вводе команд установки флага '+' и '-' возможно собщение

error in flag : ⟨лексема⟩

означающее, что после символа '+' или '-' следует лексема, отличная от идентификатора флага.

Конструкция ⟨аргумент⟩ входит в команды write и присваивания с унарной операцией, бинарной операцией, операцией '↑' и функцией. Во всех этих случаях возможно появление сообщений об ошибках при вводе аргумента. Сообщение

error in argument : ⟨лексема⟩

означает, что на месте аргумента (возможно, после символа '@')

находится лексема, которая не является идентификатором переменной нужного типа (или в тех случаях, когда допустим любой тип аргумента, т.е. в командах delete и write и в присваивании с унарной операцией—переменной любого типа). Сообщения

error in undefined argument 1 : ⟨полин.перем.⟩
error in undefined argument 2 : ⟨полин.перем.⟩
error in undefined tens argument 1 : ⟨тенз.перем.⟩
error in undefined tens argument 2 : ⟨тенз.перем.⟩

означают, что указанная переменная не имеет значения (возможно, оно было уничтожено раньше в этом же присваивании).

При вводе присваивания возможно сообщение

error in assignment : ⟨лексема⟩

означающее, что после идентификатора полиномиальной или тензорной переменной следует лексема, отличная от символа '='. Сообщение

error in operation : ⟨лексема⟩

означает, что после аргумента в правой части присваивания следует лексема, отличная от бинарной операции, а в случае полиномиального аргумента—и от операции '↑'. После выяснения типа присваивания и ввода всех аргументов могут появиться сообщения

error in redefined result : ⟨полин.перем.⟩
error in redefined tens result : ⟨тенз.перем.⟩

означают, что делается попытка присвоить значение переменной, которая его уже имеет.

При вводе функции dif со скалярным произведением возможен ряд сообщений об ошибках. Сообщение

error in scalar product : ⟨лексема⟩

означает, что после идентификатора первого вектора или индекса следует лексема, отличная от символа '.'. Сообщение

error in second vector : ⟨лексема⟩

означает, что после символа '.' следует лексема, отличная от идентификатора вектора или индекса. После окончания ввода скалярного произведения могут появиться сообщения

error in vector : ⟨индекс⟩
error in index : ⟨вектор⟩

означающие, что использован индекс вместо вектора или наоборот.

Сообщение

error in dif vector : ⟨лексема⟩

означает, что после скалярного произведения следует лексема, отличная от символа ':'.

При вводе функции sub без списка индексов возможно сообщение

error in sub : ⟨лексема⟩

означающее, что после второго аргумента следует лексема, отличная от символа ':'. При вводе функции sub со списком индексов возможен ряд сообщений об ошибках. Сообщение

error in index : ⟨лексема⟩

означает, что в списке произвольных индексов встречена лексема, отличная от идентификаатора индекса. Сообщение

error in index list : ⟨лексема⟩

означает, что после индекса в списке следует лексема, отличная от символов ',' и ':'. Сообщение

error in sub tensor : ⟨лексема⟩

означает, что после второго аргумента следует лексема, отличная от символа ':'.

Приложение 3. Машинно-зависимые особенности

В настоящее время программа доступна на Одре-1305, ЕС ЭВМ, Электронике 100-25 и совместимых с ними машинах; ожидается, что она будет перенесена на CDC-6700 и БЭСМ-6. Количественные характеристики и ограничения собраны в таблице.

	ICL	IBM	PDP
Максимальная длина идентификатора	8	8	8
Стандартное значение чисел скаляров, индексов, векторов	8	8	8
Их рекомендуется округлять вверх до кратного	4	4	2
Число скаляров ⩽ Число индексов ⩽ Число полин. перем. ⩽ Число тенз. перем. ⩽	47	63	255
Степень полиномов ⩽ Число индексов + число векторов ⩽	63	255	255
Стандартная длина строки вывода	80	120	80
Флаг dialog по умолчанию	+	—	+
Команда halt	+	—	+

A. ОДРА-1305
и другие машины, совместимые с ICL-1900

Разработка программы велась на машине ОДРА-1305 с операционой системой GEORGE-3. Для работы с программой в диалоговом режиме необходимо выполнить команды

```
lo diracbin
as *cr0
as *lp0
en 0
```

Если необходим ввод из файла или вывод в файл, он указывается в соответствующей команде as. Если стандартный заказ памяти (20K слов) не является удовлетворительным, перед командой en нужно выполнить команду

```
al 147,⟨память⟩
```

где ⟨память⟩—требуемое количество памяти в словах.

После начала работы программа выводит пустую строку и запрашивает ввод. Приглашение к вводу имеет вид '__'; конец строки—символ etx. Первая строка ввода игнорируется! Поэтому при вводе с терминала следует сразу нажать клавишу etx, а при вводе из файла его первая строка должна быть пустой. Далее программа представляется, затем запашивает ввод команды.

