ОРДЕНА ЛЕНИНА
ИНСТИТУТ ПРИКЛАДНОЙ МАТЕМАТИКИ
АКАДЕМИИ НАУК СССР

Отдел автоматизации программирования

С.Н.Флоренцев В.Ю.Олюнин

В.Ф.Турчин

ЭФФЕКТИВНЫЙ ИНТЕРПРЕТАТОР ДЛЯ ЯЗЫКА «РЕФАЛ»

ПРЕПРИНТ № 29
за 1969 год

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

Москва, 1969г.

I. ВВЕДЕНИЕ

Язык РЕФАЛ (алгоритмический язык рекуровивных функций) ориентирован на задачи преобразования символьной информации (перевод с формализованных и естественных языков, алгебраические преобразования, машинное доказательство теорем и т.п.). Этот язык возник как специализированное упрощение метаалгоритмического языка ([I , 2]), его описание содержится в [3].

Для практического 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 чрезвычайно неэффективной. Действительно, при выполнении каждого шага она будет просматривать значительную часть поля зрения в поисках ведущего символа конкретизации, затем полностью просматривать все конкретизируемое выражение (возможно, неоднократно) в процессе отождествления и, наконец, полностью переписывать 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 стороны, ясно, что составление транслятора-компилятора с такого языка, как РЕФАЛ- задача весьма сложная и, быть может, без введения каких-то ограничений неразрешимая.

Поэтому 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 вычислительных машин. Эти принципы являются достаточно общими и могут применяться при построении интерпретаторов на различных машинах. Основное внимание уделялось вопросу повышения скорости работы интерпретатора. Решающим моментом на пути 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.

2.1. Расположение информации в поле зрения интерпретатора

Языки, 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 [3].

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 РЕФАЛ-машина предписывает набирать содержимое каждой переменной, посимвольно просматривая поле зрения. Это не всегда бывает необходимо с точки зрения самого существа алгоритма. Поясним это на примерах.

Пусть имеется предложение, которое содержит следующую левую часть:

$$\S\ \underline{K}\,\alpha\,\underline{E}1+\underline{E}2 \sim \ldots \qquad (пр.1)$$

Процедура α 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 $\underline{E}1$. Именно ее содержимое указывает в каком месте расположен ближайший слева

знак + . Содержимое $\underline{E}2$ можно было бы не просматривать, отнеся к нему оставшуюся после знака + часть конкретизируемого выражения. Однако этого сделать нельзя, т.к. нельзя найти адрес конца $\underline{E}2$, не просмотрев его содержимого.

Приведем еще пример:

$$\S\,\underline{K}\beta\,\underline{S}1\,\underline{W}2\,(\underline{E}3+\underline{E}4)\,\underline{E}5\sim\ldots \qquad (пр.2)$$

Предположим, что данное предложение является подходящим для какого-то конкретизируемого выражения. В этом случае скобки левой части предложения оказываются совершенно жестко связанными со скобками конкретизируемого выражения. Внутри этих скобок совершенно необходимо,конечно, набирать содержимое $\underline{E}3$, однако содержимое $\underline{E}4$ можно было бы и не набирать, отнеся к нему символы оставшиеся после нахождения знака + внутри скобок. Но этого сделать нельзя, так как не известен адрес правой скобки.

Содержимое $\underline{E}5$ так же как и содержимое $\underline{E}2$ 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 счёт лишних просмотров содержимого переменных типа $\underline{E}$, не вызванных самой сутью алгоритма.

2.I.2. Каждый раз после того, как полностью выполнена очередная конкретизация, машина ищет ведущий символ конкретизации в соответствии с его формальным определением. Это может быть достигнуто только посимвольным просмотром всего поля зрения. Это,конечно, занимает значительную часть времени выполнения шага.

Для ликвидации указанных потерь эффективности в РЕФАЛ-интерпретаторе принят следующий способ расположения информации в поле зрения.

Каждому символу поле зрения отводится три адресных поля (A1, A2, A3) и признаковая часть $\mathcal{f}$ (2 разряда):

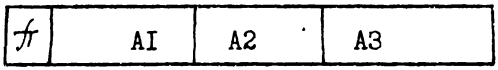


Рис.I

Признак $\mathcal{f}$ определяет, является ли этот символ значащим символом, скобкой или символом конкретизации $\underline{K}$. От содержания $\mathcal{f}$ зависит содержимое $A1$, $A2$ и $A3$.

а) Если $\mathcal{f}$ соответствует значащему символу, то в $A2$ хранится код символа, в $A1$ - адрес предыдущего, а в $A3$ - последующего символа.

б) Если $\mathcal{f}$ соответствует скобке, то в $A2$ хранится адрес парной скобки, в $A1$ и $A3$ - соответственно адрес предыдущего и последующего символа.

