

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

# Препринты ИПМ • Препринт № 29 за 1969 г.



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

С.Н. Флоренцев, В.Ю. Олюнин, В.Ф. Турчин

Эффективный интерпретатор для языка Рефал

Статья доступна по лицензии Creative Commons Attribution 4.0 International



**Рекомендуемая форма библиографической ссылки:** Флоренцев С.Н., Олюнин В.Ю., Турчин В.Ф. Эффективный интерпретатор для языка Рефал // Препринты ИПМ им. М.В.Келдыша. 1969. № 29. 104 с.

https://library.keldysh.ru/preprint.asp?id=1969-29

# ОРДЕНА ЛЕНИНА ИНСТИТУТ ПРИКЛАДНОЙ МАТЕМАТИКИ

АКАЛЕМИИ НАУК СССР

Отдел автоматизации программирования

С.Н.Флоренцев В.Ю.Олюнин В.Ф.Турчин

# ЭФФЕКТИВНЫЙ ИНТЕРПРЕТАТОР ДЛЯ ЯЗЫКА «РЕФАЛ»

ПРЕПРИНТ № 29 за 1969год

### ОРДЕНА ЛЕНИНА ИНСТИТУТ ПРИКЛАДНОЙ МАТЕМАТИКИ АКАДЕМИИ НАУК СССР

### Отдел автоматизации программирования

### ЭФФЕКТИВНЫЙ ИНТЕРПРЕТАТОР ДЛЯ ЯЗЫКА РЕФАЛ

Зав.отделом - М.Р.Шура-Бура

Исполнители: С.Н.Флоренцев

В.Ю.Олюнин

В.Ф.Турчин

#### I. BREHEHUE

Язик РЕФАЛ (алгоритмический язык рекурсивных функций) ориентирован на задачи преобразования символьной информации (перевод с формализованных и естественных языков, алгебраи-ческие преобразования, машинное доказательство теорем и т.п.). Этот язык возник как специализированное упрощение метаалго-ритмического языка ( [ I , 2 ] ), его описание содержится в [ 3]

**Іля пратического использования машинно-независимого языка** программирования необходимо иметь транслятор с этого языка. Из формального описания семантики языка РЕФАЛ (см. 3) легко видеть, что программа, которая будет имитировать действия РЕФАЛ машины в буквальном соответствии с ее формальным описанием, окажется чрезвичайно неэффективной. Действительно, при выполнении каждого шага она будет просматривать значительную часть подя эрения в поисках ведущего символа конкретизации. затем полностью просматривать все конкретизируемое выражение (возможно, неоднократно) в процессе отождествления и. наконец, полностью переписивать вводимое в поле зрения выражение. Эти действия, повторяемые на каждом шаге, сделают работу программы чрезвычайно медленной. Между тем, по смыслу . самого алгоритма они вовсе не нужны. Следовательно такая программа, то есть транслятор-интерпретатор, буквально имитирующий абстрактную РЕФАЛ-машину. будет неэффективной и непригодной для практического использования.

С другой сторони, ясно, что составление транслятора-компилятора с такого язика, как РЕФАЛ- задача весьма сложная и, бить может, без введения каких-то ограничений неразрешимая. Псэтому важно было попытаться разработать такой интерпретатор, который свел бы потери эффективности к минимуму и дал бы возможность в сочетании с определенным стилем программирования на РЕФАЛе практически использовать этот язык как язык программирования. В настоящей работе излагаются общие принципы построения эффективного РЕФАЛ — интерпретатора для ЭВМ, описывается такой интерпретатор для машины БЭСМ—6 и приводятся некоторые сведения о результатах его эксплуатации.

# 2. Общие принципы эффективной реализации языка РЕФАЛ на вычислительных машинах.

В настоящей главе описываются основные принципы построения эффективного РЕФАЛ-интерпретатора для вичислительных машин. Эти принципы являются достаточно общими и могут применяться при построении интерпретаторов на различных машинах. Основное внимание уделялось вопросу повчтения скорости работы интерпретатора. Решающим моментом на п. ти к достижению этой цели является описываемый в настоящей главе способ расположения информации в поле зрения интерпретатора.

Алгоритм синтаксического отождествления строится так, что исключаются все лишние просмотры.

В процессе замены левой части предложения на правую исключаются все лишние переписи.

## 2. I. Расположение информации в поле зрения интерпретатора

Язики, ориентированные на символьные задачи [ 4 ] требуют динамического распределения памяти. В РЕФАЛ интерпретаторе это требование касается прежде всего распределения

памяти в поле зрения.

Проблема динамического распределения памяти решается путем списочного построения памяти, занятой под программу, и свободного поля памяти. В случае РЕФАЛ-интерпретатора этот список должен быть симметричным, поскольку язык РЕФАЛ допускает симметричное обращение со строками [ 5 ].

Симметричное построение поля зрения предполагает расположение кода символа и двух адресов, связи в одной или в двух (в зависимости от адресности машины) подряд идущих ячейках оперативного запоминающего устройства.

Однако представление поля зрения в виде симметричного списка, состоящего из однородных элементов недостаточно для построения эффективного РЕФАЛ-интерпретатора, исключающего лишние просмотры. Укажем источники лишних просмотров.

2.I.I. В процессе синтаксического отождествления РЕФАЛмашина предписывает набирать содержимое каждой переменной,
посимвольно просматривая поле зрения. Это не всегда бывает
необходимо с точки зрения самого существа айгоритма. Поясним
это на примерах.

Пусть имеется предложение, которое содержит следующую левую часть:

Процедура < в этом примере находит в процессе отождествления ближайший слева знак + в конкретизируемом выражении,
если он не расположен внутри скобок. Существенной информацией
здесь является содержимое переменной  $\not \in$  . Именно ее
содержимое указывает в каком месте расположен ближайший слева

знак + . Содержимое  $\underline{\mathcal{E}}_2$  можно было бы не просматривать, отнеся к нему оставшуюся после знака + часть конкретизируемого выражения. Однако этого сделать нельзя, т.к. нельзя найти адрес конца  $\underline{\mathcal{E}}_2$ , не просмотрев его содержимого.

Приведем еще пример:

Предположим, что данное предложение является подходящим для какого-то конкретизируемого выражения. В этом случае скоски левой части предложения оказываются совершенно жестко связанными со скобками конкретизируемого выражения. Внутри этих скобок совершенно необходимо, конечно, нафирать содержимое

— 5 , однако содержимое — 4 можно было бы и не набирать, отнеся к нему символы оставшиеся после нахождения знака + внутри скобок. Но этого сделать нельзя, так как не известен эчрес правой скобки.

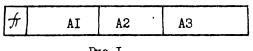
Содержимое E5 так же как и содержимое E2 в примере 1 можно было бы не просматривать, отнеся к нему оставшуюся после отождествления внутренней скобки часть конкретизируемого выражения.

В приведенных примерах происходит потеря эффективности за очёт лишних просмотров содержимого переменных типа  $\underline{\mathcal{E}}$  , не вызванных самой сутью алгоритма.

2.1.2. 'аждый раз после того, как полностью выполнена очередная конкретизация, машина ищет ведущий символ конкретизацик
в соответствии с его формальным определением. Это может быть
достигнуто только посимвольным просмотром всего поля зрения.
Это,конечно, занимает значительную часть времени выполнения ша-

Пля ликвидации указанных потерь эффективности в РЕФАЛинтерпретаторе принят следующий способ расположения информации в поле зрения.

Каждому символу поле фения отводится дри адресных поля (AI, A2, A3) и признаковая часть  $\pi$  ( 2 разряда):



Puc.I

Признак 🗡 определяет, является ли этот символ значащим символом, скобкой или символом конкретизации  $\,\mathcal{K}\,\,$  . От содержания  $\pi$  зависит содержимое  $A_1$  .  $A_2$  и  $A_3$  .

- а) Если  $\pi$  соответствует значащему символу, то в A2хранится код символа, в A1 - адрес предндущего, а в A3 последующего символа.
- б) Если  $\pi$  соответствует скобке, то в A2 хранится адрес парной скобки, в A1 и A3 - соответственно адрес предыдущего и последующего символа.
- в) Если 🛨 соответствует символу конкретизации. то в A1 хранится адрес предыдущего символа, в A5 - адрес последующего, а в A2 - адрес того символа K , который станет ведущим, когда будет полностью выполнена данная кон-. кретизация. В символе К следовало он также указать адрес соответствующей ему подчеркнутой точки, однако это потребует дополнительного адресного поля. Вместо этого полчеркнутая точка в память машины не пишется, а вся область действия символа K заключается в скобки так, что левая скобка располагается сразу же за символом K .

Точно так же с помощью адресов связи организуется свободное поле памяти.

Будем называть адреса A1 , A2 и A3 адресами связи. Использование этих адресов позволяет рассматривать поле зрения в виде сплошного массива независимо от физического расположения ячеек в памяти машины. С помощью адресов связи можно осуществлять хождение по полю зрения в обоих направлениях. Например, если поле зрения имеет вид:

то внутри машины оно будет представлено следующим образом (чертой без стрелок обозначается двусторонняя связь)

Наличие адреса связи (A2) у парных скобок и символов K позволяет построить схему интерпретатора, свободную от указанных выше потерь эффективности. Кроме этого, появляется дополнительная возможность для увеличения скорости работы блока синтаксического отождествления. Эти возможности следующие.

2.1.3. При наборе содержимого переменной типа W в том случае, когда она представляет из себя заключенное в скобки выражение, теперь нет необходимости посимвольно просматривать это выражение для нахождения заключительной скобки. Независимо от направления синтаксического отождествления, адрес конца

терма хранится в его начале.

В самом деле, пусть, например, мы имеем предложение:

И пусть конкретизируемым выражением является выражение:

$$KPABC + ((A+B)-C)+A$$
.

Отождествление данного выражения интерпретатор будет проводить следующим образом. Сначала наберется содержимое до ближайшего слева знака + . После этого интерпретатор определит, что следующей переменной является переменная типа терма. Ей в конкретизируемом выражении соответствует выражение, заключенное в скобки. Адрес начала и конца такого терма хранится в ячейке, соответствующей левой скобке. Дальнейшее движение по полю зрения начнется с символа, стоящего после правой заключительной скобки.

2.І.4. С помощью переменной типа  $\mathcal{E}$  в РЕФАЛе часто организуются алгоритми просмотра конкретизируемого виражения с целью определения местоположения какого-то конкретного симвода в конкретизируемом виражении см. [5]. В дальнейшем такой символ ми будем называть опорним символом. При этом типичной конструкцией левой части является.

В данном примере сеть как раз тот символ, который ищется в поле зрения. Согласно синтаксису языка РЕФАЛ ясно, что с помощью такой записи можно найти ближайший слева символ сето конкретизируемого выражения. При этом скобочные структуры следующего уровня просто бессмысленно просматривать.

РЕФАЛ-интерпретатор, встречая такие скобочные структуры, перескакивает через них, используя адреса связи парных скобок.

Пусть в нашем примере конкретизируемое выражение будет  $K \Psi A (A \vee B) C B (AB (CD \vee E)) \sim ()$ 

РЕФАЛ-интерпретатор набор содержимого  $\mathcal{E}_{\ell}$  будет проводить следующим образом: сравнит символ  $\mathcal{A}$  с  $\ll$  , перепритнет через скобки  $\binom{1}{2}$  , сравнит с  $\ll$  символы  $\mathcal{C}$  и  $\mathcal{B}$ , перепритнет через скобки  $\binom{1}{2}$  , так как следующий символ совпадает с опорным, все просмотренное выражение отнесет к  $\mathcal{E}_{\ell}$  .

Из приведенного примера видно, что РЕФАЛ-интерпретатор производит действия, буквально совпадающие с самим существом алгоритма просмотра, не делая никаких лишних просмотров.

2.1.5. Покажем теперь как ликвидируются лишние просмотры и переписи, указанные в пунктах 2.1.1 и 2.1.2.

Обратимся к первому примеру . іосле набора  $\underline{\mathcal{E}}^1$  содержимое  $\underline{\mathcal{E}}^2$  теперь становится известным. Оно заключено между знаком + и правой заключительной скобкой конкретизируемого выражения, адрес которой известен . Этот адрес содержится в левой скобке, стоящей сразу за символом  $\underline{\mathcal{K}}$  . В процессе синтаксического отождествления этот адрес можно хранить в рабочей ячейке. Точно так же определяется содержимое  $\underline{\mathcal{E}}^4$  во

х/ Для н°глядности пояснения в данном случае и инсгда в дальнейшем мы будем помечать скобки индексами, которые разумеется не входят в РЕФАЛ-выражение.

втором примере • Здесь адрес правой скобки определяется при входе внутрь встретившейся скобочной структуры.

Эти два примера иллюстрируют общий принцип РЕФАЛ-интерпретатора — не просматривать содержимое переменных типа  $\underline{\varepsilon}$  в тех случаях, когда оно не вызвано самой сутью работы алгорит—ма.

Что касается потерь времени, о которых указывалось в 2.1.2, то их также удается устранить за счёт адреса связи у символов  $\mathcal{K}$ . Для этого в РЕФАЛ-интерпретаторе создан специальный алгоритм связи символов  $\mathcal{K}$  между собой, который включается один раз при первой записи информации в поле зрения и каждый раз при выполнении шага машины, если в правой части очередного предложения встречаются символы  $\mathcal{K}$ . В дальнейшем, при описании препроцессора (см. раздел 3.2) мы подробно опишем этот алгоритм, а пока на примерах покажем результаты его работы.

Пусть в поле зрения интерпретатора вводится выражение (в примере указаны только символы  $\underline{K} \ \mbox{$\mathscr{U}$} \ \underline{\ }$  )

$$K_1$$
  $K_2$   $K_3$   $K_4$   $\dot{}$ 

Интерпретатор с помощью адресов связи A2 организует все символи K в список  $K_3$   $K_4$   $K_2$   $K_1$  так что слева всегда нахощится ведущий символ конкретизации, следом за ним следует тот символ K, который станет ведущим, когда будет полностью выполнена предыдущая конкретизация и  $T_0$ . Интерпретатор в процессе работы все время "смотрит" на левый край списка. Изменяться список может также только с левого края.

Пусть, например, в правой части предложения, которое оказалось подходящим для области действия символа  $\mathcal{K}_5$ , стоит выражение:

В этом случае наш список будет содержать такую последовательность символов  $\mathcal K$  :

Даже, не уточняя самого алгоритма построения списка символов K, очевидно, что его нельзя построить, не просматривая всего выражения, куда эти символы входят. Однако в процессе замены левой части предложения на правую этот просмотр значительно сокращается, так как внутри переменных, стоящих в правой части предложения, символов K не может быть. В дальнейшем мы увидим ,что при замене левой части предложения на правую содержимое переменных, стоящих в правой части, не просматривается. И для связи символь K просматриваются только управляющие и жначащие символы, число которых, естественно, не может быть велико.

Например, пусть имеется следующее предложение:

и пусть конкретизируемое выражение есть

содержащее 1000 символов, просматриваться не будут.

После того, как в списке не останется ни одного символа  ${\cal K}$  , интерпретатор заканчивает работу.

