ИПМ им.М.В.Келдыша РАН • Электронная библиотека

Препринты ИПМ • Препринт № 243 за 1987 г.

ISSN 2071-2898 (Print)
ISSN 2071-2901 (Online)

А. В. Климов, С. А. Романенко

Система программирования РЕФАЛ-2 для ЕС ЭВМ. Описание входного языка

Статья доступна по лицензии
Creative Commons Attribution 4.0 International

Рекомендуемая форма библиографической ссылки: Климов А. В., Романенко С. А. Система программирования РЕФАЛ-2 для ЕС ЭВМ. Описание входного языка // Препринты ИПМ им. М.В.Келдыша. 1987. № 243. 54 с.
https://library.keldysh.ru/preprint.asp?id=1987-243

Ордена Ленина
ИНСТИТУТ ПРИКЛАДНОЙ МАТЕМАТИКИ
имени М.В. Келдыша
Академии наук СССР

А.В. Климов, С.А. Романенко

СИСТЕМА ПРОГРАММИРОВАНИЯ РЕФАЛ-2 ДЛЯ ЕС ЭВМ. ОПИСАНИЕ ВХОДНОГО ЯЗЫКА

Москва

Ордена Ленина
ИНСТИТУТ ПРИКЛАДНОЙ МАТЕМАТИКИ
им. М.В. Келдыша
Академии Наук СССР

Анд.В.Климов, С.А.Романенко

СИСТЕМА ПРОГРАММИРОВАНИЯ РЕФАЛ-2 ДЛЯ ЕС ЭВМ.

ОПИСАНИЕ ВХОДНОГО ЯЗЫКА

Москва
1987

УДК 681.3.06

Описан входной язык системы программирования Рефал-2 для ЕС ЭВМ. Язык Рефал (алгоритмический язык рекурсивных функций) предназначен для обработки символьной информации, представленной в виде выражений, имеющих древовидную структуру. Основными изобразительными средствами Рефала являются сопоставление с образцом, подстановка и рекурсия.

КЛЮЧЕВЫЕ СЛОВА И ФРАЗЫ: Рефал, обработка символьной информации, функциональное программирование, язык программирования.

С О Д Е Р Ж А Н И Е

ВВЕДЕНИЕ

Различные описания языка Рефал отличаются в некоторых деталях. Вариант Рефала, соответствующий [БР 1977, Т 1974], получил название "базисного рефала".

В системе программирования Рефал-2 для ЕС ЭВМ в качестве входного языка служит некоторое расширение базисного Рефала. Это же расширение было реализовано в трансляторе Рефала для БЭСМ-6 в рамках мониторной системы "Дубна" [КПРТР 1975, КР 1975].

Ниже описан входной язык системы программирования Рефал-2, который в дальнейшем для краткости именуется просто "Рефал".

1. НАЗНАЧЕНИЕ ЯЗЫКА

Язык Рефал (алгоритмический язык рекурсивных функций) был создан в качестве абстрактного метаалгоритмического языка, предназначенного для формализации семантики алгоритмических языков [Т 1966, Т 1968, БР 1977].

Хотя Рефал был задуман как метаалгоритмический язык, он представляет собой некоторый язык для обработки символьной информации, поэтому, помимо описания семантики алгоритмических языков, он нашел и другие, не менее важные применения. В первую очередь это машинное выполнение громоздких аналитических выкладок в теоретической физике и прикладной математике, интерпретация и компиляция языков программирования, машинное доказательство теорем, моделирование целенаправленного поведения и т.п. Общим для всех этих применений является то, что мы заставляем машину совершать сложные

преобразования над об'ектами, определенными в некоторых формализованных языках (алгоритмические языки, язык алгебры, язык исчисления предикатов и т.д.).

2. ОБРАБАТЫВАЕМЫЕ ДАННЫЕ

Данные, обрабатываемые рефал-программами, являются языковыми об'ектами, т.е. некоторыми последовательностями знаков.

В каждой конкретной реализации Рефала множество знаков - это тот набор литер, который обеспечен устройствами подготовки и отображения данных (перфораторами, дисплеями, принтерами).

Следующие знаки имеют в рефал-программах особое, фиксированное значение:

< > ( ) / ' = S W V E +

Эти знаки называются **специальными знаками**. Они, конечно, имеются в любой реализации Рефала.

Из знаков строятся более крупные единицы - **символы, термы и выражения**.

Символ является минимальной семантической единицей, не расчленимой на составные части средствами языка Рефал.

Символы делятся на два класса: **составные символы и символы-литеры** (или, как их еще называют, "об'ектные знаки").

Любой составной символ имеет следующий вид:

/[ТЕЛО СОСТАВНОГО СИМВОЛА]/

где [ТЕЛО СОСТАВНОГО СИМВОЛА] - это последовательность литер, не содержащая литеру "/".

Множество составных символов разбивается на непересекающиеся классы, в соответствии с тем, какой вид имеют их тела. В данной реализации Рефала имеются следующие классы составных символов: символы-метки, символы-числа и символы-ссылки.

С и м в о л а м и - м е т к а м и называются составные символы, телом которых является идентификатор, т.е. последовательность букв, цифр и знаков "-", начинающаяся с буквы. Длина тела символа-метки не ограничивается, однако принимаются во внимание только первые 255 литер тела, а все последующие - игнорируются. Таким образом, все символы-метки, у тел которых совпадают первые 255 литер, считаются совпадающими.

Например, следующие символы являются символами-метками:

```
/ALPHA/
/L2A4/
/ЭТОПРИМЕРДЛИННОЙМЕТКИ/
/ЭТО-ПРИМЕР-ДРУГОЙ-ДЛИННОЙ-МЕТКИ/
/Z---Z---/
```

С и м в о л а м и - ч и с л а м и называются составные символы, тело которых представляет собой целое неотрицательное число, т.е. последовательность десятичных цифр. Например:

```
/Ø/  /I/  /5I2/  /23/
```

Тело символа-числа должно находиться в диапазоне от Ø до I6777215 (2**24-I).

С и м в о л а м и - с с ы л к а м и называются составные символы, тело которых начинается с литеры "%", вслед за которой идет шесть шестнадцатеричных цифр. Например:

/%ØØFAC9/ /%FFFFFF/ /%ABCDEF/

Символы-ссылки нельзя употреблять в качестве констант в рефал-программах, однако они могут порождаться в процессе работы рефал-программы и входить в обрабатываемые выражения. Назначение и использование символов-ссылок будет описано ниже.

Символы-метки используются в программах в качестве имен функций. Символы-числа служат для обозначения чисел.

Помимо составных символов имеется еще один класс символов - символы-литеры. Любой символ-литера имеет следующий вид:

'[ЛИТЕРА]'

где [ЛИТЕРА] - произвольная литера, отличная от апострофа.

Символы-литеры служат для обозначения литер. Так как апостроф тоже является литерой, которую нужно уметь обрабатывать, для него имеется особое обозначение:

''

т.е. два апострофа, идущих подряд.

Следует заметить, что символ

' '

обозначает не апостроф, а литеру "пробел", которая является такой же полноценной литерой, как и все остальные.

Более крупной, чем символ, единицей является выражение.

Выражение строится из символов, круглых скобок "(" и ")" (именуемых структурными скобками), угловых скобок "<" и ">" (именуемых функциональными или конкретизационными

скобками) и п е р е м е н н ы х.

Структурные скобки придают обрабатываемым об'ектам структуру дерева. Назначение функциональных скобок и переменных об'ясняется в следующих разделах.

В ы р а ж е н и е м называется произвольная последовательность символов, переменных и скобок, правильно построенная относительно скобок. Например:

```
'A' 'B' 'C' ()
(('A') 'B'()) ()
<</X/> (</Y/>)>
```

Т е р м о м называется выражение, которое представляет собой либо символ, либо выражение, заключенное в структурные или функциональные скобки. Например:

```
'+'
( 'A' 'B' 'C' () )
((()) ())
</PI/ /I/>
```

Таким образом, всякое выражение есть последовательность из некоторого (быть может - нулевого) числа термов.