При нормальном окончании работы программы происходит событие

```
:halted : ok
```

Команда halt вызывает событие

```
:halted : 00
```

после чего можно командами as изменить направление ввода или вывода и продолжить работу по команде rm. Событие

```
:halted : error 307 at xxxxx
```

означает нехватку памяти для вычисления. Любое другое событие, вероятно, сигнализирует об ошибке в программе.

B. ЕС ЭВМ
и другие машины, совместимые с IBM-360, 370

Программа была перенесена на ЕС ЭВМ с операционной системой OS MVT. Для ее выполнения в пакетном режиме необходим шаг задания типа

```
// exec pgm=dirac
//sysprint dd sysout=a
//sysin dd *
```

Требуемое количество памяти указывается в карте exec. Сообщение

pascal termination log : heap overflow или stack overflow

свидетельствует о нехватке памяти для вычисления. Любое другое сообщение, вероятно, сигнализирует об ошибке в программе.

С. Электроника 100-25, СМ-4
и другие машины, совместимые с PDP-11

Программа была перенесена на минимашины типа Электроника 100-25, СМ-4 и другие, совместимые с PDP-11, с операционной системой RSX-11M. Для ее выполнения в диалоговом режиме необходимо выполнить команду

run dirac

Приглашение к вводу имеет вид 'dir⟩' и выводится только при включенном флаге dialog; конец строки—символ cr. После начала работы программа представляется; необходимо нажать клавишу cr, тогда программа выведет приглашение, после чего можно вводить команды.

Команда halt прикрепляет ввод и вывод к терминалу и включает флаг dialog; после этого все происходит, как в начале работы. Дополнительно реализованы команды

⟨ ⟨файл⟩
⟩ ⟨файл⟩

присоединяющие, соответственно, канал ввода и вывода к указанному файлу и выключающие флаг dialog. Первая строка вводимого файла игнорируется. Конструкция ⟨файл⟩ представляет собой стандартную спецификацию файла в RSX, в качестве расширения по умолчанию используется '.dir'. Спецификация файла должна оканчиваться пробелом или переходом на новую строку, перед ней они пропускаются. Если программе не удается открыть файл, выводится сообщение об ошибке

error in file : ⟨файл⟩

Сообщения об ошибках выполнения

1 not enough memory

или

33 stack overflow

свидетельствуют о нехватке памяти для вычисления. Любые другие, вероятно, сигнализируют об ошибке в программе.