### 2.2. Отождествление скобочных структур

Понятие скобочной структуры в РЕФАЛе имеет обычный смысл и отражает процесс построения выражения с помощью скобок из ссставляющих его подвыражений.

Скобочная структура имеет ряд уровней. Самый внешний уровень будем называть нулевым.

Например, скобочная структура

$$\lambda(E1+E2)+E3$$

имеет два уровня (нулевой и первый).

В левых частях предложений могут встретиться скобочные структуры сколь угодно сложные и это самым существенным образом влияет на алгоритм синтаксического отождествления.

В п. І.І. уже говорилось о том, что в примерах І и 2 для набора содержимых  $E^2$  и  $E^4$  нет необходимости просматривать содержимое поле эрения. Что касается переменной  $E^2$  в примере І, то осуществить такой алгоритм весьма просто. В примере 2 переменная  $E^4$  расположена на первом уровне внешней скобочной структуры. И для того, чтобы упростить алгоритм синтаксического отождествления, в РЕФАЛ-интерпретаторе создан единый алгоритм отождествления скобочной структуры, независимо от того, является ли данная скобочная структура самой внешней или содержащейся на каком-то внутреннем уровне внешней скобочной структуры. Ниже мы опишем этот алгоритм, а пока введем несколько определений, необходимых в дальнейшем.

### Определение І.

Если на нулевом уровне скобочной структуры нет ни одной переменной  $\not \sqsubseteq$  либо она всего одна, то такая скобочная структура называется закрытой. И переменную  $\not \sqsubseteq$  в такой структуре (если она присутствует) мы также будем называть закрытой. Если таких переменных несколько, то скобочная структура называется открытой. И все переменные  $\not \sqsubseteq$  , кроме последней (правой при отождествлении слева направо и левой — справа налево) мы будем называть открытыми. Последняя переменная  $\not \sqsubseteq$  называется закрытой.

В примере I левая часть является открытой скобочной структурой (напомним, что символ K и  $\sim$  интерпретатором понимаются как левая и правая скобка). В примере 2 внешняя скобочная структура является закрытой, а внутренняя — открытой.

Приведем еще пример сложной скобочной структуры в левой части предложения:

В данном примере самая внешняя скобочная структура — откритая, ( ... ) — закритая, ( ... ) — закритая, ( ... ) — закритая, ( ... ) — закритая. Откритыми переменными являются  $E \cap E$   $E \cap E$  ,  $E \cap E$  .

Тот факт, что интерпретатор различает два типа переменных E, позволяет ему избегать лишних просмотров поля зрения для набора содержимого закрытой переменной E. В этом случае интерпретатор, дойдя до переменной E, обращается в скобке

закривающей соответствующую скобочную структуру, меняет направление отождествления на противоположное и после того, как в левой части предложения вновь дойдет до закрытой переменной  $\underline{\mathcal{E}}$ , оставшуюся часть поля зрения относит к содержимому указанной переменной. Просмотра конкретизируемого выражения для набора открытой переменной  $\underline{\mathcal{E}}$  не избежать. Однако, при этом все встретившиеся скобочные структуры конкретизируемого выражения обходятся за один шаг с помощью адресов связи у парных скобок (см.  $\mathbf{n}$ 21.4).

Приведем пример. Пусть имеется предложение

и конкретизируемое выражение

$$K(YABC+(...)(...)AD(...))$$

Отождествление данного выражения будет проводиться следующим образом.

Сначала при движении слева направо будет определено, что содержимим  $E^1$  является ABC. После этого интерпретатор, определив, что  $E^2$  является закрытой переменной, обратиться к правой заключительной скобке  $\mathcal{L}$  и начнет отождествление справа налево. К переменной  $W^4$  отойдет содержимое скобок  $\mathcal{L}$  , переменная  $\mathcal{L}^3$  примет значение  $\mathcal{L}$  , а оставшееся выражение  $\mathcal{L}$  .  $\mathcal{L}$   $\mathcal{L}$  без просмотра будет отнесено к содержимому  $\mathcal{L}^2$  .

Теперь опишем алгоритм отождествления многоуровневых скобочных структур. Отождествление любой скобочной структуры ведется только на нулевом уровне. Всем скобкам первого уровня (если они есть) при этом сопоставляется пара скобок из конкретизируемото выражения. Если отождествление на данном уровне прошло успешно, отождествляются скобочные структуры следующего уровня. При этом вступает в силу принцип последовательного приоритета самой левой (или правой при обратном отождествлении) закрытой скобочной структуры. Приоритет этот не распространяется на закрытые скобочные структуры, находящиеся внутри открытых скобочных структурь.

Отождествление открытых скообчных структур начинается после того, как будут отождествлены все закрытые скообчные структуры, не находящиеся внутри открытых. При этом в зависимости от направления отождествления в первую очередь отоже дествляется либо левая открытая скообчная структура ( при прямом отождествлении), либо правая ( при этом отождествлении)

Приведем пример. Пусть имеется предложение, левая часть котороро имеет следующий вид:

И пусть конкретизируемое выражение есть

Порядс: отождествления данного конкретизируемого выражения с левой частью указанного предложения будет следующий.

Сначала отождествление ведется на нулевом уровне скобочной структуры. Если учесть, что все скобочные структуры первого уровня при отождествлении обходятся как в левой части

предложения, так и в конкретизируемом выражении, то нетрудно видеть, что результатом отождествления явится то, что переменная Е: примет значение Ро , а переменная Е: Гримет значение группу мы сопоставили скобкам левой части предложения скобки конкретизируемого выражения, и этот факт в примере отмечен одинаковыми индексами у соответствую — щих скобок.

Теперь из неотождествленных остались только одии откритие скобочные структури:  $(\dots) \bowtie (\dots)$  . Первой будет отождествляться скобочная структура  $(\dots)$  . В результате E примет значение A , а E - значение B . После отождествления данной скобочной структуры интерпретатор будет искать следующую ближайшую слева открытую скобочную структуру. Ей окажется скобочная структура  $(\dots)$  После ее отождествления E примет значение A . а E окажется пустым. При этом будет пропущена скобочная структура  $(\dots)$  . Она будет отождествляться в последнюю очередь, и переменная E примет значение A .

Описанный выше порядок отождествления скобочных структур обусловлен тем, что отождествление открытых скобочных структур связано по крайней мере с просмотром, а возможно с последовательным многократным удлинением какой-нибудь из открытых

переменных  $\subseteq$  . А каждое удлинение, как правило, вызывает дополнительные просмотры поля зрения. Закрытые же скобочные структуры отождествляются без просмотра.

Приоритет, отдаваемый при отождествлении закрытым скобочним структурам перед открытыми, открывает дополнительные возможности избежать многократного удлинения. Это происходит в тех случаях, когда одна и та же переменная типа  $\mathcal{E}$  встречается как в открытой, так и в закрытой скобочной структуре, и в соответствии со строением левой части закрытая структура отождествляется раньше открытой. В этом случае при отождествлечии открытая переменная  $\mathcal{E}$  набирается за один просмотр путем сравнения с последовательностью символов, относящихся к идентичной закрытой переменной  $\mathcal{E}$ .

Например, пусть предложение, левая часть которого имеет вид

является подходящим для конкретизируемого выражения

Интерпретатор отождествит закрытую скобочную структуру, содержащую  $E^2$  раньше, чем открытую скобочную структуру, содержащую ту же самую переменную. И при отождествлении открытой жобочной структуры интерпретатор уже не будет присваивать  $E^2$  сначала значение пусто, затем  $\ll$ , затем  $\ll \ll$  и  $\ll \ll$ . Правильное значение  $e^2$ , равное  $e^2 \ll$  будет определено сразу путем сравнения с содержимым переменной  $e^2$ , находящейся в закрытой скобочной структуре.

Отождествление предложения считается успешным после того, как отождествятся все скобочные структуры.

В случае неудачного стождествления какой-либо свобочной структури отождествление всего предложения считается неуспешним, если только данная структура не находится внутри какой-либо открытой скобочной структуры. В последнем случае управление передается на удлинение соответствующей открытой переменной  $\mathcal{L}$ 

При записи левой части предложения в память машини происходит анализ скобочных структур и разделение на типы переменных Е . Сначала производится запись символов на нулевом
уровне скобочной структуры. Все внутренние скобочные структуры при этом опускаются. В качестве их представителей на нулевом уровне остаются символы левых скобок. После таких скобок
записываются номера, указывающие порядок отождествления соответствующих скобочных структур.

Внутренние скобочные структуры записываются после внешней. При этом порядок записи таких структур определяется вышеизло-женными принципами. Например, левая часть предложения в примере будет записана в общую память интерпретатора в следующем виде:

В результате такой записи процессор в дальнейшем отождествияет скобочные структуры последовательно слева направо и при этом правила отождествления скобочных структур будут соблюдены. При наличии в комментарии знака направления отождествления (  $\underline{\mathcal{K}}$  ) левая часть предложения записывается в поле памяти в обратном порядке. Однако при этом определение детерминатива не меняется.

Все предложения общей памяти записываются в память машины плотно упакованными, по 5 символов в ячейке.

### 2.3. Замена левой части предложения на правую

Наличие адресов связи у символов поля зрения позволяет полностью ликвидировать переписи при замене левой части предложения на правую в том случае когда переменная в правой части встречается не в большем числе, чем в левой.

В процессе замены используется информация, хранящаяся в специальной таблице, так называемой таблице регистров. В эту таблицу в процессе синтаксического отождествления для каждой переменной заносится адрес начала и конда ее содержимо-го. Соответствующая ячейка таблицы р гистров выбирается с по-мощью идентификатора переменной. РЕФАЛ-интерпретатор обращается к таблице регистров, находит адрес начала и конца соответствующей переменной и при замене изменяет адреса связи начала и конца (А1 — начала и А3 — конца). Само тело переменной остается на прежних участках памяти. И весь процесс замены сводится фактически к перекоммутации адресов связи начала в конца используемых при этом переменных.

Напт мер. пусть мы имеем предложение

Покажем в таблице на какие символы указывают адреса начала и A3 конца переменных E1 и E2 до и после замены:

	до замени	после замены
AI начала EI	×	/3
A3 конца EI	β	)
AI начала E2	. β	Y
АЗ конца Е2	)	β

Время, необходимое для замени, теперь совершенно не зависит от числа символов, входящих в переменние. И только в том случае, когда в правой части какая-либо переменная встречается в большем числе, чем в левой части, переписи не избежать. Но данный случай не есть потеря эффективности, а является результатом существа самого алгоритма, связанного с размножением информации.

Например, в предложении

переменная <u>Е</u> в правой части встречается трижды, в то время как левой части она встречается только один раз. В этом случае <u>Е</u> когда она встречается второй и третий раз будет посимвольно переписана на свободном поле памяти.

### 2.4. Объединение предложений в цепочку просматриваемых предложений

РЕФАЛ-машина для нахождения подходящего предложения общей памяти просматривает не все предложения, а только те, детерми-

натив которых совпадает с данным. В настоящей главе мы уточним понятие детерминатива, введенное в L3J.

Определение 2:

Детерминативом называется последовательность символов, расположенных в левой части предложения после символа K, до ближайшей переменной типа S, W или E. В настоящем интерпретаторе число символов, входящих в детерминатив, ограничено тремя. В частности, детерминатив может быть и пустим.

Например:

имеет детерминатив, состоящий из трех символов  $\mathscr{S}$  с. Предложение  $\mathscr{S}$  K W1 & W2  $\sim$  W2 имеет пустой детерминатив.

Введение понятия детерминатива и аг элиз символов с которых начинается конкретизируемое выражение, дает возможность избежать просмотра тех предложений, которые заведомо являются
неприменимими. Для этого перед началом работы интерпретатора
предложения, вводимые в поле памяти, объединяются в цепочки
просматриваемых предложений (ЦПП). Смысл такого объединения
состоит в том, что, двигаясь по цепочке просматриваемых предложений в поисках подходящего, интерпретатор обходит те предложения, двтерминатив которых дает достаточные основания считать их неподходящими для данного конкретизируемого выражени.

Приведем пример, иллюстрирующий принцип объединения в цепочку просматриваемых предложений. Пусть имеется последовательность предложений (в примере не указываются правые части предложений):

и пусть конкретизируемым выражением является

Первим будет отождествляться предложение 3.1.

Отождествление будет неудачним. После этого бессмисленно, конечно, питаться отождествлять предложения 3.2 и 3.3, т.к. второй символ детерминатива у данных предложений не совпадает со вторым символом, стоящим в конкретизируемом выражении. Цепочка просматриваемых предложений организуется так, что за предложением 3.1 будет следовать сразу предложение 3.4.

Для того, чтобы обратиться к цепочке просматриваемых предложений необходим механизм определения начала цепочки. В примере (4) мы имеем три цепочки. Началом одной из них является предложение 3.1, другой 3.2 и третьей 3.3. Конец у всех этих цепочек общий — предложение 3.4. Ниже мы опишем алгоритм определения начала цепочки просматриваемых предложений, а пожа введем несколько определений, необходимых для формального описания такого алгоритма. Заметим только, что предложение вводится в память машины в порядке, обратном направлению прос-

метра предложений, указанном при описании языка РЕФАЛ [ ]
В дальнейшем мы будем писать предложения, так как они вводятся в память машины. При этом направление просмотра меняется
на противоположное.

Определение 3.

Пусть им имеем последовательность предложений:  $A_1, A_2, \dots, A_n$ . Назовем предложение  $A_i$  более общим, чем  $A_i$ , а предложение  $A_i$  менее общим, чем  $A_i$ , если выполняется следующее равенство:

где Det(A) обозначает детерминатив предложения A , а C — непустая цепочка символов.

В том случае, когда C пуста, будем называть предложения  $A_i$  и  $A_j$  равнообщими.

Например, пусть мы имеем последователя чость предложений (в примере в левых частях указываются в лько детерминативы)

§ 1 K a ~	
§ 2 K a 6 ~	
§ 3 K aa ~	
§ 4 K a B c ~	
\$5 Ka ~	(np.5)
\$ 6 K aaa ~	
\$7 Kabe ~	
§ 8 K aa ~	
§ 9 K 99	
§ 10 K abc ~	
§ II K a 6 d ~	

Предложение I является более общим чем предложения 2,3,4,6+II, но не предложение 5. Предложение 2 является более общим по отношению к 4,7,I0,II, но не по отношению к 1,3,5,6,8,9.

Все предложения являются менее общими, чем предложение I, за исключением предложения 5. Оно по отношению к предложению I является равнообщим предложением. Также равнообщими предложения жениями являются предложения 3,8 и 9, т.к. они имеют один и тот же детерминатив (  $\mathcal{O}\mathcal{Q}$  ).

Определение 4.

Назовем последовательность вводимых в память предложений монотонной по общности (или просто монотонной), если никакое следующее предложение не является более общим, чем преднаущее.

В примере ( 5 ) мы имеем три монотонных последовательности: первая включает предложения I,2,3, 4; вторая - 5,6,7;
третья - 8,9,10,II. В самом деле ни одно из следующих предложений в последовательности предложений I,2,3,4 не является
более общим, чем предндущее. Предложение же 5 является более
общим, чем предндущее предложение 4. Так что с появлением данного предложения монотонность последовательности нарушается.
Предложение 5 становится началом следующей монотонной последовательности. Нарушает ее монотонность предложение 8, т.к.
оно является более общим, чем предложение 6.