Приведенные выше определения выражения и терма можно формализовать следующим образом:

```
[ВЫРАЖЕНИЕ] ::=  [ПУСТО] | [ТЕРМ] [ВЫРАЖЕНИЕ]
[ТЕРМ] ::=  [СИМВОЛ] | [ПЕРЕМЕННАЯ] |
            [СТРУКТУРНЫЙ ТЕРМ] |
            [ФУНКЦИОНАЛЬНЫЙ ТЕРМ]
[СТРУКТУРНЫЙ ТЕРМ] ::=      ( [ВЫРАЖЕНИЕ] )
[ФУНКЦИОНАЛЬНЫЙ ТЕРМ] ::=  < [ВЫРАЖЕНИЕ] >
[ПУСТО] ::=
```

Очень часто приходится иметь дело не с отдельными символами-литерами, а с цепочками из нескольких подряд идущих литер. Для таких случаев предусмотрено сокращенное

обозначение. А именно, литеры записываются подряд и вся цепочка литер заключается в апострофы. Например, цепочку литер "ABC", изображаемую как

```
'A' 'B' 'C'
```

можно в сокращенной форме записывать как

```
'ABC'
```

Если цепочка литер содержит апострофы, они изображаются парами апострофов. При этом цепочка, составленная из одних апострофов, удваивается, но апострофами дополнительно не обрамляется. Например:

```
ABC     →     'ABC'
A'C     →     'A''C'
'       →     ''
''      →     ''''
'A'B    →     '''A''B'
A'B'    →     'A''B'''
```

где стрелка "→" обозначает слова "изображается в виде".

Таким образом, одну и ту же цепочку символов-литер можно изобразить многими способами. Например, цепочка литер "A'B" может быть представлена любым из следующих способов:

```
'A' '' 'B'
'A''' 'B'
'A' '''B'
'A''B'
```

Для повышения наглядности рефал-программы можно вставлять любое количество пробелов между переменными, скобками, составными символами и цепочками литер. Цепочку литер можно разбить на несколько цепочек, но при этом между получившимися частями обязательно должен стоять по крайней мере один пробел, так как

'A' 'B' обозначает AB
'A''B' обозначает A'B

Кроме того, следует обратить внимание на то, что структурные скобки "(" и ")" в Рефале не являются символами. Поэтому структурные скобки не имеют ничего общего с символами-литерами '(' и ')'.

3. ФУНКЦИИ

Всякая программа, написанная на Рефале, определяет некоторый набор функций. Каждая из этих функций имеет один аргумент, значениями которого могут быть выражения.

Аргументами функций могут быть не произвольные выражения, а только такие, которые не содержат функциональных скобок и переменных. Такие выражения в дальнейшем именуются о б ' е к т н ы м и.

В общепринятой математической записи обращение к функции FUNC с аргументом [ARG] имеет следующий вид:

FUNC([ARG])

В Рефале же принят другой синтаксис для вызовов функций:

</FUNC/ [ARG]>

т.е вызов функции заключается в функциональные скобки "<" и ">", а имя функции указывается с помощью символа-метки. Фактически, имя вызываемой функции представляет собой часть функционального терма - первый символ его содержимого.

Если "/FUNC/" - некоторый символ-метка, то комбинацию вида "</FUNC/" разрешается записывать в сокращенном виде: "<FUNC". Таким образом, функциональный терм

```
</FUNC/ </PI/ /3/>>
```

может быть записан любым из следующих способов:

```
<FUNC <PI /3/>>
<FUNC<PI/3/>>
```

В тех случаях, когда использование сокращенной записи могло бы привести к двусмысленности, следует вставлять один или несколько пробелов. Например

```
</FUNC/SX>
```

можно записать в виде

```
<FUNC SX>
```

но нельзя записать в виде

```
<FUNCSX>
```

так как это было бы воспринято как обращение к функции "FUNCSX". По этой же причине, если сразу же вслед за "<" находится переменная, после "<" следует вставить хотя бы один пробел. Например, нельзя

```
< SX EA>
```

сократить до

```
<SX EA>
```

Для обеспечения совместимости с другими реализациями Рефала в программах на Рефале-2 вместо знака "<" разрешается использовать знак "К", а вместо знака ">" - знак ".". Например, вместо

```
</FUNC/ </PI/ /3/>>
```

можно писать

```
K/FUNC/ K/PI/ /3/..
```

При использовании "К" и "." в качестве функциональных скобок не разрешается опускать знаки "/" после левой функциональной скобки.

Возникает вопрос, почему же в Рефале функции имеют только один аргумент? Ответ заключается в том, что список из нескольких аргументов является некоторым выражением, поэтому всякое обращение к функции от нескольких аргументов

```
FUNC([ARG1], [ARG2], ..., [ARGN])
```

можно рассматривать как обращение к функции от одного "большого" аргумента:

```
<FUNC ([ARG1]) ([ARG2]) ... ([ARGN]) >
```

П р о г р а м м а на языке Рефал представляет собой описание некоторого набора функций. Описание каждой функции имеет вид:

```
FUNC
        [L-1] = [R-1]
        [L-2] = [R-2]
        . . . . . ..
        [L-N] = [R-N]
```

где "FUNC" - имя функции, а

```
        [L-İ] = [R-İ]
```

суть п р е д л о ж е н и я или п р а в и л а к о н к р е т и з а ц и и.

Каждое предложение является указанием для замены одного выражения на другое, поэтому оно состоит из левой части [L-İ] (заменяемое выражение) и правой части [R-İ] (результат замены). Синтаксически [L-İ] и [R-İ] являются выражениями.

Перед левой частью каждого предложения должен находиться хотя бы один пробел.

Если предложение не помещается в очередной строке, его можно перенести на следующие строки. Перенос допускается делать в тех местах, где разрешается вставлять пробелы. В любом из таких мест можно поставить знак "+" и продолжить предложение с любой позиции следующей строки.

Имя функции не обязано располагаться на отдельной строке. Его можно поместить в начале той строки, в которой находится первое предложение. При этом имя функции отделяется от начала предложения одним или несколькими пробелами. В этом случае описание функции имеет следующий вид:

```
FUNC    [L-1] = [R-1]
        [L-2] = [R-2]
        . . . . . ..
        [L-N] = [R-N]
```

С е м а н т и к а рефал-программы описывается в терминах абстрактной р е ф а л - м а ш и н ы. Рефал-машина имеет два запоминающих устройства: п о л е п а м я т и и п о л е з р е н и я.

Перед началом работы в поле памяти заносится описание набора функций, а в поле зрения - выражение, подлежащее обработке.

Пусть, например, в поле памяти находится единственное предложение:

```
XXX     = '137'
```

а в поле зрения - выражение

```
        <XXX>
```

Тогда рефал-машина заменит содержимое поля зрения на выражение

'137'

и остановится, поскольку в поле зрения не осталось ни одной пары функциональных скобок.

А что будет, если занести в поле памяти два предложения:

```
XXX     = '137'
        = '274'
```

Теперь для вычисления "<XXX>" пригодно не одно, а два предложения. Неоднозначность устраняется следующим образом. Рефал-машина просматривает предложения в том порядке, в котором они стоят в описании функции и применяет первое из них, которое окажется подходящим. Таким образом, в данном случае "<XXX>" будет заменено на '137'.

Поле зрения может содержать сколь угодно много функциональных термов, которые могут быть как угодно вложены друг в друга. Поэтому нужно договориться, каким образом рефал-машина будет выбирать выражение, с которого надо начинать процесс вычисления.

Пусть в поле зрения находится выражение, в котором имеются функциональные термы. Тогда некоторые из этих термов являются самыми внутренними, т.е. не содержат внутри себя других функциональных термов. Самый левый из таких термов мы будем называть в е д у щ и м.

Теперь опишем работу рефал-машины (для частного случая, когда рефал-программа не содержит переменных).

Работа рефал-машины разбивается на шаги. В начале каждого шага рефал-машина находит ведущий функциональный терм. Пусть этот терм имеет вид:

```
<FUNC [ARG]>
```

где "FUNC" - имя некоторой функции, а [ARG] - об'ектное выражение. Таким образом, предполагается, что содержимое

ведущего терма начинается с символа-метки. (Случай, когда это не выполнено, будет рассмотрен позже.)

Рефал-машина находит описание функции "FUNC" и начинает сравнивать [ARG] с левыми частями предложений в описании этой функции.

Пусть $\dot{I}$ - номер самого первого предложения, для которого [L-$\dot{I}$]=[ARG]. Тогда рефал-машина производит з а м е н у ведущего функционального терма, т.е.

<FUNC [ARG]>

заменяется на правую часть предложения

[R-$\dot{I}$]

после чего рефал-машина переходит к исполнению следующего шага.