Приложение 4. Пример работы с программой

В этом приложении воспроизведен сеанс диалога с программой dirac, иллюстрирующий ее основные возможности и содержащий подробные комментарии. Перед строками текста, введенными пользователем, стоит приглашающий к вводу знак '__ ', перед строками, выведенными программой, его нет. Максимальное количество символов в выводимой программой строке было установлено равным 60.

```
dirac version 2.1
__ com 'dirac is a pascal program for algebraic calculations
        with polynomials and tensors'
__ scalar x,y,z:1;              declaration of scalars, z has order 1
__ poly a,b,c,d;                declaration of polynomial variables
__ order 2                      maximum order of smallness
__ a=(1+2/3*x*y↑2—3/2*x↑2*y)
__ write @a                       value of a variable is retained
1+2/3*x*y↑2—3/2*x↑2*y
__ write a                      if it is preceeded by @
1+2/3*x*y↑2—3/2*x↑2*y
__ write a                      and otherwise deleted
          ↑
error at line 9 in undefined argument 1 : a
__ a=(1+x)                      new value cannot be assigned to a
__ a=(1+x+y)                    variable until old one is deleted
       ↑
error at line 11 in redefined result : a
__ delete a
__ a=(1+x+y)
__ b==@a↑5                        raise a to the power 5
__ write @b
1+5*y+10*y↑2+10*y↑3+5*y↑4+y↑5+5*x+20*x*y+30*x*y↑2+20*x
*y↑3+5*x*y↑4+10*x↑2+30*x↑2*y+30*x↑2*y↑2+10*x↑2*y↑3+10
*x↑3+20*x↑3*y+10*x↑3*y↑2+5*x↑4+5*x↑4*y+x↑5
__ c==@a↑3
__ d=a↑2
__ d=—d
__ c=c*d
__ b=b+c
__ write b                      result should be 0
0
__ a=(1+x+z)
__ b=a↑5                        terms with too large order of
```

```
__ write @b                          smallness are dropped out
1+5*z+10*z↑2+5*x+20*x*z+30*x*z↑2+10*x↑2+30*x↑2*z+30*x↑2
*z↑2+10*x↑3+20*x↑3*z+10*x↑3*z↑2+5*x↑4+5*x↑4*z+x↑5
__ a=dif x:@b                        derivative in x
__ write @a
5+20*z+30*z↑2+20*x+60*x*z+60*x*z↑2+30*x↑2+60*x↑2*z+30*x↑2
*z↑2+20*x↑3+20*x↑3*z+5*x↑4
__ c=dif x*z↑2:@b                    first derivative in x and second in z
__ write c
60+120*x+60*x↑2
__ a=dif x↑—1:a                      integral in x
__ a=—a
__ a=a+@b
__ write a                           x independent terms only
1+5*z+10*z↑2
__ a=(y)
__ b=sub x↑2*z=a:b                   substitute a for x↑2*z in b
__ write b
1+5*z+10*z↑2+30*y+30*y*z+5*x+20*x*z+30*x*z↑2+20*x*y+10*x
*y*z+10*x↑2+5*x↑2*y+10*x↑3+5*x↑4+x↑5

__
__ com 'now some examples with tensors'
__ vector u,v,w,p;                   declaration of vectors
__ tensor e,f,g;                     declaration of tensor variables
__ dim(4)                            space dimension
__ eps 4                             number of indices of the tensor eps
__ index l,m,n,r;                    declaration of indices
__ (u.u=x)
__ (v.v=y)                           scalar products
__ (u.v=z)
__ e=((1+z)*u.m*l.n*u.r+u.m*u.n*u.l*u.r)        u.m means vector
__ f=(l.m*n.r)                       with index, and l.n means metric tensor
__ g=e*f                             constraction over repeated indices
__ write g
(x+x*z+x↑2)
__ e=([u,v,m,n])                     i* unit antisymmetric tensor
__ f=—+@e
__ g=e*f
__ write g
(—2*z↑2+2*x*y)
__ e=(u.m*u.n*v.l)
```



```
__ f=dif u.r:@e                      derivative in u.r
__ write f
1*v.l*r.m*u.n+1*v.l*u.m*r.n
__ f=dif u.l:@e
__ write f
1*u.m*v.n+1*v.m*u.n
__ f=dif u.m:e
__ write f
5*v.l*u.n
__ c=((1+x)*u.m*u.n)
__ f=(v.m—w.m)
__ g=sub:x*u.m=f:e                   substitute f for x*u.m in e
__ write g
1*u.m*u.n+1*v.m*u.n—1*w.m*u.n
__ e=(v.l*u.m*u.n+v.m*u.n*u.l+v.n*u.m*u.l)
__ f=(m.n/4)
__ g=sub m,n:u.m*u.n=f:e  for all m,n substitute f
__ write g for u.m*u.n in e
1/4*m.l*v.n+1/4*n.l*v.m+1/4*v.l*n.m
__ e=(u.l*[p,w,u,m])
__ f=(u.m*v.n—u.n*v.m)
__ g=sub m,n:[p,w,m,n]=f:e           in [] a vector may appear
__ write g                           instead of an arbitrary index
—z*u.l*u.m+x*u.l*v.m
__ end
```

ЛИТЕРАТУРА

1. *D.Barton, J.P.Fitch.* Rep. Progr. Phys. 35, 235 (1972)
2. *В.П.Гердт, О.В.Тарасов, Д.В.Ширков.* УФН 130, 113 (1980)
3. *A.C.Hearn.* REDUCE User's Manual, 2-nd ed. Univ. of Utah (1973)
4. *H.Strubbe.* Comp. Phys. Comm. 8, 1 (1974)
5. *J.P.Fitch.* CAMAL User's Manual. Univ. of Cambridge (1975)
6. *K.Jensen, N.Wirth.* PASCAL User Manual and Report. 2-nd ed. Springer (1975); русский перевод: *К.Йенсен, Н.Вирт.* ПАСКАЛЬ: руководство для пользователя и описание языка. Финансы и статистика (1982);
   A.M.Addyman a.o. A draft description of PASCAL. Software—practice and experience 9, 381 (1979);
   русский перевод в кн.: *О.Н.Перминов.* Язык программирования ПАСКАЛЬ. Радио и связь (1983)
7. *А.Г.Грозин.* Доклад на 3 Всесоюзной конференции "Диалог Человек—ЭВМ", Протвино, 5—7 июля 1983; будет опубликовано в материалах конференции.

А.Г.Грозин

**DIRAC—
МАШИННО-НЕЗАВИСИМАЯ ПРОГРАММА
ДЛЯ АЛГЕБРАИЧЕСКИХ ВЫЧИСЛЕНИЙ
С ПОЛИНОМАМИ И ТЕНЗОРАМИ**

Ответственный за выпуск—С.Г.Попов

Подписано в печать 24 октября 1983 г. МН 00972
Формат бумаги 60×90 1/16.
Объем 1,9 печ.л., 1,5 учетно-изд.л.
Тираж 290 экз. Бесплатно. Заказ № 117

Набрано в автоматизированной системе на базе фотонаборного автомата ФА-1000 и ЭВМ «Электроника» и отпечатано на ротапринте Института ядерной физики СО АН СССР,
Новосибирск, 630090, пр. академика Лаврентьева, 11