Каждой монотонной последовательности в памяти машины сопоставляется иерархия, состоящая из узлов. Узел содержит символ детерминатива и адреса, с помощью которых организуется иерархия. При записи предложений в поле общей памяти интерпретатора узлы записываются перед теми предложениями, которые они

представляют. Тем инструментом, с помощью которого находится начало цепочки просматриваемых, является иерархия.

Максимальное число уровней в иерархии равно трем.

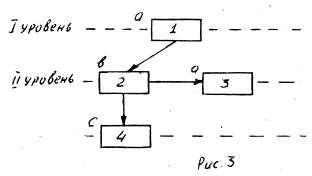
На первом уровне располагается узел предложения, детерминатив которого состоит из одного символа. На втором уровне находятся узян предложений, детерминатив которых состоит из двух символов, и на третьем уровне — из трех символов. При этом следует заметить, что каждый узел хранит только тот символ детерминатива, место которого в детерминативе совпадает с номером уровня иерархии, на котором расположен данный узел.

Например, детерминатив предложения 4 в примере (5) является как бы рассредоточенным: первый его символ (а) будет расположен в узле предложения I (на первом уровне мераржим), второй символ (б) — в узле предложения 2 (на втором уровне мерархии) и только третий символ будет располагаться в узле, соответствующем предложению 4 (на с этьем уровне мерархии).

Узлы одного и того же уровня иерархии связываются между собой в однонаправленную цепочку, и каждый из них может иметь адрес — ссылку на соответствующий узел следующего (нижнего) уровня. Цепочка первого уровня состоит из одного узла. На третьем уровне может быть несколько цепочек, каждая из которых порождается узлом предыдущего уровня.

Узям образуются и связываются между собой по мере поступления предложений в память машины.

Обратимся вновь к примеру ( 5 ) и покажем, как заполнятся уровни иерархии, соответствующей монотонной последовательности, состоящей из предложений 1,2,3,4. Покажем такую иерархию на рисунке.



Стрелки на рисунке заменяют адреса, расположенные в узлах. В прямоугольниках указаны номера предложений, к которым относятся данные узлы.

Узел, расположенный на I урсвне иерархии, называется вершиной иерархии. Его адрес указывается в специальной таблице,
так называемой таблице детерминативов (¬Д) .Число ячеек
в такой таблице равно числу возможных значащих символов в языке РЕФАЛ плюс еще два (для левой и правой скобки), так как
все они могут быть первым символом детерминатива. Обращение
к такой таблице производится по первому символу детерминатива.

Теперь в случае мерархии, показанной на рисунке, нетрудно представить процесс поиска первого подходящего предложения общей памяти, например, для конкретизируемого выражения следующего вида:

K (aBc .. )

Мы сознательно не уточняем всего выражения. Нас интересуют только три первых символа — abc. По первому символу обращаемся к  $\mathcal{TD}$  и определяем узел, соответствующий вершине иерархии. А затем , сравнивая по порядку следующие символы

конкретизируемого выражения с символами, храняющимися в узлах на соответствующих уровнях, и используя адреса связи, мы приходим к узлу предложения 4.

Теперь дадим формальное определение цепочки просматриваемых предложений (ЦПП).

Определение 5.

ЩП называется такое объединение предложений, при котором каждое следующее является либо менее общим, либо равнообщим по отношению к предыдущему.

Такая организация ЦПП позволяет РЕФАЛ-интерпретатору при отнакании подходящего предложения избегать просмотра ненужных предложений. В самом деле, обратимся вновь к примеру ( 4 ). Мы убеждаемся, что пояснения к данному примеру, иллюстрирующие ранее данное неформальное определение ЦПП, полностью согласуется с только что данным формальным определением. Каждое из предложений 3.2, 3.3, и 3.4 является менее общим, чем предложение 3.1. И поэтому ЦПП, имеющая началом, ст жем, предложение 3.4, следующим элементом будет иметь предложение 3.1, так как только оне является более общим, чем предложение 3.4.

Нетрудно видеть, что в ППП происходит фактически обратное движение, с нижних уровней мерархии к верхним, но только это движение направлено строго вверх. При этом пропускаются все предложения, детерминативы которых расположены на одном уровне.

Напрямер, если началом ЦПП внутри мерархии, изображенной на рисунке стр. 25, является предложение 3, то ЦПП от этого предложения будет указывать сразу на предложение 1, минуя предложение 2.

В целях экономии памяти для равнообщих предложений заводится только один узел — для самого последнего из них. Все остальные связиваются в ШП.

Связь предложений в ШП осуществляется с помощью ячеек ШП, в которых хранятся адреса связи. Такие ячейки так же как и узлы находятся перед предложениями при записи их в поле общей памяти.

Ранее уже упоминалось, что монотонность последовательности может нарушаться. При этом образуется новая иерархия. В таблице детерминативов указывается адрес вершини последней из них. В ячейке ППП, соответствующей такой вершине, указивается адрес вершини той иерархии, нарушением монотонности которой явилось данное предложение.

Таким образом, в общем случае, все предложения, детерминачтив которых начинается с одного и того же символа, могут быть объединены в целур серию иерархий. Поиск начала ШШ производится в самой последней из них. Это полностью согласуется с описанием РЕФАЛа. Абстрактная РЕФАЛ-машина просматривает предложения с данным детерминативом, начиная с первого из них. В РЕФАЛ-интерпретаторе просмотр начинается с самого последнего предложения, имеющего данный детерминатив, внутри самой последней иерархии (если учесть обратный порядок ввода предложений в память РЕФАЛ-интерпретатора — это одно и то же).

После того, как найдено начало ППП, происходит поиск подходящего предложения внутри соответствующей нерархии. При какдом неудачном отождествлении движение происходит по ППП. Если внутри данной исрархии подходящее предложение не было найдено, движение по ППП временно приостанавливается и происходит обращение к предыдущей иерархии. В ней так же, как и в последней иерархии, согласно конкретизируемому выражению ищется начало ЩП внутри данной иерархии, которое фактически является предолжением ЩП, относящейся к предыдущей иерархии. После этого движение вновь осуществляется по ЩП и т.д.

Если в результате поиска подходящего предложения мы дошли до вершины самой первой иерархии, дальнейший поиск осуществляется в цепочке просматриваемых предложений с пустым детерминативом (предложения типа G). Предложения типа G объединяются в ПЛП также с помощью адресов, расположенных в ячейках ПЛП. Начало такой цепочки хранится в специальной ячейке и указывает на самое последнее предложение типа G.

Вынос цепочки просматриваемых предложений типа G выше любой цепочки предложений с непустым детерминативом диктуетом требованием абстрактной РЕФАЛ-машины (опять же с учетом обратного ввода предложений).

В следующей главе мы подробно опишем алгоритм построения мерархии и ШШ, пока же укажем какая информация должна храниться в узлах и в ячейках ШШ.

- адрес начала левой части самого последнего предложения
   из числа равнообщих данному внутри данной иерархии;
- адрес узла менее общего предложения, чем предложение с данным детерминативом (адрес начала второго уровня иерархии).

Узел второго уровня иерархии должен содержать:

- I) код, соответствующий второму символу детерминатива:
- 2) адрес начала левой части самого последнего предложения из числа равнообщих данному внутри данной иерархии:

- адрес узла менее общего предложения, чем предложение с данным детерминативом ( адрес начала соответствующей цепочки узлов третьего уровня иерархии);
- 4) адрес узла предложения, детерминатив которого отличается от детерминатива данного предложения только последним символом (адрес следующего узла второго уровня иерархии).

Узел третьего уровня должен содержать :

- І) код, соответствующий третьему символу детерминатива;
- адрес начала левой части самого последнего предложения из числа равнообщих данному внутри данной мерархии;
- 3) адрес узла предложения, детерминатив которого отличается от детерминатива данного предложения только последним символом (адрес следующего узла данного уровня).

Ячейка ШШ должна содержать признаковую часть 🗡 и адресное поле.

Если  $\pi = 0$ , то адресное поле содержит адрес, указываю— щий начало левой части того предложения, которое необходимо этождествлять после неудачного отождествления данного предложения внутри данной иерархии. С помощью этих адресов происходит движения по цепочке просматриваемых предложений.

Если  $\mathcal{J}=1$  , то адресное поле содержит адрес вершины той мерархии, нарушением монотонности которой явилось данное предложение.

Объединение предложений в иерархии и в ШШ позволяет существенным образом сократить время нахождения подходящего предложения для данного конкретизируемого виражения. Если бы этого не было, отбор предложений можно было бы производить только с помощью синтаксического отождествления, что привело бы к лишним просмотрам.

### 2.5. Программа "чистильщик"

Как уже указывалось, в РЕФАЛ-интерпретаторе, кроме поля зрения и поля общей памяти, заводится свободное поле памяти (СПП), которое объединяет ячейки памяти, не занятые под поле зрения и поле общей памяти.

Основное назначение программы "чистильщик" — эффективное использование памяти машины при работе интерпретатора. Будем считать, что память в интерпретаторе используется эффективно, если все участки памяти, оказавшиеся ненужными в результате какого-либо шага машины, могут быть использованы в дальнейшем в качестве массива свободной памяти.

Работает программа "чистильщик" следующим образом. После окончания синтаксического отождествления вые конкретизируемое выражение подпивается к СПП. И если ни одна из переменных, иеходящихся в левой части, не используется при замене левой части предложения на правую, весь этот массив остается в СПП.

А переменная не используется при замене в том случае, если она отсутствует в правой части предложения.

Если какая—то переменная используется при замене, то из подшитого к СШ массива содержимое такой переменной викидивает—ся. Это достигается тем, что символ, преднествующий содержимо—му такой переменной, и символ следующий за ним, связиваются между собой с помонью адресов связи. Такая процедура применяет—ся к каждой переменной, используемой при замене левой части

предложения на правую. В результате к СШІ подшивается массив ячеек, оказавшийся ненужным при выполнении очередного шага машины.

Например, пусть имеется предложение

и конкретизируемое выражение

Так как при замене используется только переменная  $\[Mathbb{W}\]^2$  ее содержимое выкидывается из содержимого подшитого к СПП конкретизируемого выражения. Для этого символы + и  $\[Beta]\]^3$  свизываются между собой. И к СПП будет подшито выражение

Наличие программи "чистильщик" позволяет полностью использовать память машини.

### 3. РЕФАЛ-интерпретатор для машины БЭСМ-6

В настоящей главе описывается общая схема и основные олоки  $\text{РЕФАЛ-интерпретатора} \left[ \begin{array}{c} 5 \end{array} \right]$  .

Работа блоков иллюстрируется с помощью блок-схем. Для описания блок-схем используется специальный полуформализованний язык блок-схем, который подробно описывается в настоящей главе.

При описании каждого блока мы специально будем подчерки-

вать те ограничения, которые он накладывает на входной язык. В конце главы эти ограничения будут суммированы в специальном разделе, посвященном описанию входного языка РЕФАЛ-интерпретатора.

Описываемый интерпретатор был написан на автокоде для машины БЭСМ-6 [См. 6] . Описание машины БЭСМ-6 см. в [ $\frac{7}{2}$ ].

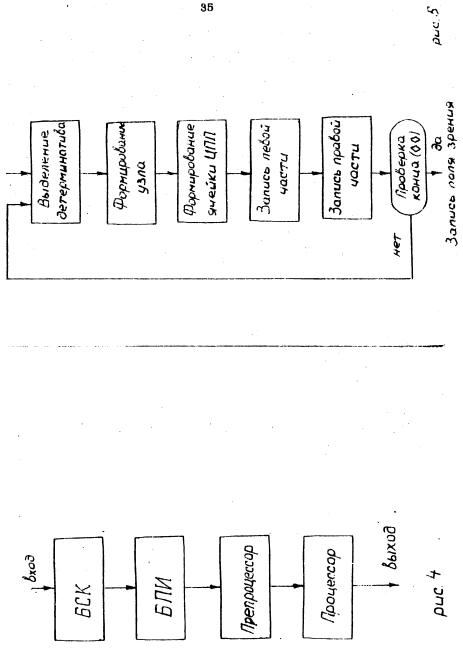
На рис. 4 изображена общая схема интерпретатора. Интерпретатор состоит из четнрех основных блоков: блок перекодировки вводимой информации во внутренний код интерпретатора (БПИ),
блок синтаксического контроля (БСК), препроцессор и процессор.

Собственно интерпретатором является фактически только процессор, который и представляет из себя модель РЕФАЛ-машины. Однако для повышения эффективности работы процессора совершенно необходим препроцессор, который в соответствии с принципами, описанными в главе II, записывает информацию в поле зрения и в поле общей памяти в удобном для процессора виде. В настоящей главе мы подробно опишем работу этих основных частей интерпретатора.

БПИ и БСК никакой принципиальной нагрузки не несут. Поэтому подробно описывать работу этих слоков мы не будем. Укажем только вкратце их назначение.

БПИ перекодирует 8-ми разрядные коды вводимых символов в 9-ти разрядные. Это делается для того, чтобы увеличить общее число возможных символов в РЕФАЛе. Таким образом максимальное число символов в РЕФАЛ-интерпретаторе равно 512.

Цепочка символов, заключенная в угловые скобки, также заменяется 9-ти разрыщным кодом. Содержимое самой цепочки



запоминается в таблице 71 и в дальнейшем используется при печати промежуточных или окончательных результатов. Таблица ограничивается 2400<sub>8</sub> ячейками, а это приводит к тому, что максимальное число символов, расположенных в кавычках, не должно превышать I2. Если после записи всех символов таблица 71 использовалась не полностью, оставшаяся часть интерпретатором относится к СШІ.

Особо выделяются при перекодировке управляющие символы языка. Они приобретают следующие внутренние коды:

Управляющий символ	Код
K	700
Ē	600
$oldsymbol{ar{w}}$	500
<u>\$</u>	400
<b>)</b>	300
(	200
-	100

Такая кодировка управляющих символов позволяет процессору бистро отличать их от значащих символов. Признаком, по которому они отличаются от значащих символов, являются нули в млад ших шести разрядах.

При перекодировке все буквенные идентификаторы переменных заменяются цифрами.

БСК производит синтаксический контроль вводимого текста в соответствии с формальным описанием синтаксиса РЕФАЛа.

При проверже предложений общей памяти анализируется каждое предложение отдельно. При обнаружении ошибки ECK печатает тип

ошибки и то предложение, в котором она обнаружена. Часть предложения, находящаяся справа от места ошибки, при этом не анализируется. Так что в каждом предложении ЕСК может обнаружить максимум одну ошибку. Остальные ошибки можно обнаружить, анализируя визуально напечатанный текст предложения, либо при повторном контроле с помощью ЕСК.