Если же рефал-машина не находит ни одного предложения для которого [L-$\dot{I}$]=[ARG], она сообщает, что "отождествление невозможно" и останавливается. Это означает, что либо программа на Рефале, либо исходные данные для нее заданы неверно.

Работа рефал-машины продолжается до тех пор, пока в поле зрения имеется хотя бы один функциональный терм. Если же в поле зрения после завершения очередного шага не остается ни одного функционального терма, рефал-машина сообщает, что "вычисление окончено" и останавливается. При этом, результатом работы рефал-машины считается выражение, которое находится в поле зрения.

Пусть, например, поле памяти содержит следующий набор функций:

```
XXX
        = '137'
        = '274'
```

```
YYY
        = '2'
ADD
        ('137') '2' = '139'
```

а в поле зрения находится выражение

```
        <ADD (<XXX>) <YYY>>
```

Тогда на первом шаге ведущим будет терм "<XXX>", который заменится на '137', в результате чего поле зрения примет вид:

```
        <ADD ('137') <YYY>>
```

Теперь подлежит вычислению терм "<YYY>", что дает

```
        <ADD ('137') '2'>
```

На третьем шаге применяется функция "ADD", и в поле зрения оказывается выражение

```
        '139'
```

которое уже не содержит функциональных скобок и является окончательным результатом работы рефал-машины.

4. ПЕРЕМЕННЫЕ

Средства, описанные в предыдущих разделах, позволяют описывать только функции, области определения которых – конечные множества, ибо для каждого конкретного значения аргумента приходилось предусматривать особое предложение. Ясно, что так мы далеко не уйдем. Нужно уметь записывать предложения, применимые более, чем к одному об'ектному выражению. Для этого нужно ввести в предложения п е р е м е н н ы е, которые при различных применениях предложения могут

принимать различные значения.

Языковые об'екты, с которыми имеет дело рефал-машина, это всегда выражения, которые могут быть, в частности, термами или символами. Поэтому в Рефале-2 используются переменные четырех типов:

S-переменные, значением которых могут быть только символы;

W-переменные, значением которых могут быть только термы;

V-переменные, значением которых могут быть только непустые выражения;

E-переменные, значениями которых могут быть выражения (в том числе и пустые).

Любая переменная имеет следующий вид:

[ПРИЗНАК ТИПА] [СПЕЦИФИКАЦИЯ] [ИНДЕКС]

[ПРИЗНАК ТИПА] - это один из специальных знаков "S", "W", "V" или "E". Он указывает, к какому из четырех вышеперечисленных типов принадлежит переменная.

[СПЕЦИФИКАЦИЯ] - это описание дополнительных условий, налагаемых на множество допустимых значений переменной. Спецификация может быть пустой. В этом случае считается, что на возможные значения переменной не налагается никаких дополнительных ограничений. Т.е. значением S-переменной может быть любой символ, значением W-переменной - любой терм, значением V-переменной - любое непустое выражение, значением E-переменной - любое (в том числе и пустое) выражение.

[ИНДЕКС] переменной - это цифра, либо прописная буква русского или латинского алфавитов. Индексы переменных служат для того, чтобы различать между собой различные

переменные.

Например, S-переменными являются "SI", "S2" и "SA", W-переменными являются "WI" и "WX", V-переменными являются "V9" и "VZ", E-переменными являются "EI", "E5" и "EA".

Разрешив употреблять переменные в левых и правых частях предложений, мы получаем мощное изобразительное средство. Теперь, чтобы решить, применимо ли предложение

[L] = [R]

к об'ектному выражению [ARG], рефал-машина должна определить, является ли [ARG] частным случаем [L], т.е. можно ли вместо переменных, входящих в [L] подставить такие значения, чтобы получившееся об'ектное выражение совпало с [ARG]. При этом, значения переменных должны быть допустимыми, т.е. соответствовать их типам и спецификациям. Кроме того, все вхождения одной и той же переменной должны заменяться на одно и то же значение.

Описанное выше действие рефал-машины, по подбору значений переменных, называется *синтаксическим отождествлением*.

В том случае, если отождествление левой части [L] с [ARG] возможно, предложение является применимым, в противном случае - неприменимым.

Теперь мы следующим образом уточним описание работы рефал-машины.

На каждом шаге рефал-машина просматривает описание функции и находит самое первое применимое предложение. Затем она заменяет ведущий функциональный терм на правую часть этого предложения, предварительно подставив в нее вместо переменных те значения, которые получили эти переменные в результате синтаксического стождествления. Если же окажется, что все предложения функции неприменимы, рефал-машина сообщает, что "отождествление невозможно" и останавливается.

Разумеется, в правой части предложения разрешается использовать только такие переменные, которые входят в левую часть. Кроме того, все вхождения одной и той же переменной в левую и правую часть должны иметь одинаковый указатель типа переменной.

Рассмотрим несколько примеров. Допустим, что нужно описать функцию "FIRST-SYM", которая в качестве значения выдает первый символ аргумента. Например, чтобы результатом вычисления терма

```
<FIRST-SYM 'Z'('AB')'+F'>
```

был символ 'Z', а результатом вычисления

```
<FIRST-SYM /X1/ /X2/>
```

был символ /X1/.

Для этого достаточно ввести в поле памяти рефал-машины следующее описание функции:

```
FIRST-SYM
        SA EX = SA
```

Аналогично можно описать функцию, значением которой является последний символ аргумента:

```
LAST-SYM
        EX SA = SA
```

5. РЕКУРСИВНЫЕ ФУНКЦИИ

Поскольку правые части предложений могут содержать функциональные скобки, можно описывать функции в терминах других функций или применять рекурсию.

Опишем, например, функцию "REV", определенную для всех выражений. Значением этой функции является "зеркально перевернутое" исходное выражение. Так, вычисление терма

```
<REV 'A'('B'('CD')'F')>
```

должно давать

```
('F'('DC')'B')'A'
```

Функция "REV" описывается тремя предложениями:

```
REV     EI SX   = SX <REV EI>
        EI (EX) = (<REV EX>) <REV EI>
        =
```

Обратите внимание на то, что структурные скобки "(" и ")" не являются символами, и, в отличие от символов-литер '(' и ')', не могут быть значениями S-переменной. Поэтому, если аргумент кончается на правую структурную скобку, то первое предложение функции - не применимо. В этом случае применяется второе предложение.

Другой пример. Функция "SYMM" принимает значение 'T' если аргумент - симметричное выражение (т.е. такое, которое не изменяется в результате применения к нему функции "REV"), и принимает значение 'F' если аргумент не является симметричным выражением.

```
SYMM
        EX = <EQUAL (EX) <REV EX>>
EQUAL
        (EX) EX = 'T'
        (EX) EY = 'F'
```

Функцию "SYMM" можно описать и без обращения к функции "REV":

```
SYMM    = 'T'
        SX = 'T'
        SX EA SX = <SYMM EA>
        (EA) = <SYMM EA>
        () EA () = <SYMM EA>
        (WX E1) EA (E2 WY) = +
           <SYMM WX (E1) EA (E2) WY>
        EA = 'F'
```

6. СНЯТИЕ НЕОДНОЗНАЧНОСТИ ПРИ ОТОЖДЕСТВЛЕНИИ

Будем говорить, что некоторая переменная является VE-переменной, если она является V-переменной или E-переменной.

Если левая часть предложения содержит несколько VE-переменных, то может случиться, что существует несколько вариантов приписывания переменным значений, приводящих к отождествлению левой части предложения с аргументом функции. Пусть, например, в поле памяти находится функция

```
F       E1 ';' E2 = <G E1>  <F E2>
```

а в поле зрения - выражение

```
        <F 'A1:=A2;B1:=B2;C1:=C2'>
```

Это выражение может быть отождествлено с левой частью таким образом, что переменные "E1" и "E2" примут следующие значения:

```
        E1  ←  'A1:=A2'
        E2  ←  'B1:=B2;C1:=C2'
```

Однако, возможен и другой вариант отождествления, при котором переменные примут такие значения:

```
E1  ←  'A1:=A2;B1:=B2'
E2  ←  'C1:=C2'
```

Следовательно, необходимо договориться, как поступает рефал-машина при наличии такой неоднозначности.

Для устранения неоднозначности в Рефале-2 могут использоваться два метода: отождествление слева направо и отождествление справа налево.

При отождествлении слева направо рефал-машина выбирает тот вариант отождествления, при котором первая слева VE-переменная принимает самое короткое значение. Если это не устраняет неоднозначности, то такой же отбор производится по второй слева VE-переменной, затем - третьей слева и т.д. В нашем примере будет выбран первый из двух способов отождествления.