К числу наиболее типичных ощибок, обнаруживаемых БСК при анализе предложений общей памяти, относятся:

- I) отсутствие § в начале предложения;
- 2) отсутствие символа К в начале левой части;
- отсутствие знака 
   разделяющего левую и правую часть предложения;
- 4) подчеркивание символа, отличного от  $S_{i}W_{i}Eu_{i}$
- при отсутствии подчеркивания у символов S, W или E
  печатается предупреждение, но анализ предложения
  не прекращается;
- 6) отсутствие левой или правой угловой скобки в том случае, когда имеется сложный символ, заключенный в кавычки;
- 7) Наличие внутри угловых скобок цепочки символов, состоящей более чем из 12 символов;
- 8) наличие в предложении переменных разного типа, имеющих один и тот же идентификатор;
- наличие в правой части предложения переменной, отсутствующей в левой части;
- 10) наличие несбалансированной скобки как в левой, так и в правой части предложения;

II) отсутствие  $\cdot$  , относящейся к соответствующему символу K , или наоборот, наличие лишней  $\cdot$  , не сбалансированной соответствующим символом K .

При синтаксическом контроле поля зрения БСК обнаруживает несбалансированные скобки, а также проверяется соответствие между символами  $\kappa$  и \_\_\_\_.

## 3.1. Изик блок-схем

В данном разделе мы опишем полуформализованный язык, который используется для записи алгоритмов в виде блок-схем. В дальнейшем все блоки интерпретатора будут описаны на этом языке. Все вновь вводимые формальные обозначения при описании нового блока каждый раз будут подробно описываться.

Всякая блок-схема представляет из себя совокупность отдельных блоков, соединенных стрелками. В описываемом нами языке стрелки имеют свое обычное значение. Что же касается тех действий, которые совершаются в отдельных блоках, для их описания используются следующие основные понятия (в данном случае в качестве ограничительй металингвистических переменных в нормальной бэкусовской форме используются угловые скобки).

## І.І. Адрес

I.I.I. CHHTAKCHC

<буква> ::= любая буква латинского, греческого или русского аліравита

<umppa> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

< идентификатор> ::= <буква> \ < идентификатор> < буква? \ < идентификатор> < цифра >

І.І.З. Семантика

Адрес предназначен для указания расположения ячейки или начала массива ячеек в памяти машины.

```
І.2. Переменная
```

I.2.I. Cuntakene

I.2.2. Примеры

Простая переменная 
$$H4$$
 $M1_{A1+2}$ 
 $S_{X}$ 
 $I(I)$ 

Косвенная переменная  $A1[B3_{X}]$ 
 $H2[A2[Y1_{X-1}]-3]$ 

Содержимое таблицы  $M1[T5(Spi)]$ 
 $B[T1(X[T2(A5_{Q})])]$ 

1.2.3. Семантика.

Простея переменная позволяет обращаться не только к целой ячейке, но и к ее части. В этом случае адрес ячейки указывается в индексном выражении, а выделяемая часть ячейки указывается с помощью идентификатора. Например, если в ячейке по адресу с выделяемая часть АІ расположена в восьми младших разрядах, для обращения к этой части достаточно написать АІ. . Для полной формализации записи алгоритма в языке необходимо иметь средства, позволяющие формально задавать идентификато цию различных частей ячейки. Однако, такого средства мы не даем, а в дальнейшем, в необходимых случаях, будем содержательно описывать такую идентификацию. Символ , при описании простой переменной имеет служебное значение. Он применяется в тех случаях, когдя несущественно с точки зрения описываемого алгоритма к какой части ячейки в данный момент идет обращени

Задание простой переменной в виде самоопределенной вели - чины позволяет во многих случаях упростить запись алгоритма.

Самоопределенная величина - это, как правило, константа.

С введением косвенной переменной в языке появляется возможность косвенной адресации. Например, если в части 3 ячейки, имеющей адрес  $\propto$  , хранится адрес  $\beta$ , то справедливо следующее равенство:

С помощью переменной «содержимое таблицы» можно обращаться к соответствующей ячейке таблицы, начало которой обозначается с помощью служебного символа 7 и следующего за ним целого десятичного числа. Значение выражения, стоящего в круглых скобках, есть то число, которое прибавляется к началу таблицы для указания адреса искомой ячейки таблицы.

І.З. Операторы

I.3.I. CUNTAKENE

< оператор присваивания > := < переменная> : = < переменная>

< оператор сравнения > ::= < переменная > < операция > < операция сравнения > ::= < | < | > | > | >

I.3.2. Примеры

$$A1_{d} := B2_{\beta}$$
 $M1[T2(\beta_{d})] := H2[A1_{\beta}+1]$ 
 $H4_{d} := '('$ 
 $A1[C1_{CDK}] := 'K'$ 

І.З.З. Семантика.

Оператор присваивания имеет свой обычный смысл. Операторы сравнения аналогичны командам машины, осуществляющим условную передачу управления, и предназначены для разветвления програм-

Если в дальнейшем при описании отдельных олоков нам потребуется новые операторы, мы будем указывать их синтаксис и семантику.

## 3.2. Препроцессор

Препроцессор записывает предложения общей памяти и поле зрения в память машинь.

На рис. 5 изображена общая схема препроцессора.

Запись каждого предложения начинается с выделения детерминатива. Затем в соответствии с принципами п.4 главы II формируется узел и ячейка ЦПП для данного предложения. После этого записывается левая и правая части предложения.

После записи каждого предложения проверяется, не является ли данное предложение последним в массиве предложений общей намяти. Если после данного предложения стоят подряд два параграфа ( § § ), запись предложений общей памяти заканчивается, и управление передается на запись поля эрения.

Рассмотрим более подробно работу каждого из этих блоков.

## 3.2.1. Виделение детерминатива (ВЦ)

Блок ВД выделяет детерминатив предложения в соответстви с его формальным определением (см.п.4 гл.П). Практически это осуществляется просмотром первых трех символов, стоящих в левой части предложения сразу за символом К. Появление среди них какой-нибудь свободной переменной отраничивает детерминатив числом символов, расположенных слева от такой переменной. В противном случае все три символа составляют детерминатив данного предложения.

На рис. 6 изображена блок-схема работы блока выделения детерминатива.

После того, как БПИ перекодирует вводимую информацию во внутренний код интерпретатора, она переписивается на маг-нитный барабан (МБ). Препроцессор переписывает ее с МБ квантами по 256 слов, для чего отводится соответствующий буферный массив ячеек МОЗУ. Обозначим его через 

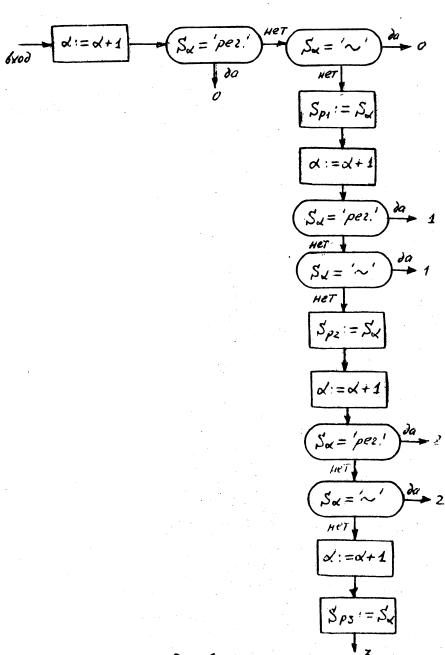
.

Первим символом каждого предложения является §. Он в память машини не записывается так же как и весь следующий за ним комментарий, символ  $\mathcal{K}$  ( если он есть) и символ  $\mathcal{K}$  . Если встречается символ  $\mathcal{K}$  , это значит, что левая часть данного предложения должна бить записана в обратном порядке. В дальнейшем мы особо выделим этот случай, а пока будем считать, что указатель направления синтаксического отождествления отсутствует.

Если после символа  $\kappa$  идет  $\kappa$ , это значит, что детерминатив пустой. Если не  $\kappa$ , то проверяется не является ли следующий символ символом переменной. Если да, это также означает пустой детерминатив. Если нет, то данный символ запоминается в рабочей ячейке РІ, берется следующий символ и производится аналогичная проверка.

Из данного блока существует четыре выхода: пустой детерминатив, детерминатив, состоящий из одного символа (запоминается в РІ), детерминатив, состоящий из двух символов (запоминается в РІ и Р2) и детерминатив, состоящий из трех символов (запоминается в РІ, Р2 и Р3).

Из блок-схемы видно, что скобки в соответствии с определением детерминатива включаются в детерминатив. Если , например,



Puc. 6

левая часть предложения имеет вид

то данное предложение будет иметь детерминатив, состоящий из трех символов.

Каждый из четырех случаев в препроцессоре является выделенным. Для каждого существует своя программа формирования узла. Мы рассмотрим только один случай — наличие детерминатива, состоящего из двух символов, Формирование узла и ячейки ЦПП очень трудно разделить, поэтому объединим их в одном блоке.

# 3.2.2. <u>Включение в цепочку просматриваемых предложений</u> (<u>ШШ</u>)

Исходной информацией для данного блока является: начало таблицы детерминативов (обозначим его через Т2) и детерминатив данного предложения, первый символ которого находится в ра-бочей ячейке PI, а второй — в P2.

По первому символу детерминатива происходит обращение к соответствующей ячейке таблицы Т2. В такой ячейке хранится адрес начала той иерархии, к которой относится данное предложение. В дальнейшем движение происходит по узлам иерархии.

В результате работы блока формируется узел для данного предложения, который связнвается с узлами второго уровня ме-. рархии, и ячейка ЦПП, в которой указываетая адрес ближайшего более общего предложения, относящегося к данной мерархии. Узлу второго уровня мерархии отводятся две ячейки памяти машины БЭСМ-6.

Ý	3	39	24	15	· ·
	W1	<i>u</i> 2		43	
				44	

Puc. 7

#### где:

ИІ - код, соответствующий второму символу детерминатива;

И2 - адрес начала левой части самого последнего предложения из числа равнообщих данному внутри одной иерархии:

ИЗ - адрес увла менее общего предложения;

И4 - адрес следующего узла второго уровня иерархии.

Ячейка ЦПІ записывается в память сразу после узла. Для нее отводится одна ячейка памяти машины БЭСМ-6.

4	8	34	24	16 15	0
	B1	82		t 8	5

Puc. 8

#### rge:

ВЗ содержит адрес связи предложений в ЦПІ (см.п.4 гл.11); В2 содержит адрес начала правой части предложения, а поскольку предложения общей памяти хранятся в упакованном виде, то в В1находится константа сдвига, указывающая число разрядов, на которое нужно одвинуть первый символ правой части для того, чтобн он оказался в стандартном положении. Если при движении по узлам второго уровня находится узел предложения равнообщего данному, часть из которого отлична от нуля, это значит, что данное предложение нарушает монотонность последовательности: В этом случае образуется вершина новой иерархии.

На рис. 9 изображена блок-схема алгоритма включения очередного предложения в ШШ.

Часть ячейки таблицы детерминативов (Т2), содержащая адрес вершины иерархии на схеме осозначается через МІ. Адрес начала свободной памяти, в которой записывается очередное предложение, обозначен через  $\sim$  .

Содержимое МІ ячейки Т2, соответствующей первому символу детерминатива, запоминается в рабочей ячейке РЯІ.

МІ будет пустым в том случае, когда на одного солее общего предложения не было записано в память машины. В этом случае узел данного предложения становится вершиной новой мерархим. Для этого в Т2 помещается адрес первой ячейки данного узла  $M1\left[T2\left(S\rho_1\right)\right] = \mathcal{N}$ . В дальнейшем заполняются ИІ, И2, ИЗ и И4 первой и второй ячейки узла. В ячейке ШШ  $\mathcal{I}$  и ИЗ остаются равными О. Для процессора это означает, что при движения по ШШ после неудачного отождествления следует обратиться к ШШ с пустым детерминативом.

Если  $M1 \neq 0$ , это значит, что более общее предложение уже было записано. В этом случае мы обращаемся к ячейке по адресу МІ. Если ИЗ данной ячейки равно 0, это значит, что на втором уровне мерархим до этого не было записано ни одного предложения. В таком случае узел, находящийся в вершине мерархим,

$$S_{PR1} := M1[T2(S_{P1})]$$

$$S_{PR1} = \emptyset$$

$$00$$

$$T.2$$

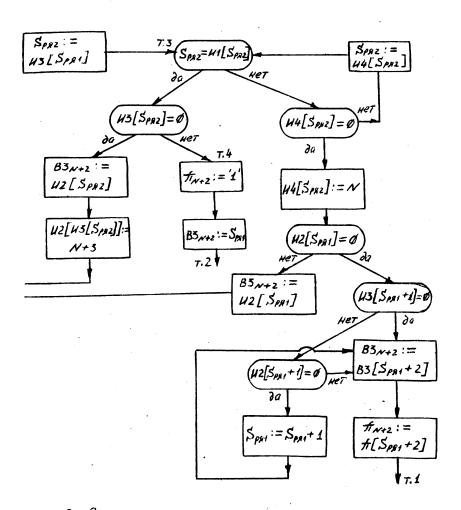
$$M1[T2(S_{P1})] := M$$

$$H3_{N} := N+1$$

$$W1_{N} := S_{P2}$$

$$W2_{N} := N+3$$

$$N := N+3$$



Puc. 9

должен указывать на узел данного предложения  $(3 [S_{PS}]) = N$ В ячейку ШП посылается адрес начала предложения, стоящего в вершине иерархии  $(33_{N+2}) = (2[S_{PS}])$ . Остальные ячейки вновь образуемого узла формируются как и в предыдущем случае, для чего управление передается в (7, 1).

Если ИЗ[ Зат] не равно 0, в этом случае ИЗ указывает на первый узел второго уровня иерархии. Все остальные узлы данного уровня связаны в однонаправленную цепочку с помощью адресов, расположенных в И4 (см.рис. 7). Среди таких узлов ищется узел предложения, детерминатив которого равен данному. Для этого содержимое ячейки Р2 сравнивается с частью ИІ каждого узла.

При удачном поиске проверяется, не содержит ли найденный узел адрес—ссылку на следующий уровень иерархии. Если нет, т.е.  $\mathcal{W}3\left[S_{\rho_{HZ}}\right]\neq0$ , это значит что данное предложение должно быть связано с равнообщими ему предложениями, записанными ранее в память машины. Для этого в узле, соответствующем всем равнообщим предложениям, в И2 должен стоять адрес начана левой части последнего из них  $\mathcal{W}2\left[\mathcal{W}3\left[S_{\rho_{BZ}}\right]\right]:=\mathcal{N}+3$  Ячейка ШП последнего предложения должна содержать адрес начала левой части предыдущего равнообщего предложения  $\mathcal{B}3_{\mathcal{N}+2}:=\mathcal{W}2\left[S_{\rho_{BZ}}\right]$ . Например, если мы имеем последовать нарагомения равнообщих предложений

то по мере записи этих предложений в И2 узла, относящегося

к предложению 2.2. І будет находиться адрес начала левой части предложений: сначала 2.2. І, затем 2.2. 2, 2.2. З и наконец 2.2.4. Так что, хотя формально узел стоит перед предложением 2.2. І, фактически он относится к предложению 2.2.4, так как указывает на начало левой части именно этого предложения.

Если узел, относящийся к предложениям, равнообщим данному, содержит адрес-ссылку на следующий уровень иерархии, это означает, что записываемое вновь предложение нарушает монотонность последовательности. В этом случае согласно п.4 главы П образуется вершина новой иерархии. Для этого в ячейке ЦПП данного предложения (см. 74 на блок-схеме) принимает значение I, а в ВЗ посылается адрес вершины той иерархии, нарушением монотонности которой явилось данное предложение  $8.5_{\text{MAZ}} := S_{\text{PMI}}$  В дальнейшем управление передается в 7.2 для формирования узла данного предложения.

Если при движении по цепочке узлов второго уровня не будет

найден узел предложения, равнообщего данному, рано или поздно ми дойдем до узла, у которого  $\mathcal{M}^4=\mathcal{O}$ . В этом случае образуется новый узел данного уровня мерархии. В ИА последнего найденного узла посылается адрес вновь образованного узла  $\mathcal{M}^4\left[S_{F\mu 2}\right]=\mathcal{N}$ . В дальнейшем проверяется, не стоит ли в вершине последней мерархии предложение, детерминатив которого состоит из двух символов. Если это так, т.е.  $\mathcal{M}^2\left[S_{FR1}\right]=\mathcal{O}$  и  $\mathcal{M}^3\left[S_{FR1}+1\right]=\mathcal{O}$ , то в ячейку ЦПП записывамого предложения, посылается содержимие  $\mathcal{M}$  и ВЗ вершины мерархии ( см. 76). Например, пусть мы имеем последовательность вводимых в память машины предложений

§ 2.2.5 <u>K</u> a ... ~ § 2.2.6 <u>K</u> a & c ... ~ § 2.2.7 <u>K</u> a & ... ~ § 2.2.8 <u>K</u> a d ... ~

и пусть в данний момент записывается предложение 2.2.8. В его ячейку ЦПП перенесется содержимое ли ВЗ ячейки ЦПП предложения 2.2.7, которое является вершиной последней иерархии. Т.е. предложение 2.2.8 тоже становится как бы вершиной иерархии. И, действительно, нет никакого смысла в случае неудачного отождествления предложения 2.2.8 передавать управление на отождествление предложения 2.2.7, так как заранее ясно, что оно не отождествится. И поэтому управление передается на поиск соответствующего предложения в предыдущей иерархии. В этом состоит специфика построения иерархии, в вершине которой стоит предложение с детерминативом, состоящим из двух символов.

Если  $WS[S_{phi}+1]$  не равно 0, то тем самым выделяется случай, когда предложение с детерминативом из двух символов записывается сразу после предложения, детерминатив которого содержит 3 символа, причем вторые символы у них не совпадают. В этом случае в ячейку ЩШ заносится значение  $\mathcal{T}$  и ВЗ вермину иерархии. Например, пусть мы имеем последовательность предложений

§ 2.2.9 Kob...~ § 2.1.10 Kab...~ § 1.2.11 Kobc...~ § 2.2.12 Kad...~

и пусть предложение 2.2.9 находится в вершине последней иерар-

хии. В ячейку ЦПП предложения 2.2.12 будет записана ссылка на вершину предндущей мерархии, так как ни одно из предложений 2.2.9 + 2.2.11 не может оказаться подходящим, если в качестве первого из проверяемых было выбрано предложение 2.2.12

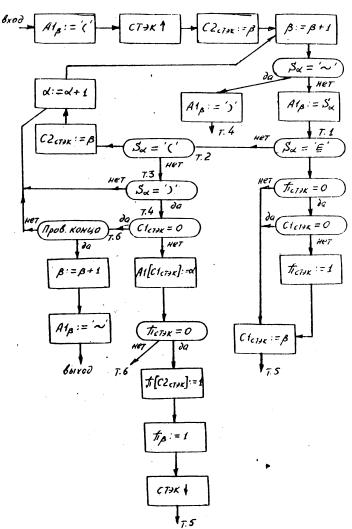
И наконец, когда  $W2[S_{PS}]$  отлично от нуля, в этом случае в ячейку ШП записываемого предложения посылается адрес, указывающий начало левой части более общего предложения  $BS_{N+2} = W2[S_{PS}]$ . В дальнейшем управление рередаетоя в  $\tau$  1 (см.блок-схему) на формирование узла данного предложения.

Аналогично производится включение в ЦПП предложений, детерминатив которых состоит из одного или из трех символов. Следует только заметить, что предложение, детерминатив которого
состоит из трех символов, никогда не может нарушить монотонность последовательности предложений. Это следует из определения детерминатива.

## 3.2.3. Запись девой части предложения

Запись девой части предложения происходит в два этапа. Сначала левая часть предложения переписывается в развернутом виде (по символу в ячейке) на отдельном массива ТЗ, а затем включается блок записи левой части в массив предложений общей намяти.

На рис. 11 изображена блок-схема алгоритма переписи левой части предложения в развернутом виде. Под массив ТЗ отводится 100 ячеек. Это приводит к тому, что в левой части предложения числе симвелов не должно превосходить 100. Практически это не накладывает на язик существенных ограничений, т.к. практика



Puc. 11

использования РЕФАДа [ 8, 9,  $\hbar$ ,  $\eta$ ], показала, что в левой части редко когда бывает больше 30 символов.

В процессе записи левой части в развернутом виде одновременно происходит анализ скобочных структур и разделение на типы переменных <u>Е</u> в соответствии с определениями, данными в п.2 гл..П.

При развертивании левой части каждый символ помещается в отдельной ячейке массива ТЗ (см. рис. 10):

48 47 39	24	15	O
#	A1		

PMC- 10

АІ содержит код самого символа (разряды с 16 по 24),  $\pi$  — признак для скооки ( $\pi$ =0 — закрытая скооочная структура,  $\pi$ =1 — открытая). Для открытой переменной E в АІ записывается код 600, для закрытой 700.

При анализе скосочных структур заводится стак. На рис. 12 изображена ячейка такого стака:

48 47	39	24	15	0
#	CI			C2

Puc. 12

- СІ адрес закрытой переменной  $\mathcal{E}$  (если она есть в скобочной структуре),
- С2 адрес соответствующей левой скооки,
- признак, который равен 0 для закрытой скобочной структуры и I в случае открытой скобочной структуры.

Обратимся вновь к рис. // Текущий адрес массива ТЗ обозначается через /3 , переписываемого массива через  $\alpha$  . Образование новой ячейки стэка обозначается СТЭК  $\alpha$  , ликвидация последней строки — СТЭК  $\alpha$  .

Перед входом в блок массив ТЗ и стэк очищаются,  $\beta$  принимает значение адреса начала массива ТЗ,  $\propto$  - адреса начала левой части предложения.

Работа блока заключается в посимвольной переписи содержимого массива  $\alpha'$  в массив ТЗ. При этом выделяются случаи появления  $\widehat{F}$ , ( или ) (см. точки I,2 и З на схеме).

При появлении левой скобки в стэк заносится адрес этой скобки в массиве ТЗ. Если скобочная структура, относящаяся к данной скобке, при дальнейшем анализе окажется открытой, то адрес, содержащийся в стэке используется для изменения ж данной левой скобки:

При появлении <u>Е</u> анализируется содержимое последней строки стака.

При первом появлении  $\underline{E}$  на нулевом уровне скобочной структури значение  $\pi$  остается равным I, что говорит о том, что данная скобочная структура является открытой. В СІ последней ячейки стэка хранится адрес последней переменной  $\underline{E}$  и эта последняя переменная  $\underline{E}$  является закрытой.

Правая скобка означает конец очередной скобочной структурн. При этом происходит окончательный анализ данного уровня.

Для открытых скобок 

принимает значение I, для закрытых 0.

Закрытая переменная 

принимает код 700. Для этого исполь—

зуется адрес, хранящийся в СІ последней строки стэка.

При появлении символа  $\sim$  работа блока заканчивается и управление передается на блок записи левой части и массив предложений общей памяти (см. рис. 15).

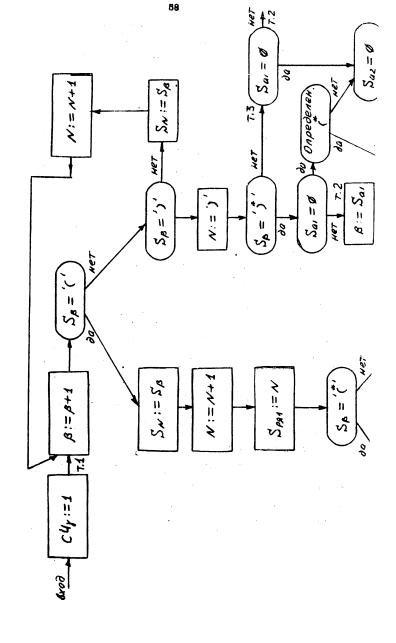
Если присутствует указатель направления синтаксического отождествления, то левая часть предложения записывается в массив ТЗ в обратном порядке. При этом левая скобка заменяется на правую, а правая — на левую; указатель переменной и ее идентификатор меняются местами.

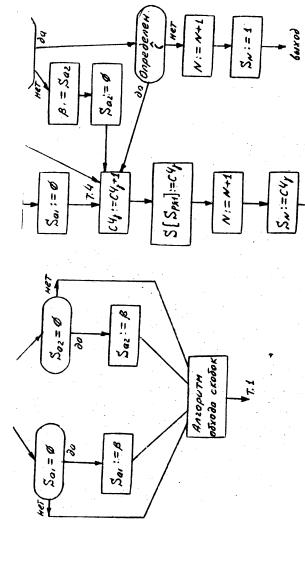
Запись левой части предложения начинается сразу после ячейки 1000. Массив памяти машини, в который записывается предложение, мы как и раньше обозначаем через  $\mathcal{N}$ . На блокскеме не показывается плотная упаковка символов в ячейке, но подразумевается, что, например,  $\mathcal{S}_{\mathcal{N}} = \mathcal{S}_{\mathcal{S}}$  означает включение подпрограмми, записывающей символ в первую свободную позицию ячейки памяти (напомним, что каждая ячейка имеет иять таких позиций).

Исходной информацией для данного олока служит послед овательность символов, расположенная в массиве ТЗ. Текущий адрес этого массива мы обозначаем через В. И здесь выбор следующего символа означает просто считывание символа из следующей ячейки. Перед входом в олок В становится равным адресу начала массива ТЗ.

Назначение блока состоит в реализации принципов отождествление сложных скобочных структур, которые изложени в п.2 гл. IV.

Работа блока начинается с того, что заводится счетик Счу , которому вначале присваивается І. Этот счетчик необходим для того, чтобы каждой скобочной структуре присвоить соответствующий номер. Левая скобка самой внешней скобочной





uc. 13

59

структуры в массив N не записывается, поэтому из массива ТЗ берется следующий символ S = S + 1. Если текущий символ массива ТЗ не левая и не правая скобка, то он записывается в массив N и управление передается на выборку следующего символа в  $7 \cdot 1$ .

Если очередной символ оказался левой скобкой, то она также записивается в массив  $\mathcal{N}$ , после нее резервируется место для номера  $\mathcal{N}:=\mathcal{N}+1$ . Полный адрес номера (адрес ячейки и номер позиции) запоминаются в рабочей ячейке РЯІ. После этого проверяется не является ли скобка закрытой  $\mathcal{S}_{\rho}=\mathcal{N}$  (На блок-схеме закрытая скобка обозначается  $\mathcal{N}$ , а открытая  $\mathcal{N}$ ).

Если скобка оказалась закрытой, то проверяется содержимое рабочей ячейки G1. В G1 хранится адрес начала самой левой закрытой скобочной структуры. Если  $S_{G1} = C^2$ , это значит, что при движении по внешней скобочной структуре данная закрыт тая окобочная структура встретилась впервые. В этом случае в G1 заносится адрес G левой скобки указанной скобочной структуры, а сама скобочная структура при этом пропускается. Это достигается включением алгоритма обхода скобок. Если  $S_{G1} \neq C$ , содержимое G1 не меняется, а управление передается на обход встретившейся скобочной структуры.

В том случае, когда скобка оказалась открытой, проверяется содержимое рабочей ячейки *О2*, которая предназначена для хранения адреса начала самой левой открытой скобочной структуры. Алгоритм проверки точно такой же, как и в случае закритой скобочной структуры.

Коли очередним символом массива ТЗ оказалась правая скобка,

то прежде всего пишется символ конца скобочной структуры  $\bullet$  После этого анализируется сама скобка. Если скобка оказалась закрытой, то проверяется содержимое  $\mathcal{O}$  i

Если  $S_{GI} \neq \mathcal{O}$ , то  $\beta$  становится равным адресу, расположенному в GI,  $S_{GI}$  становится равным 0, счетчик GI увеличивается на I и его содержимое записывается по полному адресу, расположенному в РЯІ. После этого полный адрес  $\mathcal{N}$  сдвигается на следующую позицию, куда и записывается последнее содержимое GI, управление передается в  $\mathcal{I}$  .

Если  $S_{al} = O$  это значит, что при выходе из данной скобочной структуры на ее первом уровне закрытой скобочной структуры на ее первом уровне закрытой скобочной структуры не оказалось. В этом случае включается блок определения закрытой окобочной структуры (см. рис. 14) во внешней по отношению к данной скобочной структуре.

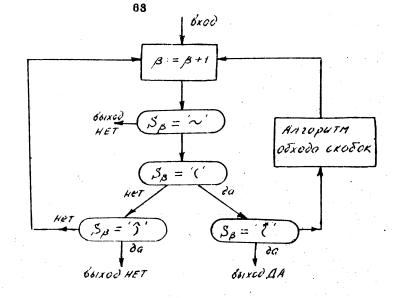
Открытая скобочная структура может внутри себя содержать закрытые скобочные структуры. Поэтому, если правая скобка оказалась открытой, то прежде всего проверяется содержимое Q1
(см. 73 блок-схемы).

Если  $S_{c} \neq C$  , то управление передается в 7.2 . Если  $S_{c} \neq C$  это значит, что ни внутри ни во вне данной скообочной структуры закрытых скообочных структур нет (в расчет не принимаются согласно п.2 гл.П закрытые скообочные структуры, находящиеся внутри открытых). Поэтому блок определения закрытых скообочных структур не включается, а проверяется содержимое ячейки  $C_{c} = C_{c}$ 

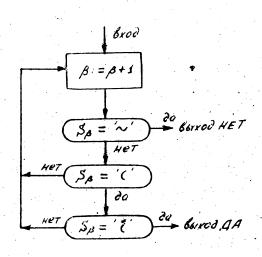
Если  $S_{G2} \neq Q$  , то  $\beta$  принимает адрес начала соответствуюшей открытой скосочной структуры  $\beta = S_{G2}$  , Q2 становитоя равным 0 и управление передается в 74 . Если  $S_{62} = 0$ , то определяется адрес начала самой левой открытой скобочной структуры во внешней по отношению к данной скобочной структуре.

На рис. 14 изображена блок-схема алгоритма определения адреса начала закрытой скобочной структуры. Каждый символ массива 72 провернется не является ли он  $\sim$  . Если да, это означает конец просмотра. До тех пор, пока такой символ не появился, выпеляются встречающиеся левые скобки. Каждая такая скобка затем проверяется не является ли она закрытой. Если да, это значит, что мы нашли очередную закрытую скобочную структуру. В этом случае адрес В как раз равен адресу начала скобочной структуры. Если левая скобка оказалась открытой, то такая скобочная структура пропускается, для чего включается ангорити обхода скобок. Если появляется правая открытая скобка, это также означает конец просмотра, т.к. вне данной скобочной структуры уже не может быть закрытых скобочных структур, **которые бы не находились внут**ри открытых скобочных структур. А также скобочные структуры согласно принципам п.2 гл.П не обладают приоритетом перед открытыми структурами.