При отождествлении справа налево рефал-машина выбирает тот вариант отождествления, при котором первая справа VE-переменная принимает самое короткое значение. Если это не устраняет неоднозначности, то такой же отбор производится по второй справа VE-переменной, затем третьей справа и т.д. В нашем примере будет выбран второй из двух способов отождествления.

Если мы хотим, чтобы при применении некоторого предложения использовалось отождествление справа налево, следует сообщить об этом рефал-машине, поместив перед левой частью предложения ключевое слово "R", т.е. знак "R", за которым следует хотя бы один пробел. Если же мы хотим, чтобы при применении некоторого предложения использовалось отождествление слева направо, следует сообщить об этом рефал-машине, поместив перед левой частью предложения ключевое слово "L", т.е. знак "L", за которым следует хотя бы один пробел.

Если направление отождествления не указано явно, рефал-машина считает, что отождествление следует выполнять слева направо, поэтому ключевое слово "L" указывать не обязательно (и оно отсутствовало во всех приведенных примерах).

Например, описанная выше функция "F", выдаст в результате замены

```
<G 'A1:=A2'> <F 'B1:=B2;C1:=C2'>
```

Но если описать "F" следующим образом:

```
F       R E1 ';' E2 = <G E1>  <F E2>
```

то результатом замены будет

```
<G 'A1:=A2;B1:=B2'> <F 'C1:=C2'>
```

Отождествление слева направо и справо налево широко используются при программировании на Рефале. В качестве примера рассмотрим функцию "MAKE-SET", которая порождает множество термов, входящих в аргумент на нулевом уровне скобочной структуры. Эта функция просматривает выражение слева направо терм за термом. Для очередного терма проверяется, не стоит ли справа от него точно такой же терм. Если да, то очередной терм вычеркивается, в противном случае - оставляется. Оставшееся выражение обладает тем свойством, что составляющие его термы попарно различны. Например, результатом вычисления

```
<MAKE-SET 'AAACBDBEAAF'>
```

будет выражение

```
'CDBEAF'
```

"MAKE-SET" описывается следующим образом:

```
MAKE-SET
        E1 WX E2 WX E3 =  E1 <MAKE-SET E2 WX E3>
        E1 = E1
```

Функция "MAKE-SET" вычеркивает все вхождения каждого терма, кроме последнего. Нетрудно, однако описать функцию "MAKE-SETR", которая будет вычеркивать все вхождения терма,

кроме первого. Для этого можно воспользоваться отождествлением справа налево.

```
MAKE-SETR
        R E3 WX E2 WX E1 =  <MAKE-SETR E3 WX E2> E1
          E1 = E1
```

При желании можно описать функцию "MAKE-SET" так, чтобы при отождествлении не возникало неоднозначностей:

```
MAKE-SET
        WA E1 = <MAKE-SET- WA () E1>
        =
MAKE-SET-
        WA (E1) WA E2 = <MAKE-SET E1 WA E2>
        WA (E1) WB E2 = <MAKE-SET- WA (E1 WB) E2>
        WA (E1) = WA <MAKE-SET E1>
```

Видно, что первоначальное описание было короче и нагляднее. Кроме того, во втором описании пришлось использовать вспомогательную функцию "MAKE-SET-".

7. СПЕЦИФИКАТОРЫ ПЕРЕМЕННЫХ

Любая переменная может иметь с п е ц и ф и к а ц и ю, которая располагается между признаком типа переменной и ее индексом.

Спецификации позволяют накладывать дополнительные ограничения на множества допустимых значений переменных. Например, значением "SX" может быть любой символ, в то время как значением "S('ABC')X" могут быть только символы-литеры 'A', 'B' и 'C'.

Спецификация может иметь одну из следующих форм:

[ПУСТО]
[ИМЯ СПЕЦИФИКАТОРА]
([СПЕЦИФИКАТОР])

Таким образом, если спецификация не пуста, она представляет собой либо имя спецификатора, либо непосредственно сам спецификатор, заключенный в скобки. Между левой скобкой "(" и спецификатором, а также между спецификатором и правой скобкой ")" можно вставлять пробелы. В то же время нельзя вставлять пробелы между указателем типа и спецификацией, а также между спецификацией и индексом переменной. Во всех местах, где можно вставлять пробелы, можно также поставить знак "+" и перенести предложение на следующую строку.

Имя спецификатора выглядит так же, как символ-метка, за исключением того, что в качестве ограничителей используются не знаки "/", а знаки ":", т.е. имеет вид

:[ИДЕНТИФИКАТОР]:

Каждый спецификатор представляет собой описание некоторого множества термов. Это описание строится исходя из некоторого набора элементарных множеств, которые будут перечислены ниже. Обозначения этих элементарных множеств именуются э л е м е н т а м и.

Допустимы следующие элементы.

Во-первых, в качестве элементов могут использоваться имена других спецификаторов. В этом случае имя спецификатора обозначает множество, которое задает именуемый им спецификатор.

Во-вторых, множество, состоящее из одного символа [S] изображается самим этим символом. Таким образом, в качестве элемента может использоваться любой символ.

В третьих, имеется еще конечное множество элементов, перечисленных ниже:

S – множество всех символов;
B – множество термов вида "([E])", где [E] – произвольное об'ектное выражение;
W – множество всех термов;
F – множество символов-меток;
N – множество символов-чисел;
R – множество символов-ссылок;
O – множество символов-литер (об'ектных знаков);
L – множество прописных букв (русских и латинских);
D – множество десятичных цифр.

Последовательность элементов спецификатора, называется ц е п о ч к о й элементов. Цепочка элементов обозначает множество, которое представляет собой об'единение тех множеств, которые соответствуют элементам цепочки. Таким образом, если цепочка имеет вид

X1 X2 ... XN

то она обозначает множество

X1 + X2 + ... + XN

где "+" обозначает об'единение множеств. Например, "LD" обозначает множество букв и цифр.

Между элементами цепочки можно вставлять произвольное число пробелов. Если в цепочке элементов записано несколько символов-литер подряд, их можно слить в одну цепочку литер. Например,

'A' 'B' '' 'C'

эквивалентно

'AB''C'

В общем случае спецификатор имеет вид:

P1(Q1)P2(Q2)...PN(QN)PØ

где "РК" и "QК" – произвольные (может быть пустые) цепочки элементов спецификатора.

Множество значений, изображаемое спецификатором, вычисляется следующим образом.

Если N=Ø, т.е. спецификатор имеет вид РØ, то он изображает множество РØ.

Если N>Ø и РØ – пусто, то следует в конце спецификатора приписать элемент "W", т.е. считать РØ равным "W".

После этого значение спецификатора вычисляется рекурсивно следующим образом. Пусть R' – это множество термов, изображаемых спецификатором

P2(Q2)...PN(QN)PØ

Тогда множество термов, изображаемое спецификатором

PI(QI)P2(Q2)...PN(QN)PØ

вычисляется по формуле

R = PI + (R' – QI)

где "+" обозначает об'единение множеств, а "–" обозначает разность множеств.

Ниже приведены примеры спецификаторов. Для каждого спецификатора описано множество, которое он изображает.

'ABC' – любой из символов-литер 'A', 'B', 'C'.

('ABC') – любой терм, за исключением символов-литер 'A', 'B', 'C'.

('A') L – любая буква, за исключением буквы 'A'.

('A')L('Ø')N – любая буква, за исключением буквы 'A'

или любая цифра, за исключением цифры '∅'.

Множество значений, обозначаемое спецификатором, можно найти и другим способом. Можно рассматривать спецификатор как предикат, который определен на множестве термов и для каждого терма вырабатывает одно из двух значений: "ИСТИНА" или "ЛОЖЬ".

Допустим, что задан некоторый терм и нужно узнать: удовлетворяет он спецификатору или нет. Для этого просматриваем спецификатор слева направо, элемент за элементом, пока не встретим самый первый элемент, которому удовлетворяет проверяемый терм. После этого спецификатор дальше не просматривается и вырабатывается значение "ИСТИНА" или "ЛОЖЬ".

Если найденный элемент принадлежит ОК, т.е. находится внутри скобок, то вырабатывается значение "ЛОЖЬ", а если принадлежит РК, т.е. находится не в скобках, - то вырабатывается значение "ИСТИНА".

Если же мы дошли до конца спецификатора и не нашли ни одного элемента, которому удовлетворял бы проверяемый терм, то следует посмотреть, чем заканчивается спецификатор. Если он оканчивается на правую скобку, вырабатывается значение "ИСТИНА", в противном случае - "ЛОЖЬ".

В соответствии с принятыми соглашениями пустой спецификатор обозначает пустое множество, а спецификатор вида "()" - множество всех термов.

Элементарные множества, обозначаемые элементами спецификатора, обладают следующим свойством: для любых двух элементарных множеств X1 и X2 либо пересечение X1 и X2 пусто, либо X1 - подмножество X2, либо X2 - подмножество X1. Поэтому можно доказать, что если два спецификатора представляют множества R1 и R2, то можно представить некоторыми спецификаторами также R1+R2, R1*R2 и R1-R2, где "+", "*" и "-" обозначают об'единение, пересечение и разность множеств соответственно. Таким образом, множество спецификаторов замкнуто относительно теоретико-множественных операций.

Имена спецификаторов описываются с помощью ключевого слова "S". Эти описания имеют следующий вид:

```
[ID]     S  [SP]
```

где [ID] - имя спецификатора без ограничителей ":", а [SP] - спецификатор. Например:

```
ADDOP    S  '+-'
MULTOP   S  '*/'
```

Теперь, имена :ADDOP: и :MULTOP: можно употреблять в качестве спецификаций и в качестве элементов других спецификаторов. Например, значением переменных "S:ADDOP:X" и "S(:ADDOP:)Y" может быть только '+' или '-'.

Теперь опишем подробно, какой смысл имеют спецификации для переменных различных типов.

Спецификация вида ":ID:" равносильна спецификации "(:ID:)". Например, переменная "S:ZZZ:I" имеет то же множество допустимых значений, что и переменная "S(:ZZZ:)I".

Если спецификатор задан для S- или W-переменной, то это означает, что значение переменной должно принадлежать множеству, которое описывает спецификатор.

Если спецификатор задан для VE-переменной, то это означает, что каждый терм значения переменной, стоящий на нулевом уровне скобочной структуры, должен удовлетворять спецификатору.

Например, E('+-')X - это последовательность (может быть пустая) из литер '+' и '-', E(B)X - это выражение вида (E1)(E2)...(EN) , а S(L)X E(LD)Y - это идентификатор.

В то же время, E(('+-'))X - это выражение, которое не содержит на нулевом уровне скобок ни одной литеры '+' или '-'. Например, в качестве значения годится ('+')('-') , но не годится '+'('-') .

Если у переменной есть несколько вхождений в левую часть, то у каждого вхождения может быть своя спецификация.

Во время отождествления значение к а ж д о г о вхождения должно удовлетворять спецификации э т о г о вхождения. Таким образом, получается, что множество допустимых значений переменной - это пересечение множеств допустимых значений ее вхождений. Например:

```
S(('A'))X S(('B'))X = SX
```

равносильно

```
S(('AB'))X SX = SX
```

Все спецификации, которые заданы для вхождений переменных в правую часть предложения - игнорируются.

На использование имен спецификаторов наложено следующее ограничение: если имя некоторого спецификатора используется при описании другого спецификатора, то оно должно быть описано раньше.

Благодаря этому ограничению запрещаются циклические определения вроде

```
SPC1    S  :SPC2:
SPC2    S  :SPC1:
```

Приведем примеры использования спецификаторов.

Функция "IDENT" отщепляет от аргумента слева идентификатор максимальной длины.

```
IDENT
        S(L)X EI = <IDENT1 (SX) EI>
        EI = '*' EI
IDENT1
        (EI) S(LD)A EI = <IDENT1 (EI SA) EI>
        (EI) EI = (EI) EI
```

Если использовать отождествление справа налево и специфицированные E-переменные, ту же функцию можно записать короче:

```
IDENT    R S(L)X E(LD)Y EI = (SX EY) EI
           EI = '*' EI
```

По семантике Рефала в первом предложении нужно подобрать самое короткое "EI", при котором возможно отождествление. Но ведь самому короткому "EI" соответствует самое длинное "EY"!

Компилятор Рефала распознает такие случаи, и вместо того, чтобы несколько раз удлинять значение "EI", он сразу же, слева направо наберет максимально возможное "EY".

Еще пример. Функция "ERASE-BL" просматривает цепочку символов и заменяет каждую группу из нескольких последовательно идущих пробелов на один пробел.

```
ERASE-BL
        EI ' ' E2 = EI ' ' <ERASE-BL- E2>
        EI = EI
ERASE-BL-
        R E(' ')X EI = <ERASE-BL EI>
```

8. ОБЩАЯ СТРУКТУРА ПРОГРАММЫ

В системе программирования Рефал-2 тексты исходных рефал-программ подготавливаются в виде последовательных файлов, которые хранятся на машинных носителях. Рефал-программы можно либо создавать с помощью редакторов текстов, либо набивать на перфокартах и вводить их в машину.

В любом случае рефал-программа представляет собой последовательность з а п и с е й.

Если текст программы набивается на перфокартах, то каждая запись занимает одну перфокарту. Если подпрограмма создается редактором текстов или выводится на печать, каждая запись располагается на отдельной печатной строке.

Каждая запись представляет собой последовательность из 80 литер. Рефал-система использует только первые 72 позиции. Позиции с 73 по 80 игнорируются и обычно используются для нумерации записей.

Все записи делятся на два класса: к о м м е н т а р и и и д и р е к т и в ы.

Записи-комментарии начинаются с литеры "*" (перед которой разрешается вставлять от 1 до 70 пробелов).
В позициях после "*" они содержат произвольную информацию. Эти записи игнорируются рефал-системой и не влияют на смысл рефал-программы.

Записи, которые не являются записями-комментариями, содержат директивы.

Рефал-программа представляет собой последовательность д и р е к т и в. Каждая директива занимает одну или несколько рядом расположенных записей. Пока что будем предполагать, что каждая директива занимает отдельную запись. Правила переноса директив с одной записи на другую будут описаны ниже.

Все директивы имеют следующий вид:

[ID] [KEY] [INF]

где [ID] - идентификатор, [KEY] - ключевое слово, которое представляет собой цепочку литер, не содержащую пробелов, [INF] - информация, которая зависит от типа директивы.

Присутствие всех трех компонентов директивы [ID], [KEY] и [INF] - не обязательно: часть из них может отсутствовать. [ID] отделяется от [KEY] одним или несколькими пробелами.

Если [ID] опущен, запись должна начинаться хотя бы с одного пробела. [KEY] отделяется от [INF] одним или несколькими пробелами. Если [KEY] опущено, а [INF] начинается с буквы, перед [INF] должен быть хотя бы один пробел.

Директива, в которой отсутствуют и [ID], и [KEY], и [INF] (другими словами, состоящая из одних пробелов), является п у с т о й. Пустые директивы не влияют на смысл рефал-программы и могут использоваться для улучшения ее внешнего вида.

В качестве ключевых слов допускаются следующие цепочки литер:

```
EMPTY
END
ENTRY
EXTRN
L
R
S
SWAP
START
```

Основное место в рефал-программах занимают п р е д л о ж е н и я, которые служат для описания функций и с которыми мы познакомились в предыдущих разделах. Предложения представляют собой директивы с ключевым словом "L" или "R", причем ключевое слово "L" разрешается опускать.

Другой известный нам тип директив - это S-директивы, которые имеют ключевое слово "S" и служат для описания спецификаторов.

Директива, в которой присутствует [ID], но опущены [KEY] и [INF] является признаком начала описания новой функции. А именно, все последующие директивы-предложения, вплоть до начала описания следующей функции, считаются относящимися к функции [ID].

Все прочие директивы будут описаны в последующих разделах.

В заключение опишем правила переноса директив с одной записи на другую.

Для перехода с одной записи на другую имеется два способа:

- литера "+" в любой позиции, в которой допустимо вставить пробел;

- любая литера, отличная от пробела, в 72 позиции.

Назовем элементами предложения следующие об'екты: цепочку об'ектных знаков (заключенную в апострофы), составной символ, свободную переменную, знаки "<", ">", "=", "(", ")".

Между любыми двумя элементами предложения разрешается вставлять произвольное число пробелов. Кроме того, разрешается вставлять пробелы и между элементами спецификатора и скобками, входящими в спецификатор. Например, "('A')LD" эквивалентно " ('A') L D ". В то же время, нельзя вставлять пробелы между признаком типа переменной и спецификацией, а также между спецификацией и индексом переменной.