На рис. 15 изображена блок-схема агоритма определения адреса начала открытой скобочной структуры. Данный алгоритм включается только тогда, когда в массиве ТЗ остались непереписанным только открытие скобочные структуры, которые внутри себя в свою очередь могут содеражть закрытые скобочные структуры. Алгоритм выделяет ближайную левую открытую скобочную структуру. При этом встречающиеся закрытые скобочные структуры не пропускаются, а происходит вхождение внутрь их, т.к. внутри себя они могут содержать непереписанные открытые скобочные



Puc 14



Puc 15

отруктуры. Когда находится адрес открытой левой скобки, управление передается в 74 блок-схемы на рис. 3 Если встречается символ  $\sim$ , это означает конец работы всего блока переписи левой части и управление передается на записы правой части предложения.

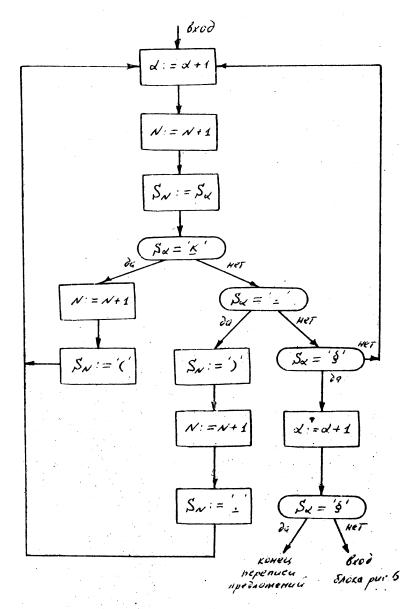
## 3.2.4. Запись правой части предложения

Правая часть предложения так же, как и левая часть, записивается в массив предложений общей памяти плотно упакованная ( по 5 символов в ячейке). Запись правой части начинается следом за левой. Знак ~ в память не пишется. Вместо него в информационной ячейке указывается полный адрес начала правой части.

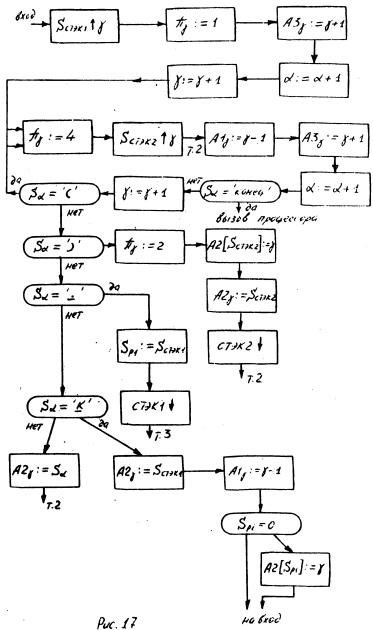
На рис. 16 изображена блок-схема блока записи правой час? 
ти. Ми вновь здесь обращаемся к массиву « . В данном блоке посимвольно переписивается содержимое массива « в массив » . Когда встречается символ « , после него записивается левая скобка. Перед . записивается правая скобка. Так что область действия каждого символа конкретизации заключается в скобки. При появлении символа § проверяется не является ли следующий символ также §. Если это так, это означает конец записи предложений общей памяти. Если нет, передается управление в 7 блока на рис. 6 на запись следующего предложения общей памяти.

## 3.2.5. Запись поля зрения

На рис. 17 изображена схема блока записи поля зрения (БЗПЗ). БЗПЗ записивает информацию в память машини в соответствии с принципами, описанними в п.І гл.П. Каждому символу поля зрения отводится отдельная ячейка рпамяти машины БЭСМ-6.



Puc 16



Каждое адресное поле (AI, A2 и A3 (см. рис. 1) ) составляет 15 разрядов в соответствии с адресностью машини БЭСМ-6.

В блок-схеме текущий адрес массива поля зрения обозначается через  $\mathcal J$  . Переписываемый массив по-прежнему обозначается через  $\mathcal L$  . Обозначение частей отдельной ячейки совпадает с обозначениями п.1 гл.П (см.рис.  $\mathcal I$  ).

Для установления связи между скобками и символами заводятся два стэка:

стэк I - для символов K ,

стэк 2 - для скобок.

Заполнение адресов связи для значащего символа ясно из олок-схемы.

Когда встречается левая скобка, ее адрес посылается в стак 2. При появлении правой скобки ее адрес посылается в АЗ для данной скобки. Кроме этого по адресу, расположенному в стак 2, заносится текущий адрес, соответствующий правой скобке. Таким образом связываются парные скобки. Таким образом связываются парные скобки. Связь с предыдущими и последующим символами устанавливаются так же, как и в случае значащего символа.

Алгоритм установления связи у символов К следущий. Кроме стака I, заводится еще рабочая ячейка Р1 . При появлении символа К в стак I заносится его адрес в поле зрения. Если этот символ первый, то производится только связь его о последующим символом. Если такой символ уже был записан, то в А2 текущей ячейки заносится адрес, находящийся в последней строке стака I. Проверяется также содержимое ячейки Р1 Если она не пуста, то в А2 ячейки, имеющей адрес, находящий-

оя в Р1, заносится текущее значение адреса поля зрения.

В ячейку P1 каждый раз, как встретится символ ..., посылается адрес из последней строки стэка I. Сама точка заменяется на правую скобку и связывается с левой скобкой, которая стоит сразу же за тем символом K, которому соответствует данная подчеркнутая точка. При появлении ... каждый раз последняя строка стэка I уничтожается.

Работа данного блока заканчивается, когда встречается символ КСНЕЦ. В этом случае на место препроцессора с ленты записивается процессор вместе с машинными операциями и работа отдается данному блоку.

На рис. 18 приведен пример поля зрения, содержащего только символы K и \_ , и последовательная работа вышеописанного алгоритма. Связи между символами K указываются с помощью стрелок. Из примера нетрудно видеть, что алгоритм установления связей между символами K полностью соответствует требованиям п.1. гл.П.

На рис. 19 приведено поле фения обдержащее значащие символи, скобки, символи <u>К</u> и . . В данном примере также показан окончательный результат работы блока записи поля эрения в память машины. Адреса ичеек указываются слева.

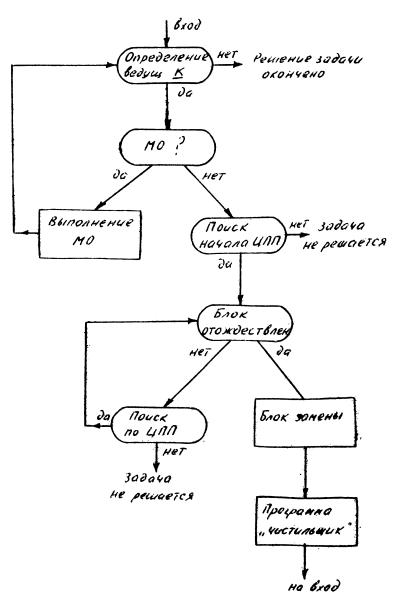
Описанием данного блока мы заканчиваем описание препроцессо-

## 3.3. Процессор

Процессор интерпретирует шаг РЕФАЛ-машини [3]. На рис. 20 изображена общая схема шага. Каждый шаг начинается с отыс-кания ведущего символа конкретизации. Если первый символ кон-кретного выражения (так в дальнейшем мы будем называть область

KKK .KK ..KKK ...K ...K ... KIKI K(K(K() K(K(K()K(K K(K(K()K(K())K(K()) K(K(K()K(K())K(K(K( K(K(K()K(K())K(K(K()))K( K(K(K()K(K())K(K(K()))K())K())

K	20	1	0	38	21
(	21	4	20	37	22
	22	0	21	~	23
	23	0	22	. β	24
	24	0	23	+	25
(	25	4	24	36	26
	26	0	25	∠ .	27
	27	0	26	ß	28
	28	0	27	-	29
K	29	1	28	20	30
(	30	4	29.	33	31
	31	0	30	d	32
	32	0	31	8	33
	33	2	32	30	34
' . 	34	0	33	+	35
	35	0	34	Y	36
)	36	2	35	25	37
)	37	2	36	21	38
K	38	1	- 37	0	39
(	39	4	38`	42	40
	40	0.	3 <i>9</i>	٤	41
	41	0	40	f	42
)	42	2	41	39	. 0



Puc. 20

действия ведущего символа конкретизации) есть идентификатор машинной операции, то данная машинная операция выполняется и управление передается в начало шага. В противном случае с помощью таблицы детерминативов определяется вершина соответствующей иерархии и в ней находится начало ЦПП. С помощью блока синтаксического отождествления проверяется, подходит ли очередное предложение в ЦПП для данного конкретного выражения или нет. В случае удачного отождествления производиися замена левой части предложения на правую. После этого включается программа "чистильшик" и управление передается на начало шага.

Работа интерпретатора заканчивается, когда в поле зрения не останется ни одного символа K .

Если при псиске начала ППП интерпретатор не находит соответствующего предложения, считается, что в записи алроритма на РЕФАЛа допущена ошибка. В этом случае задача с заданным набором предложений общей памяти не решается. Точно так же расценивается и тот случай, когда в ППП не находится подходящего предложения для данного конкретного выражения.

При онисании процессора мы подробно остановимся на работе двух основных блоков: блока отождествления и блока замены левой части предложения на правую. Елока определения ведущего символа к фактически не существует. В самом начале работы процессора адрес ведущего символа указывается в специальной ячейке. В начале работы очередного шага этот адрес определяется с помощью адресов связи у символов К (см. п.1 гл.П ).

Что касается блока выполнения машинных операций, то в указываются те минимальные требования, которые предъявлят РГФАЛ-интерпретатор к машинным операциям, вводимым самим пользоватолем.

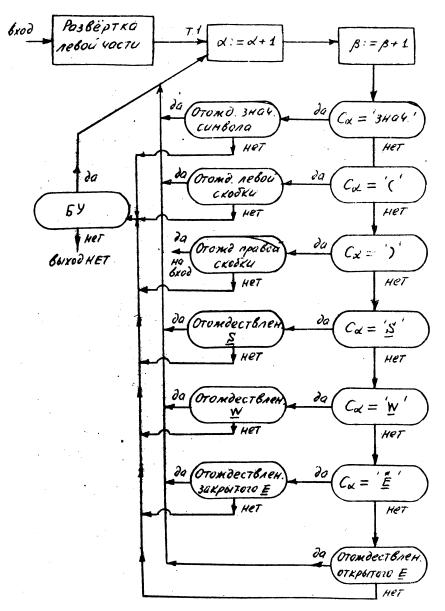
Мы также не будем описывать олоки: поиск начала ЦПП, движение по ЦПП и программу "чистильник", так как о работе этих блоков достаточно подробно говорилось и в главе П и предндущем разделе настоящей главы.

## З.З.І. Блок отождествления (БО)

Блок отождествления осуществляет алгоритм синтаксического отождествления [5]. Исходной информацией для данного блока является: начало левой части соответствующего предложения общей памяти и начало конкретизируемого выражения.

Как уже указывалось, в языке РЕФАЛ предусмотрены два алгоритма синтаксического отождествления прямий и обратный. Реализация этих алгоритмов, по существу, различается линь направлением движения по конкретному выражению. Движение но левой части предложения (в дальнейшем мы будем называть ее абстрактным выражением) в обоих случаях — одно и то же, так как при обратном отождествленная левая часть записывается в намать машины в обратном порядке. Поэтому в настоящем разделе мы опишем только прямой алгоритм синтаксического отождествления.

На рис. 21 изображеня общая схема БО. Работа БО начинается с развертки левой части. Напомним, что левая часть предлежения хранится в памяти машины в плотно упакованном виде ( по 5 омиветов в ячейке). Такая слатая запись при отождествлении ожазивается неудобной. Для эффективного хождения по абстрактному выражению удобно каждый его символ хранить в отдельной ячейке. Блок развертки левой части как раз осуществляет эту операцию. При этом развертывается не вся левая часть. Сначала разверти-



Puc. 21

вается скобочнае структура нулевого уровня, а затем, если отождествление на данном уровне прошло успешно, развертиваются скобочные структуры следующего уровня.

В случае удачного отождествления всего предложения левая часть предложения разворачивается полностью. При неудачном отождествлении может случиться так, что они будет развернута только частично, что позволяет экономить время.

Развертка левой части осуществляется в специальном массите памяти. Обозначим текущий адрес такого массива через «

При синтаксическом отождествлении происходит движение по абстрактному и конкретному выражению слева направо (текущий адрес конкретного выражения на схеме обозначен буквой  $\beta$ ). Каждый символ абстрактного выражения проверяется и при этом выделяются семь частных случаев: очередной символ может быть значащим символом, левой скобкой, правой скобкой, неременной S, переменной V, закрытой переменной выражения F, открытой переменной выражения F. Для каждого из этих частных случаев существует свой алгоритм отождествления. При удачном отождествлении происходит дальнейшее движение по абстрактному и конкретному выражению.

При отождествлении правой скобки развертывается следунная скобочная структура. Если перед этим била отождествлена последняя скобочная структура, это означает конец работы блока отожде ствления, и управление передается на блок замены левой части предложения на правую.

В случае неудачного отождествления какого-либо из указанных частных случаев управление передается в блок удлинения (БУ). БУ хранит информацию о всех отождествленных открытых переменных Е . В первую очередь удлиняется самая последняя из них. Если до данного момента не было отождествлено ни одной открытой переменной Е , удлинение невозможно. В этом случае управление передается на поиск следующего предположения в ППП.

Остановимся более подробно на описании отдельных подблоков блока синтаксического отождествления.

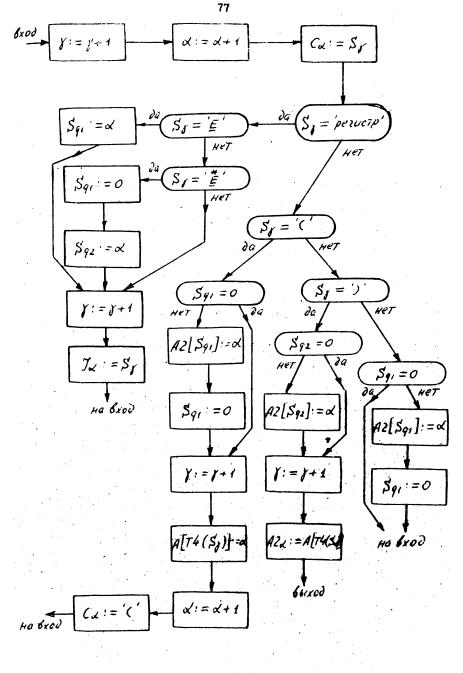
3.3.1.1. Развертка левой части предложения .

На рис. 22 изображена блок-схема алгоритма переписи левой части предложения в развернутом виде.

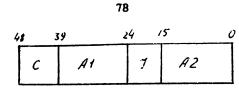
Текущий адрес массива, в котором записана левая часть в сжатом виде на схеме обозначен через У . Текущий адрес массива, на который переписывается левая часть, обозначается через ✓ .