Теперь правило переноса с помощью "+" формулируется следующим образом: в любом месте, где могут быть вставлены дополнительные незначащие пробелы, можно вставить знак "+" и продолжить директиву с любой позиции следующей записи.

П р и м е р. Функцию

```
FUNC    E1 '+' E2 = (E1) '+' (E2) <PSI>
        S(('+-')ON)X = SX
        E1 = E1
```

можно записать следующим образом:

```
FUNC +
 EI      +
   '+'   +
     E2  +
    = +
      ( EI +
        ) '+' (E2) <PSI>
          S(   +
          (    +
          '+-' +
          )    +
          0    +
          N    +
          )X = SX
          EI =EI
```

Второй способ переноса - любая литера, отличная от пробела, в 72 позиции.

В этом случае (если перенос не был уже сделан с помощью знака "+") первая позиция следующей записи считается непосредственно следующей за 71 позицией текущей записи. Таким образом, директива может быть "разрезана" в любом месте.

Второй способ переноса особенно удобен при автоматической генерации рефал-программ, так как можно сначала породить директиву нужного размера, а затем нарезать ее на куски размером в 71 литеру.

9. ПУСТЫЕ ФУНКЦИИ

В некоторых случаях возникает желание описывать пустые функции, т.е. функции, описания которых содержат нулевое число предложений. Эти функции имеют пустую область определения, т.к. при любом обращении к такой функции возникает останов "отождествление невозможно".

Пустые функции обычно бывают полезны, если нужны символы, которые заведомо отличаются от всех остальных и которые имеют удобные для человека графические представления.

Пустые функции могут быть описаны с помощью директивы "EMPTY", которая имеет следующий вид:

EMPTY [ИДЕНТ1],[ИДЕНТ2],...,[ИДЕНТN]

где [ИДЕНТ1], [ИДЕНТ2], ..., [ИДЕНТN] - имена определяемых пустых функций. Например:

EMPTY ALPHA,PSI

Пустые функции можно описывать и еще одним способом: с первой позиции записывается идентификатор, а в следующих позициях директивы остаются пробелы. Ключевое слово, таким образом, в этом случае опускается. Например:

```
ALPHA
PSI
```

10.МОДУЛИ

Часто бывает удобно разбить рефал-программу на части, которые могут обрабатываться компилятором Рефала независимо друг от друга.

Наименьшая часть рефал-программы, которая может быть обработана компилятором независимо от других, называется м о д у л е м.

Результат компиляции исходного модуля на Рефале представляет собой о б ' е к т н ы й модуль, который перед исполнением рефал-программы должен быть об'единен с другими модулями, полученными компиляцией с Рефала или других языков. Это об'единение выполняется с помощью редакторов связей и

загрузчиков. Детали зависят от используемой операционной системы.

Исходный рефал-модуль должен начинаться с директивы "START" и кончаться директивой "END". Эти директивы имеют следующий вид:

```
[ИМЯМОД] START
         END
```

где [ИМЯМОД] - имя модуля, которое является цепочкой латинских букв и цифр длиной не более 8, начинающаяся с буквы. Имя модуля может быть опущено.

Функции, описанные в разных модулях, могут обращаться друг к другу. Если в некотором модуле используется функция, которая описана в другом модуле, эту функцию следует об'явить в н е ш н е й по отношению к данному модулю с помощью директивы "EXTRN".

Если в некотором модуле описана функция, к которой есть обращения из других модулей, эта функция должна быть об'явлена в х о д н о й т о ч к о й данного модуля с помощью директивы "ENTRY".

Директивы "ENTRY" и "EXTRN" имеют следующий вид:

```
ENTRY [Ф1],[Ф2],...,[ФN]
EXTRN [Ф1],[Ф2],...,[ФN]
```

где [Фi] - описание входной точки или внешней метки соответственно.

[Фi] может иметь одну из двух следующих форм:

```
[ИДЕНТИФИКАТОР]
[ИДЕНТИФИКАТОР]([ВНЕШНИЙ ИДЕНТИФИКАТОР])
```

где [ВНЕШНИЙ ИДЕНТИФИКАТОР] - это последовательность латинских букв и цифр длиной не более 8, начинающаяся с буквы.

Если кроме имени функции задан еще внешний идентификатор, это означает, что за пределами модуля (в среде, "окружающей модули") функция имеет "внешнее" имя, которое отличается от "внутреннего" имени функции, употребляемого внутри модуля. Таким образом одна и та же функция может иметь разные внутренние имена во всех модулях, в которых она используется, но ровно одно, одинаковое для всех модулей внешнее имя.

Если при описании функции в директиве "ENTRY" или "EXTRN" внешнее имя не задано, то считается, что внешнее имя совпадает с внутренним.

Ограничения, налагаемые операционными системами, таковы, что внешние имена могут содержать только латинские буквы и иметь длину не более 8. Поэтому, в тех случаях, когда внутреннее имя содержит русские буквы, отличные от латинских или имеет длину более 8 литер, следует указывать внешнее имя.

П р и м е р. Пусть имеется два модуля "M1" и "M2". В модуле "M1" описана функция "COMMUNICATION", а в модуле "M2" - функция "DREAM" и пусть эти функции используются в модулях "M2" и "M1" соответственно. Тогда эти модули могут иметь следующую структуру:

```
M1          START
            ENTRY COMMUNICATION(COMMUN)
            EXTRN DREAM
COMMUNICATION  E1 '+' E2 = +
                  <DREAM E1> <DREAM E2>
            END
M2          START
            ENTRY DREAM
            EXTRN ОБЩЕНИЕ(COMMUN)
DREAM       SX E1 = SX <ОБЩЕНИЕ E1>
            END
```

Здесь функция, которая имеет внутреннее имя "COMMUNICATION" в модуле "M1", имеет внутреннее имя "ОБЩЕНИЕ" в модуле "M2" и внешнее имя "COMMUN".

С помощью директив "ENTRY" и "EXTRN" можно об'являть входными и внешними не только имена функций, но и имена спецификаторов.

Каждое имя, которое употребляется внутри модуля, должно быть описано либо как имя внутренней функции или спецификатора, либо как имя внешней функции или спецификатора в директиве "EXTRN".

II. ПЕРВИЧНЫЕ ФУНКЦИИ

Во многих случаях возникает необходимость из программ, написанных на Рефале вызывать программы, написанные на других языках или непосредственно в командах машины. Эта необходимость возникает, в частности, в тех случаях, когда требуется выполнять операции ввода/вывода, операции над числами и другие, которые рефал-машина "не умеет" выполнять непосредственно.

Функции, описанные не на Рефале, которые, тем не менее, можно вызывать обычным способом из программ, написанных на Рефале, называются п е р в и ч н ы м и.

Собственно говоря, с точки зрения рефал-модуля первичные функции - это просто некоторые функции, внешние по отношению к данному модулю, поэтому, вызывая какую-либо функцию можно даже не знать, что это - первичная функция или функция, написанная на Рефале. Разница состоит только в том, что исполнение вызова первичной функции занимает только один шаг с точки зрения рефал-машины, в то время как исполнение вызова обычной функции может занять несколько шагов.

Набор первичных функций, предоставляемых в данной реализации, будет описан отдельно. Помимо этих первичных функций пользователь может писать и собственные на языке PL/I или на языке ассемблера. Как это сделать будет описано отдельно.

12.КОПИЛКА

До сих пор мы считали, что рефал-машина имеет два запоминающих устройства: поле памяти и поле зрения. В действительности у нее имеется еще одно запоминающее устройство: к о п и л к а, доступ к которому возможен с помощью первичных функций "BR", "DG", "CP", "RP", "DGALL".

Имена этих функций имеют следующий смысл: "BR" - закопать (BURY), "DG" - выкопать (DIG OUT), "CP" - скопировать (COPY), "RP" - заменить (REPLACE).

Содержимое копилки всегда имеет следующий вид:

([X1]'='[Y1]) ([X2]'='[Y2]) ... ([XN]'='[YN])

где [X1], [X2], ..., [XN] и [Y1], [Y2], ..., [YN] - произвольные об'ектные выражения. Смысл содержимого копилки - следующий. [Xi] - есть имя выражения [Yi].

Перед началом работы программы копилка содержит пустое выражение.