Идентификатор частей ячейки массива < показана на рис. 23 где:

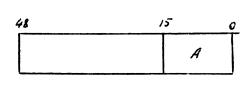
- С содержит код символа.
- Содержит информацию о конкретном выражении для опорного символа и правой заключительной скооки,
- А2 для открытой переменной Е содержит адрес опорного символа, для закрытой Е адрес правой заключительной скоски данной скосочной структуры, для заключитель ной скоски А2 содержит адрес левой скоски, за которой следует такой же номер, что и за данной заключи тельной скоской.
  - 7 идентификатор переменной.



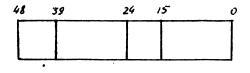
Puc. 22



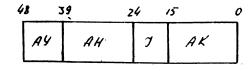
Puc 23



Puc 24



Puc. 25



В процессе переписи происходит заполнение указанных частей ячеек массива  $\checkmark$  . Для этого вводятся две рабочие ячейки  $\ref{eq:total_self}$  и  $\ref{eq:total_self}$  , а также таблица  $\ref{eq:total_self}$  .

Адрес каждой открытой переменной  $\mathcal{E}$  запоминается в ячейке 9/ . И при дальнейшей переписи адрес первого встретивнегося символа, отличного от переменной и правой скооки, посылается в A2 ячейки, адрес которой находится в 9/ . Такой символ в дальнейшем мы и будем называть опорным символом.

Точно так же посылается адрес правой скобки в ячейку, соответствующую закрытой переменной E . Только в этом случае адрес E запоминается в ячейке 92 .

Связь между левой и правой скобкой, имеющих одни и те же номера (относительно номеров, относящихся к скобкам, см. п.2 гл.П), осуществляется с помощью таблицы Т4. В этой таблице каждому номеру отводится одна ячейка (см.рис. 24). В часть А такой ячейки записнвается адрес левой скобки, имеющей данный номер. При развертывании очередной скобочной структуры адреса всех левых скобок, содержащихся внутри неее будут занесены в таблицу Т4. Следует также сказать, что при развертие левой части после каждой левой скобки записивается соответствующая правая скобка.

Работа блока заканчивается после записи заключительнай скобки данный скобочной структуры.

3.3.1.2. Отождествление значащих символов.

Когда в абстрактном выражении встречается значаний символ, то он просто сравнивается с очередным символом конжретного выражения. Если символы оказались одинаковыми, управление передается на продолжение движения по абстрактному и конкретному выражению (  $\tau$ . I на блок-схеме рис. 21 ). При несовпадении управление передается в БУ.

3.3.1.3. Отождествление левых скобок.

При отождествлении левых скобок также проверяется их соответствие символам конкретного выражения.

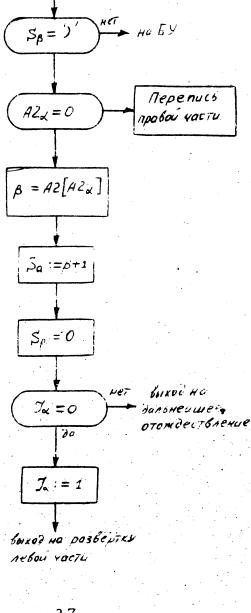
При совпадении в левую скобку абстрактного выражения ( в часть A2) посылается адрес соответствующей ей левой скобки конкретного выражения. Это делается для того, чтобы в дальней-шем имелась возможность вернуться и отождествить скобочную структуру, ограниченную данной левой скобкой. Левая и правая скобка абстрактного выражения, а также вся скобочная структура конкретного выражения пропускается, и отождествление продолжается на нулевом уровне соответствующей скобочной структурнь.

Таким образом, после отождествления очередной скобочной структури все встретившиеся левые скобки с помощью адресов, расположенных в £2, оказываются связанными с левыми скобками пропущенных скобочных структур конкретного выражения.

3.3.1.4. Отождествление правих скобок.

Появление в абстрактном выражении правой заключительной скобки означает, что данная скобочная структура отождествлена полностью. Блок-схема алгоритма отождествления правой скобки изображена на рис. 27.

Информация, содержащаяся в ячейке правой скобки, используется для дальнейшей работы интерпретатора.



DUC. 27

Если  $A2_{\times}=0$ , это значит, что левая часть предложения отождествлена полностью. В противном случае необходимо продолжить отождествление пропущенных скобочных структур. Для этого по адресу связи, находящемуся в A2 правой заключительной скобки, находится адрес связанной с ней левой скобки абстрактного выражения. А поскольку эта скобка уже связана с соответствующей скобкой конкретного выражения, тем самым находится адрес начала пропущенной скобочной структуры конкретного выражения  $\beta := A2 \left[A2_{\times}\right]$ 

Информация, заносимая в рабочие ячейки  $G \bowtie P$ , необ-ходима для работы других блоков.

3.3.1.5. Отождествление переменных.

Кроме значащих символов и скобок в абстрактном выражении могут вотретиться спободние переменные S, W H E Интерпретатор среди переменных E различает закрытые и открытие.

Каждой переменной в соответствии с ее интаксическим типом должна быть сопоставлена группа символов конкретного выражения. Причем, переменным с одним и тем же идентификатором должни быть сопоставлени идентичные группы символов.

Для запоминания содержимого отождествленных переменных заводится таблица ТІ, состоящая из 16 ячеек. На рис. 25 изображена такая ячейка. В АН заносится адрес начала содержимого переменной, а в АК — адрес конца.

Обращение и таблице ТІ осуществляется с помощью идентификатора переменной.

Например:

Пусть ( ...  $\frac{1}{2}$  ... ) — абстрактное виражение, а  $\frac{1}{2}$  ...  $\frac{1}{2}$  ...

Если какая-то переменная в абстрактном выражении встречается не первый раз, ее содержимое в конкретном выражении набирается сравнением с содержимым ранее отождествленной переменной проверяется так называемие тождество одинаковых переменных. Тот факт, что данная переменная встретилась не впервый раз, определяется с помощью таблицы Т2.

В таблице Т2 хранятся счетчики числа отождествленных переменных с одним и тем же идентификатором. Одинаковым переменным в такой таблице соответствует одна ячейка. Число ячеек в таблице Т2 равно числу возможных.

Перед отождествлением очередной переменной с помощью ее идентификатора проверяется содержимое соответствующего счетчи-ка таблицы Т2. Если счетчик равен 0, производится обычное отождествление переменной. Если нет, то проверяется тождество.

При выполнении удлинения приходится возвращаться назад, к удлинению уже отождествленной открытой переменной  $\mathcal{E}$ . При этом происходит как бы обратное движение по абстрактному выражению. При этом обратном движении могут встретиться переменные. В таких случаях из соответствующих счетчиков таблицы Т2 отнимаются единички. Такой механизм счетчиков точно эпределяет когда нужно проводить тождество.

Для более эффективного осуществления алгоритма удлинение и замени вводится таблица ТЗ (см.рис. 26). Ее точнее можно было бы назвать стэком переменных. Таблица ТЗ содержит 32 ячейки. В нее по мере отождествления переменных заносятся их идентификаторы (часть Г). Кроме этого, для некоторых переменных W и E' в эту таблицу заносятся адреса начала и конца (соответственно, АН и АК). Для открытой переменной E в АУ таблицы ТЗ заносится ее адрес в абстрактном выражении (АУ занимает 9 разрядов, что вполне достаточно для указания адреса в массиве С, так как он ограничен 100 ячейками и находится в постоянном месте памяти).

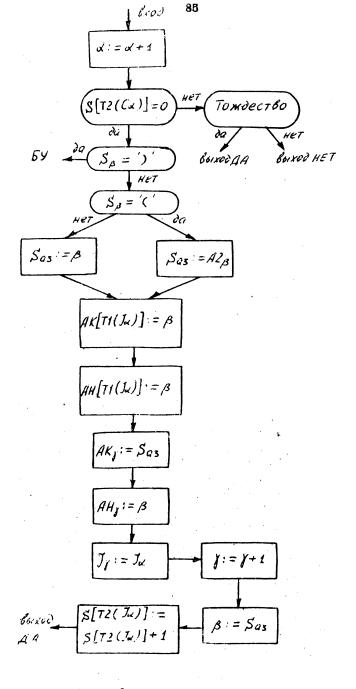
Отождествление переменных S и W совершенно одинаково в том случае, когда значением W является значащий символ. Поэтому в дальнейшем из этих двух блоков мы опишем только блок отождествления W.

3.3.I.6. Отождествление переменной  $_{igotimes}$  .

На рис. 28 изображена блок-схема алгоритма отождествления переменной w .

Прежде всего проверяется содержимое соответствующего счетчика таблици Т2. Если оно равно 0, то проверяется не является ли очередной символ конкретного выражения правой скобкой. Если это так, то управление передается в БУ (блок удлинения). В противном случае считается, что в конкретном выражении стоит либо значащий символ, либо левая скобка. И в том и в другом случае происходит заполнение таблиц Т1, Т2, Т3.

Сначала в таблицу ТІ заносится адрес конца и начала содержимого переменной:  $AK[T1(J_M)] := S_{GS}$  и  $AH[T1(J_M)] := \beta$ Затем заполняется очередная ячейка таблицы ТЗ ( текущий адрес обозначается буквой f):  $AK_f := S_{GS}$ ,  $AH_f := \beta$ и  $J_f := J_M$ . Счетчик таблицы Т2, соответствующий



Puc. 28

идентификатору переменной, увеличивается на I:  $S[T2(J_{4})] := S[T2(J_{4})] + 1$ .

Перед виходом из блока адрес конкретного выражения становится равным адресу конца переменной и управление передается на продолжение отождествления (см.т.І блок-схемы на рис. 21)

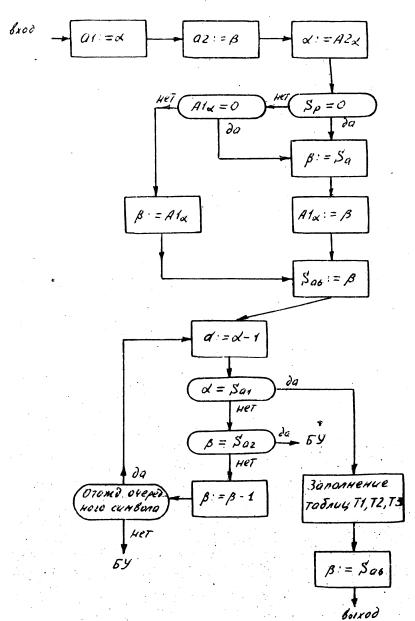
Алгоритм проверки тождества переменных одинаков для всех переменных и в общем-то достаточно прост. Поэтому специально на нем ми не останавливаемся. Укажем только, что после успешного отождествления содержимое таблицы ТІ не меняется, однако как всегда счетчик таблицы Т2 увеличивается на І, а в таблицу Т3 заносится информация об очередной отождествленной переменной. В случае неудачного отождествления управление передается на БУ.

3.3.1.7. Отождествление закрытой переменной E .

На рис. 29 изображена блок-схема алгоритма определения содержимого закрытой переменной  $\underline{\varepsilon}$ . При этом предполагается, что данная переменная в абстрактном выражении встречается впервые. В противном случае как и для переменной  $\underline{w}$  включается алгоритм проверки тождества.

По адресу, хранящемуся в A2 ячейки переменной  $\underline{\mathcal{E}}$  , находится правая заключительная скобка, ограничивающая скобочную структуру, внутри которой находится данная переменная.

В конкретном выражении также производится переход к соответствующей правой скобке по адресу, хранящемуся в рабочей ячейке Q (см. блок-схему рис. 29), либо по адресу, хранящемуся



Fuc. 29

в АІ ячейки абстрактного выражения, соответствующей правой заключительной скобке данной скобочной структуры. По какому именно адресу зависит от того, вошли ли мы в эту скобочную структуру обывным путем ( $\beta \rho = \mathcal{O}$ ) или же из БУ. Во втором случае АІ ячейки правой заключительной скобки может оказаться нулевым. Это будет означать, что управление в БУ было передано из той скобочной структуры, отождествление которой производится в данный момент. В этом случае так же, как и при  $\beta \rho = \mathcal{O}$ , адрес правой скобки конкретного выражения определяется по сощержимому рабочей ячейки  $\mathcal{O}$ .

После того, как найдени правие заключительные скобки в абстрактном и конкретном выражении, направление синтаксического отождествления меняется на противоположное.

При движении в абстрактном выражении справа налево встречающиеся значащие символы, скобки и переменные  $\sum$  и  $\underline{w}$  отождествляются обычным образом. Елок заканчивает свою работу после того, как в абстрактном выражении вновы встретится закрытая переменная  $\underline{E}$ . В этом случае оставщаяся часты конкретного выражения относится к содержимому данной переменной. Для этого заполняются таблицы ТІ. Т2 и Т3.

Отождествление будет неудачным в следующих двух случаях:

I) Когда при движении по конкретному выражению мы переходим границу начала содержимого переменной E. Например, пусть имеется абстрактное выражение

и конкретное выражение

При обратном паижении для отождествления 52 нам бы

пришлось перейти через символ  $\propto$  ,стоящий в конкретном выражении. Это является недопустимым, и поэтому управление передается в  $\text{БУ}_{\bullet}$ 

2) Когда при обратном движении произошло неудачное отождествление какого-то встретившегося символа абстрактного выражения.

Например, если у нас имеется абстрактное внражение

и конкретное выражение

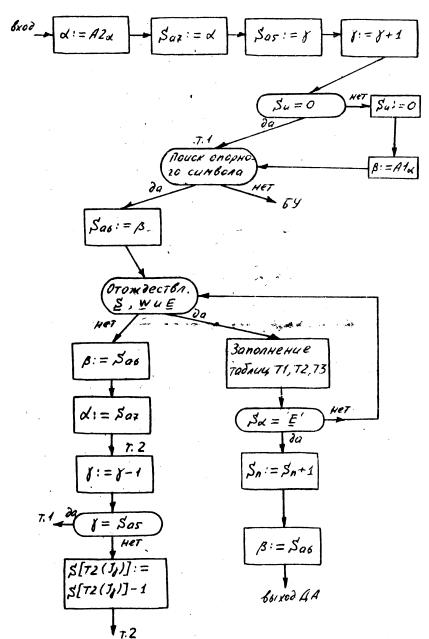
то они не могут бить отождествлены друг с другом. В самом деле, при движении справа налево переменная W1 примет значение ( . . ), а следующий символ абстрактного и конкретного выражения не совпадают. В этом случае управление также передается на БУ. Если закрытая переменная при отождествлении оказывается пустой, то в ТЗ заносится только ее идентификатор, а в ТІ только адрес очередного символа конкретного выражения.

3.3.1.8 Отождествление открытой переменной  $\underline{\mathcal{E}}$  .

На рис. 30 изображена блок-схема алгоритма определения содержимого открытой переменной в том случае, когда она имеет опорный символ в абстрактном выражении. Работа блока состоит в следующем.

Прежде всего определяется опорный симвом в абстрактном выражении. Для этого используется адрес находящийся в A2 ячей-ки, соответствующей данной переменной E. Затем включается блок поиска символа, равного опорному, в конкретном выражении.