Функции "BR", "DG", "RP" и "CP" предназначены для перемещения выражений из поля зрения в копилку и обратно. Обращения к ним имеют следующий вид:

```
<BR [X] '=' [Y]>
<DG [X]>
<RP [X] '=' [Y]>
<CP [X]>
```

где [Y] - произвольное выражение, а [X] - произвольное выражение, не содержащее символа '=' на нулевом уровне скобочной структуры.

Ф у н к ц и я "BR" (закопать).

При обращении к функции "BR" терм "([X] '=' [Y])"

добавляется к копилке слева, т.е. копилка преобразуется следующим образом:

[E] → ([X] '=' [Y]) [E]

где [E] - содержимое копилки до обращения к "BR". Результат обращения к "BR" - пусто.

Можно закапывать несколько выражений под одним и тем же именем. Например, в результате выполнения

<BR 'X=A'> <BR 'X=B'>

копилка преобразуется следующим образом:

[E] → ('X=B')('X=A') [E]

Функция "DG" (выкопать).

Функция "DG" просматривает копилку слева направо в поисках терма вида "([X] '=' [Z])" и, если находит, удаляет его из копилки и выдает [Z] в качестве результата замены. Это можно изобразить следующим образом:

Поле зрения: <DG [X]> → [Z]
Копилка: [E1] ([X] '=' [Z]) [E2] → [E1] [E2]

Если в копилке несколько выражений закопаны под одним именем, то выкапывается самое левое из них, т.е. то, которое закапывалось последним. При повторном обращении к "DG" с тем же аргументом [X] будет выкопано выражение, закопанное предпоследним и т.д.

Если "DG" не находит в копилке нужного терма, она выдает в качестве результата замены пустое выражение.

Функция "CP" (скопировать).

Функция "CP", так же, как и функция "DG", находит в копилке выражение по имени и выдает его в качестве

результата замены, но копилка при этом не изменяется, т.к. в поле зрения формируется копия выражения. Таким образом "CP" работает так:

```
Поле зрения:  <CP [X]>  →  [Y]
Копилка:      [E1] ([X] '=' [Y]) [E2]  →
                    [E1] ([X] '=' [Y]) [E2]
```

Если под именем [X] ничего не закопано, "CP" выдает "пусто". Функцию "CP" можно было бы следующим образом описать через "BR" и "DG":

```
CP      EX = <CP1 (EX) <DG EX>>
CP1     (EX) EY = EY <BR EX '=' EY>
```

Это описание несколько отличается от алгоритма, реализованного в первичной функции "CP", тем, что терм "([X] '=' [Y])" переставляется в начало копилки, в то время как первичная функция оставляет его на месте.

Ф у н к ц и я "RP" (заменить).

Функция "RP" добавляет в копилку новое выражение и выбрасывает выражение, закопанное в последний раз под тем же именем.

```
Поле зрения:  <RP [X] '=' [Y]>  →  [ПУСТО]
Копилка:      [E1] ([X] '=' [Z]) [E2]  →
                    [E1] ([X] '=' [Y]) [E2]
```

Если под именем [X] ничего не закопано, то функция "RP" делает то же самое, что и "BR".

Эквивалентное описание на Рефале имеет вид:

```
RP      EX '=' EY = <RP1 <DG EX>> <BR EX '=' EY>
RP1     E1 =
```

Ф у н к ц и я "DGALL" (выкопать все).

Функция "DGALL" позволяет вынуть из копилки все содержимое и поместить его в поле зрения. Обращение к "DGALL" имеет вид:

<DGALL>

Пусть копилка содержит выражение [E]. Тогда результатом обращения к "DGALL" будет [E] , причем в копилке останется пустое выражение.

Выражение [E] можно вернуть в копилку. Для этого достаточно описать на Рефале функцию "BRALL"

```
BRALL    E1 (E2) = <BR E2> <BRALL E1>
         =
```

После этого следует обратиться к "BRALL" следующим образом:

<BRALL [E]>

13.СТАТИЧЕСКИЕ И ДИНАМИЧЕСКИЕ ЯЩИКИ

Об'ектами обработки для программ, написанных на Рефале, являются выражения. Выражение представляет собой по существу способ представления древовидных структур в виде одномерных цепочек символов и скобок.

При решении некоторых задач, однако, оказывается, что требуется обрабатывать структуры данных, которые сложнее, чем древовидные.

Конечно, в принципе, любые конструктивные об'екты можно представить в виде деревьев, однако это не всегда удобно, а иногда приводит к существенному замедлению работы программы

(в тех случаях, когда прямой доступ к данным приходится моделировать с помощью ассоциативных поисков).

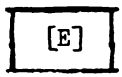
Средством Рефала-2, дающим возможность обрабатывать произвольные графы, являются статические и динамические ящики.

До сих пор предполагалось, что рефал-машина состоит из трех запоминающих устройств: поля памяти, в котором находится набор предложений, поля зрения и копилки. Теперь будем считать, что имеется еще потенциально бесконечное множество запоминающих устройств, называемых ящиками. Каждый ящик содержит произвольное об'ектное выражение, которое может изменяться в процессе работы. Это выражение мы будем называть содержимым ящика.

Каждому ящику соответствует функция, с помощью которой можно получить доступ к содержимому ящика. Эти функции мы будем называть обменными.

Таким образом, имеется взаимно-однозначное соответствие между множеством обменных функций и множеством ящиков. Обменная функция, соответствующая некоторому ящику, будет называться также именем этого ящика.

Наглядно взаимосвязь между ящиком и обменной функцией можно изобразить следующим образом.



где "FUNC" - имя обменной функции, а [E] - содержимое ящика.

Обменные функции работают следующим образом.

После вычисления терма

<FUNC [E']>

где "FUNC" – имя ящика, в поле зрения останется выражение [E] – содержимое ящика с именем "FUNC", а выражение [E'] станет содержимым ящика. Таким образом, происходит обмен информацией между полем зрения и ящиком (откуда и произошло название обменных функций).

Все ящики делятся на с т а т и ч е с к и е и д и н а м и ч е с к и е. Статические ящики существуют в течение всего времени выполнения программы и не могут ни порождаться, ни уничтожаться во время работы. Напротив, динамические ящики порождаются только во время работы и могут уничтожаться.

Все с т а т и ч е с к и е ящики должны быть описаны в программе. Для этого используются директивы "SWAP", которые выглядят следующим образом:

SWAP [ИДЕНТ1],[ИДЕНТ2],...,[ИДЕНТN]

т.е., пропустив один или несколько пробелов, следует записать ключевое слово "SWAP", затем пропустить один или несколько пробелов и перечислить через запятую имена обменных функций.

Таким образом, статические ящики описываются одновременно со своими обменными функциями. Перед началом работы программы все статические ящики содержат пустые выражения.

П р и м е р. Рассмотрим следующий фрагмент программы:

```
        SWAP  X1,X2
SWX1X2  = <X1 'A'> <X2 'B'> <SW2 /X1/ /X2/>
SW2     SX SY = < SX < SY < SX>>>
```

В процессе вычисления обращения

<SWX1X2>

в ящики "X1" и "X2" сначала занесутся символы 'A' и 'B' соответственно. Затем, в результате вычисления терма

```
<SW2 /X1/ /X2/>
```

содержимые ящиков поменяются местами. То есть в ящике "X1" окажется символ 'B' , в ящике "X2" - символ 'A' , а в поле зрения останется пустой результат замены.

Д и н а м и ч е с к и е ящики порождаются в процессе работы программы первичной функцией "NEW".

Ф у н к ц и я "NEW".

В результате вычисления терма

```
<NEW [E]>
```

создается новый ящик и в него помещается выражение [E]. Одновременно порождается новый символ-ссылка [R], который остается в поле зрения в качестве результата замены. Символ [R] является именем новой обменной функции, обеспечивающей доступ к созданному ящику.

Имена динамических ящиков являются символами особого типа - символами-ссылками. В отличие от имен статических ящиков, являющихся обычными символами-метками, символы-ссылки нельзя употреблять в виде констант в рефал-программах. Тем не менее, при отладке рефал-программ возникает необходимость как-то печатать символы-ссылки. Они печатаются в виде

```
/%FFFFFF/
```

где "FFFFFF" - шесть шестнадцатеричных цифр, обладающих следующими свойствами:

- в каждый момент работы программы различным символам-ссылкам соответствуют различные "FFFFFF"

- один и тот же символ-ссылка имеет одно и то же "FFFFFF" на протяжении одного запуска программы

Эти свойства выполняются в силу того, что "FFFFFF"

представляет собой адрес места, в котором расположен динамический ящик в памяти машины.

П р и м е р. Рассмотрим следующий фрагмент программы, аналогичный приведенному в предыдущем примере.

```
        EXTRN NEW
SWRIR2  = <SW2 <NEW 'A'> <NEW 'B'>>
SW2     SX SY = < SX < SY < SX >>>
```

Теперь, в результате вычисления терма "<SWRIR2>" будут выполнены два обращения к функции "NEW". В результате чего образуются два ящика, которые могут иметь, например, следующие имена: "/%0706I3/" и "/%053024/". Ящик "/%0706I3/" будет содержать символ 'A', а ящик "/%053024/" - символ 'B'. Затем функция "SW2" воспользуется символами-ссылками, оставшимися в поле зрения, и поменяет местами содержимое этих ящиков.

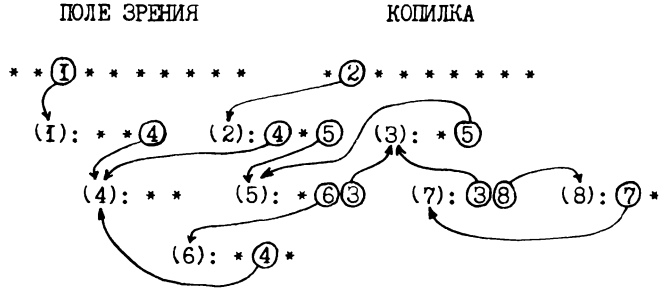
Обратите внимание на следующий факт: обращение к функции "SWRIR2" привело к появлению двух новых ящиков. Можно ли теперь как-нибудь извлечь содержимое этих ящиков. Очевидно, что нельзя. Ни в поле зрения, ни в копилке, ни в других ящиках не сохранилось имен этих ящиков. Поэтому дальнейшая работа программы не изменится, если эти ящики уничтожить.

Ясно, что если обращаться к функции "SWRIR2" много раз, то память рефал-машины будет забиваться ненужными ящиками. Конечно, для абстрактной рефал-машины это не имеет никакого значения, ибо ее память потенциально бесконечна, но память реальной вычислительной машины рано или поздно должна исчерпаться. Поэтому во всех реализациях Рефала-2 предусмотрен механизм с б о р к и м у с о р а.

Сборка мусора автоматически запускается каждый раз, когда исчерпается свободная память. При этом обнаруживаются и удаляются все ящики, к которым невозможно добраться прямо или косвенно из поля зрения, копилки или статических ящиков.

На следующем рисунке изображены поле зрения, копилка и

динамические ящики, пронумерованные цифрами от 1 до 8. Звездочки изображают некоторые элементы выражений, которые не являются символами-ссылками. Символы-ссылки обозначены цифрами.



Видно, что по ссылке из поля зрения можно добраться до ящика 1, а из него - до ящика 4. Из копилки можно непосредственно добраться до ящика 2 и косвенно (через ящик 2) до ящиков 4, 5, 6, 3. Таким образом, нет способа извлечь информацию из ящиков 7 и 8. Если в этот момент запустить сборку мусора, то ящики 7 и 8 будут уничтожены. Если теперь убрать символ-ссылку из поля зрения, то станет недоступным и ящик 1. Если же символ-ссылку в поле зрения оставить, но убрать ссылку из копилки, то окажутся ненужными все ящики, кроме 1 и 4.

Опишем теперь пять первичных функций, которые, хотя и не дают ничего принципиально нового, часто оказываются удобными.

Ф у н к ц и я "GTR" (взять по ссылке).

Извлекает содержимое ящика. Результатом замены при вычислении терма

<GTR [X]>

где [X] - имя статического или динамического ящика, является содержимое ящика. При этом в ящике остается пустое выражение.

Ф у н к ц и я "RDR" (прочитать по ссылке).

Как и "GTR" выдает содержимое ящика в поле зрения, но ящик при этом не изменяется, то есть происходит копирование его содержимого.

Ф у н к ц и я "PTR" (положить по ссылке).

Добавляет в ящик новую информацию. В результате вычисления терма

<PTR [R] [E]>

где [R] - имя ящика, а [E] - произвольное об'ектное выражение, ящик меняется так:

[EØ] → [EØ] [E]

где [EØ] - старое содержимое ящика. Результатом замены является пустое выражение.

Ф у н к ц и я "WTR" (записать по ссылке).

Помещает в ящик новую информацию, при этом старое содержимое ящика уничтожается. То есть при вычислении терма

<WTR [R] [E]>

где [R] - имя ящика, [E] - произвольное об'ектное выражение, содержимое ящика меняется так

[EØ] → [E]

где [EØ] - старое содержимое ящика. Результатом замены является пустое выражение.

Ф у н к ц и я "SWR" (обменять по ссылке).

Записывает в ящик новую информацию, а старую выдает в поле зрения. Таким образом происходит такое преобразование

Поле зрения: <SWR [R] [E]> → [EØ]
Ящик: [EØ] → [E]

где [R] – имя ящика, [E] – об'ектное выражение.

Эти функции можно было бы описать на Рефале следующим образом:

```
GTR     SX = < SX>
RDR     SX = <RDRI SX < SX>>
RDRI    SX EY = EY < SX EY>
PTR     SX EY = < SX < SX> EY>
WTR     SX EY = <WTRI < SX EY>>
WTRI    EY =
SWR     SX EY = < SX EY>
```

Отличие этих функций от соответствующих первичных функций состоит в том, что они не проверяют, что "SX" – имя ящика, поэтому их область определения шире, чем у соответствующих первичных функций.

Л И Т Е Р А Т У Р А

[БР 1977]
Базисный Рефал и его реализация на вычислительных машинах. М., ЦНИПИАСС, 1977.

[КПРТР 1975]
Ан.В.Климов, Л.В.Проворов, С.А.Романенко, Е.В.Травкина. Рефал в мониторной системе "Дубна" БЭСМ-6. Входной язык компилятора и запуск программ. Препринт ИПМ АН СССР N 8, М., 1975.

[КР 1975]
Ан.В.Климов, С.А.Романенко. Рефал в мониторной системе "Дубна" БЭСМ-6. Интерфейс рефала и фортрана. ИПМ АН СССР, М., 1975.

[КРТ 1972]
Ан.В.Климов, С.А.Романенко, В.Ф.Турчин. Компилятор с языка рефал. ИПМ АН СССР, М., 1972.

[КРТР 1974]
Ан.В.Климов, С.А.Романенко, Е.В.Травкина. Инструкция по работе с мониторной системой "РЕФАЛ" для БЭСМ-6. ИПМ АН СССР, М., 1974.

[РКТ 1973]
С.А.Романенко, Ан.В.Климов, В.Ф.Турчин. Теоретические основы синтаксического отождествления в языке рефал. Препринт ИПМ АН СССР N 13, М., 1973.

[РТ 1970]

С.А.Романенко, В.Ф.Турчин. Рефал-компилятор. В сб. "Труды 2-й всесоюзной конференции по программированию". Новосибирск, 1970.

[Т 1966]

В.Ф.Турчин. Метаязык для формального описания алгоритмических языков. В сборнике "Цифровая вычислительная техника и программирование". Сов. Радио, 1966, с.116-124.

[Т 1968]

В.Ф.Турчин. Метаалгоритмический язык. Кибернетика, N 4, 1968, с.45-54.

[Т 1974]

В.Ф.Турчин. Базисный рефал. Описание языка и основные приемы программирования. М., ЦНИПИАСС, 1974.

[ТС 1969]

В.Ф.Турчин, В.И.Сердобольский. Язык рефал и его использование для преобразования алгебраических выражений. Кибернетика, N 3, 1969, с.58-62.

АЛФАВИТНЫЙ УКАЗАТЕЛЬ ФУНКЦИЙ

Климов Андрей Валентинович, Романенко Сергей Анатольевич." Система программирования Рефал- 2 для ЕС ЭВМ. Описание входного языка."

Редактор В.С. Штаркман. Корректор А.О. Лацис.

Подписано к печати 09. 02.87г. № Т- 04873. Заказ № 89.
Формат бумаги 60X90 1/16. Тираж 450 экз.
Объём 2,3 уч.-изд.л. Цена 20 коп.

055 (02)2

Отпечатано на ротапринтах в Институте прикладной математики АН СССР

Москва, Миусская пл. 4.

©

Цена 20 коп.