После того как найдени опорние символи, дальнейшее



Puc.30

отождествление — становится аналогичным отождествлению закрытой переменной  $\not\in$  . Также меняется направление отождествления и также все встречающиеся переменные должны быть отождествлены. Если же какая—либо из переменных не отождествляется, управление не передается в БУ. В таком случае производится, так называемое, местное удлинение.

Местное удлинение формально полностью совпадает с обычным удлинением. В интерпретаторе оно выделяется только потому, что его легче осуществить, чем обычное удлинение.

При местном удлинении формально направление движения по абстрактному выражению меняется вновь на обратное. И движение осуществляется вплоть до опорного символа. При этом для всех встретившихся переменных из соответствующих счетчиков таблицы Т2 единички вычитаются. Фактически же в интерпретаторе мы сразу находим адреса опорных символов, используя содержимое рабочих ячеек А6 и А7, а обратное движение осуществляется по таблице Т3 ( на схеме текущий адрес обозначен буквой ). При этом идентификаторы переменных, хранящихся в этой таблице, используются для обращения к таблице Т2.

Приведем пример местного удлинения. Пусть  $( \dots E^1 \le 2 \le 2^+ \dots ) - \text{абстрактное выражение, а}$   $( \dots ( ) + (A-B) + AB + AA + \dots ) - \text{конкретное выражение.}$ 

В данном примере переменная £1 будет удлиняться три раза. После того как первый раз будет найден опорный символ в конкретном выражении (первый слева знак +), отождествление будет неудачным, так как переменная 52 не может быть отождествлена с правой скобкой.

Местное удлинение состоит в том, что в конкретном выражении в качестве опорного символа будет взят второй знак +. В этом случае также отождествление будет неудачним. И только тогда, когда в качестве опорного символа будет взять четвертый знак +, отождествление будет удачным. Переменная 52 примет значение A, а переменная 52 станет равной ( ) + (A-B)+AB+

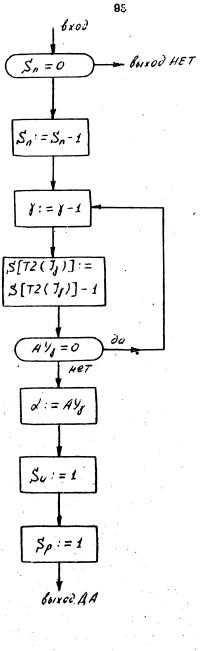
На этом же примере можно показать необходимость вторичного движения к опорному символу и вычитания единицы из счетчиков таблицы Т2. Действительно, при вторичном удлинении ЕІ
опорным символом в конкретном выражении будет третий слева
знак +. Отождествление в этом случае не произойдет из-за того,
что не выполняется тождество одинаковых переменных \$2.
Однако в этом случае правая \$2 все-таки примет значение \$
и в таблице Т2 во второй ячейке будет записана I. Если бы
мы при возврате к опорному символу не вычитали I из второй
ячейки таблицы Т2, то отождествление \$2 после третьего удлинения переменной ЕІ начиналось бы с проверки тождества. И
отождествление бы не произошло.

При существующем алгоритме единица вычитается, и  $\leq 2$  при новом отождествление предстает как впервые встретившаяся псременная.

В рабочей ячейке n заводится счетчик числа отождествленных открытых переменных E . Эта информация необходима для БУ.

3.3.1.9. Блок удлинения (БУ)

На рис. 31 изображена блок-схема алгоритма определения адреса последней из числа отождествленных открытой переменной E. Для этого используется таблица ТЗ (на блок-схеме теку-шля адрес этой таблица обозначен буквой F).



Puc. 31

БУ идет по таблице ТЗ в обратном направлении и использует идентификаторы переменных (  $\mathcal{I}_j$ ) для обращения к таблице Т2. Для всех переменных, за исключением открытых, АУ ячейки таблицы ТЗ равно 0. Так что первая же встретившаяся переменная, у которой  $A\mathcal{Y}_X \neq \mathcal{O}$ , объявляется удлиняемой.

Содержимое рабочих ячеек  $\omega$  и  $\rho$  используется для отождествления переменной E (см. 3.1.7 и 3.1.8 ).

После того, как найдена удлиняемая переменная E, отождествление с этого места абстрактного выражения осуществляется так же как и при первом проходе. При этом все те переменные, которые уже были отождествлены, отождествляются вновь.
Единственное отличие состоит в том, что те скобочные структуры, которые были развернуты при первом проходе, второй раз
не разворачиваются.

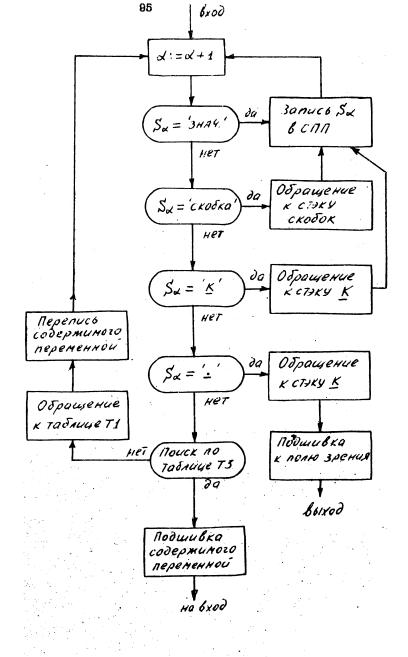
### 3.4. Замена левой части предложения на правую

На рис. 32 изображена блок-схема алгоритма замены левой части предложения на правую.

Исходной информацией для данного олока является адрес начала СШІ (свободного поля памяти), полний адрес начала правой части и содержимое таблицы Т2 и Т3.

Ячейки СШІ уже связаны должным образом (см. 5 гл.П), поэтому, если в правой части встречается значащий символ, его код просто записывается в соответствующую часть ячейки СШІ.

Париме скобки связиваются между собой с помощью адресов связи, помещаемих в A2. При этом работает точно такой же алгорити, как из в препроцессоре (см.блок-схему рис.  $1 \neq 1$ ). Для этого заводится стак скобок, к которому производится



обращение, когда в правой части появляется левая или правая скобка.

Алгоритм связи символов K совпадает с аналогичным алгоритмом препроцессора. Для этого заводится второй стэк – стэк K .

Когда встречается свободная переменная, производится обращение к таблице ТЗ. В ней находится ближайшая от начала ячейки, соответствующая данной переменной. Адреса начала и конца данной переменной используются для подшивки содержимого переменной в правой части предложения. При этом найденная ячейка таблицы ТЗ помечается, что говорит о том, что данная переменная уже была использована для замены. Если такая переменная в дальнейшем встретится еще раз, помеченные ячейки таблицы ТЗ, соответствующие данной переменной, будут пропускаться.

Если все ячейки таблицы ТЗ, относящиеся к одинаковым переменным, будут помечены и если, тем не менее, такая переменная
в правой части встречается еще раз, это говорит о том, что
переменная в правой части встречается в большем числе раз, чем
в левой части. В этом случае производится обращение к таблице
Т2, и для переписи используются адреса начала и конца указанной переменной. При этом все содержимое переменной записывается в СПП. Таким образом, фактическая перепись переменной производится только в этом частном случае.

3.5. <u>Параметри РЕФАЛ-интерпретаторы для БЭСМ-6</u> Размеры основных блоков (см. рис. 4):

БСК - 600<sub>8</sub> ячеек ;

БПИ - 300<sub>8</sub> ячеек;

Препроцессор - 4000<sub>8</sub> ячеек;
Процессор -8000<sub>8</sub> ячеек.

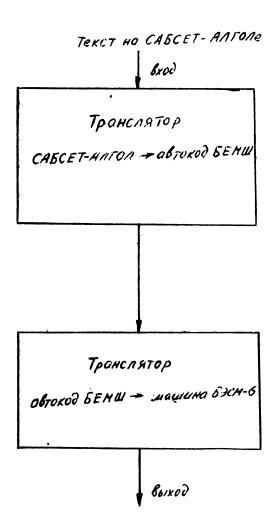
Скорость работы интерпретатора  $\approx$  300 + 400 шагов/сек.

#### 4. Некоторые применения РЕФАЛ-интерпретатора

РЕФАЛ-интерпретатор для БЭСМ-6, описанный в разделе 3, отлаживался на двух крупных задачах: трансляторе с АЛГОЛа и программе преобразования алгебраических выражений. Приведем краткие введения об этих программах.

# 4.I. Транслятор с САБСЕТ-АЛГОЛА в автокод БЕМШ для машины БЭСМ-6

Схема использования транслятора, написанного на РЕФАЛе, показана на рис. 33. Описание транслятора на РЕФАЛе вводится в поле памяти РЕФАЛ-интерпретатора. В данном случае оно состояло приблизительно из 550 предложений, занимавших около 4 тыс. ячеек машины БЭСМ-6. Текст на входном языке (CABCET-АЛТОЛе), заключенный в "скооки" К ТРАЛГ И вводится в поле зрения РЕФАЛ-интерпретатора (ТРАЛГ -детерминатив процедуры "трансляция с АЛГОЛа"). Интерпретатор произво+ дит переработку текста на входном языке в текст на выходном языке (автокол БЕМШ) в соответствии с набором предложений в его памяти. Таким образом, процесс трансляции происходит в режиме интерпретации, этот процесс напоминает работу транолятора "управляемого синтаксисом". Результат-программа на автокоде БЕМП - является конечно, скомпилированной эффективной программой, то есть транслятор, описанный на РЕФАЛе, есть обичный транслятор-компилятор. В процессе трансляции интерпретатог выдает сообщения с синтаксических ошибках в АЛГОЛ-



puc. 33

программе. В частности, ошибки могут быть таковы, что процесс трансляции будет остановлен с сообщением "Конкретизация невозможна". Эта особенность не вытекает с необходимостью из того факта, что транслятор описан на РЕФАЛе, а является индивидуальной особенностью данного транслятора. На РЕФАЛе можно было бы также написать программу предварительного синтаксического контроля.

Транслятор дает программу хорошего качества. В частности, значительные усилия посвящаются оптимизации программирования, выражений, содержащих переменные с индексами.

Транслятор не использует внешней памяти, он расочитан на перевод небольших программ (не более 2-3 тыс.команд авто-кода). Расширение возможностей транслятора в направлении увеличения объема программ может быть достигнуто путем введения машиных операций (см. [3]) обмена с внешней памятью.

Средняя скорость трансляции 50 команд автокода в секунду. Таким образом, РЕФАЛ -интерпретатор позволяет писать на языке РЕФАЛ трансляторы, которые получаются достаточно эффективными для практического использования. Отметим, что написание (включая отладку) транслятора на РЕФАЛе несравненно менее трудоемкая задача, чем на машинном языке или автокоде ( но нашим оценкам в 20-30 раз). Кроме того, различные переделки и модификации транслятора, написанного на РЕФАЛе, например, изменение выходного языка, осуществляются чрезвычайно просто. Отметим так же, что выходной язык транслятора никак не связая с языком машины, для которой написан интерпретатор. Поэтому, с помощью РЕФАЛ-интерпретатора для БЭСМ-6 можно осуществлять трансляцию с любого проблемно-ориентированного языка

на язык любой машины, если только написать на РЕФАЛе соответствующий транслятор.

#### 4.2. Программа упрощения алгебраических выражений.

Эта программа производит упрощение выражений, заданных с помощью подстановок и алгоритмически определенных операторов ( например, дифференцирование, сдвиг и т.п.). Естественное применение эта программа находит в тех случаях, когда надо произвести вычисления (на символическом уровне) по рекуррентным формулам. Эти формулы, так же как и определения используемых операторов, являются своего рода "начальными данными" для программы. Они записываются в виде предложений РЕФАЛа, по форме практически совпадающих с записью тех же формул в обычных обозначениях.

Настоящая программа была использована для решения одной конкретной задачи из области вычислительной математики. С помощью машины были получены формулы, которые ручным способом получить было практически невозможно из-за громоздкости вычислений.

## 5. Благодарность

Авторы выражают благодарность студенту МГУ С:А. Романенко, написаниему блок синтаксического контроля и участвовавшему в обсуждении ряда вопросов.

#### Литература

- В.Ф.Турчин, Метаязик для формального описания алгоритмических язиков, в сб. "Цифровая вичислительная техника и программирование", Сов.радио 1966г.
- 2. В.Ф.Турчин, Метаалгоритмический язык, "Кибернетика", № 4, 1964г.
- 3. В.Ф.Турчин, Алгоритмический язык рекурсивных функций (РЕФАЛ), ИПМ АН СССР, сентябрь 1968 г.
- 4. Bobow D.G., Ruph el & a Comparison of List- Processing compliter Congunges. Comm acm 7,4, april 1964.
- 5. С.Н. Флоренцев, В.Ю. Олюнин, В.Ф. Турчин, РЕФАЛ-интерпретатор в сб. "Трудн I-ой всесоюзной конференции по программированию", г. Киев, ноябрь 1968 г.
- 6. В.С. Штаркман, Автокод БЭСМ-6, ИПМ АН СССР, Москва, 1967г.
- 7. Математическое обеспечение машины БЭСМ-6. Инст. ТМиВТ ВИАН СССР, Москва 1967 г.
- 8. В.Ф.Турчин, В.И.Сердобольский, Язык РЕФАЛ и его использование для преобразования алгебранчёских выражений, ж. "Кибернетика" № 3. 1968г.
- 9. В.Ф. Турчин, С.Н.Флоренцев, В.А.Фисун, Использование языка
  РЕФАЛ для автоматизации программирования.
  Доклад на межвузовской конференции по автомати—
  зации сбработки экономической информации.
  Москва. май 1967г.
- 10. С.Н.Флоренцев, Транслятор с автокода для машин типа БЭСМ-4, написанный на РЕФАЛе в сб. "Труды МИФИ" (в печати>

II. В.Ф.Турчин, Транслятор с АЛГОЛа, написанный на РЕФАЛе в сб. "Труды I-ой Всесоюзной конференции по программированию ", Киев, ноябрь 1968 г.

## ОГЛАВЛЕНИЕ

		OT
I	. Введение	I
2	. Общие принципы эффективной реализации языка	,
	РЕФАЛ на вычислительных машинах	2
	2.1. Расположение информации в поле зрения	
	интерпретатора	2
	2.2. Отождествление скобочных структур	II
	2.3. Замена левой части предложения на правую	, <b>18</b>
	2.4. Объединение предложений в цепочку просмат-	
	риваемых предложений	19
	2.5. Программа чистильщик	30
3.	РЕФАЛ - интерпретатор для машины БЭСМ-6	31
	З.І Язык блок-схем	35
	3.2. Препроцессор	39
	3.3. Процессор	56
	3.4. Замена левой части предложения на правую	71
	3.5. Параметры РЕФАЛ-интерпретатора для БЭСМ-6	72
4.	Некоторые применения РЕФАЛ-интерпретатора	73
	4.1. Транслятор с САВСЕТ-АЛГОНА в автокод БЕМН	
	для машины БЭСМ-6	73
	4.2. Программа упрощения алгеоранческих выражений	75
5.	Благодарность	75
	Литература	76

Подписано к печати 12. V/ 1989 г.

М. 892 09 рт " 19 " VI 198 9 г. Заказ № 976 Тираж 300 экз.