в) Если $\mathcal{f}$ соответствует символу конкретизации, то в $A1$ хранится адрес предыдущего символа, в $A3$ - адрес последующего, а в $A2$ - адрес того символа $\underline{K}$, который станет ведущим, когда будет полностью выполнена данная конкретизация. В символе $\underline{K}$ следовало бы также указать адрес соответствующей ему подчеркнутой точки, однако это потребует дополнительного адресного поля. Вместо этого подчеркнутая точка в память машины не пишется, а вся область действия символа $\underline{K}$ заключается в скобки так, что левая скобка располагается сразу же за символом $\underline{K}$.

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:

$$\underline{K}\alpha A(\underline{K}\beta(A\alpha B)_{\perp})(A\alpha C)_{\perp} ,$$

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

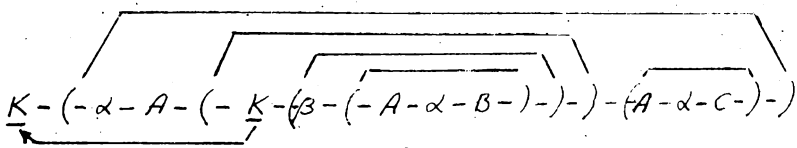


Рис. 2

Наличие адреса связи ($A2$) у парных скобок и символов $\underline{K}$ 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.I.3. При наборе содержимого переменной типа $\underline{W}$ 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:

$$\S\ \underline{K}\ \Psi\ \underline{E}1 + \underline{W}2\ \underline{E}3 \sim \ldots$$

И пусть конкретизируемым выражением является выражение:

$$\underline{K}\ \Psi A B C + ((A+B)-C)+A\ \underline{.}$$

Отождествление данного выражения интерпретатор будет проводить следующим образом. Сначала наберется содержимое $\overset{E1}{\sim}$ 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.I.4. С помощью переменной типа $\underline{E}$ в РЕФАЛе часто организуются алгоритмы просмотра конкретизируемого выражения с целью определения местоположения какого-то конкретного символа в конкретизируемом выражении см. [3] . В дальнейшем такой символ мы будем называть опорным символом. При этом типичной конструкцией левой части является.

$$\S\ \underline{K}\ \Psi\ \underline{E}1\ \alpha\ \underline{E}2 \sim \ldots$$

В данном примере α есть как раз тот символ, который ищется в поле зрения. Согласно синтаксису языка РЕФАЛ ясно, что с помощью такой записи можно найти ближайший слева символ α , расположенный только на нулевом уровне скобочной структуры всего 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

$$\underline{K}\,\Psi A\,(\,A\alpha B\,)_{1\ 1}\,CB\,(\,AB\,(\,CD\alpha E\,)_{3\ 3}\,)_{2}\,\wedge\,(\ \ )\ \underline{+}\ \ ^{x)}$$

РЕФАЛ-интерпретатор набор содержимого $\underline{E}1$ будет проводить следующим образом: сравнит символ A с α, перепрыгнет через скобки $(\ \ldots\ )_{1\ 1}$, сравнит с α символы C и B, перепрыгнет через скобки $(\ \ldots\ )_{2\ 2}$, так как следующий символ совпадает с опорным, все просмотренное выражение отнесет к $\underline{E}1$.

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

2.I.5. Покажем теперь как ликвидируются лишние просмотры и переписи, указанные в пунктах 2.I.I и 2.I.2.

Обратимся к первому примеру . После набора $\underline{E}1$ содержимое $\underline{E}2$ 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 $\underline{K}$. В процессе синтаксического отождествления этот адрес можно хранить в рабочей ячейке. Точно так же определяется содержимое $\underline{E}4$ во

---

x/ Для наглядности пояснения в данном случае и иногда 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 $\underline{E}$ в тех случаях, когда оно не вызвано самой сутью работы алгоритма.

Что касается потерь времени, о которых указывалось в 2.I.2, то их также удается устранить за счёт адреса связи у символов $\underline{K}$. Для этого в РЕФАЛ-интерпретаторе создан специальный алгоритм связи символов $\underline{K}$ 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 $\underline{K}$. 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 $\underline{K}$ и $\underline{\cdot}$)

$$\underline{K}_1 \quad \underline{K}_2 \quad \underline{K}_3 \quad \underline{\cdot} \quad \underline{K}_4 \quad \underline{\cdot}\,\underline{\cdot}\,\underline{\cdot}$$

Интерпретатор с помощью адресов связи $A2$ организует все символы $\underline{K}$ в список $\underline{K}_3\,\underline{K}_4\,\underline{K}_2\,\underline{K}_1$ так что слева всегда находится ведущий символ конкретизации, следом за ним следует тот символ $\underline{K}$, который станет ведущим, когда будет полностью выполнена предыдущая конкретизация и т.д. 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 K_5, стоит выражение:

$$K_5\ K_6\ \ldots\ K_7\ K_8\ \ldots\ \ldots$$

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

$$K_6\ K_8\ K_7\ K_5\ K_4\ K_2\ K_1$$

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 символов K просматриваются только управляющие и значащие символы, число которых, естественно, не может быть велико.

Например, пусть имеется следующее предложение:

$$\S\ K\ \Psi\ S1\ W2\ E3 \sim K\alpha\ S1\ K\beta\ W2\ K\gamma\ E3\ \ldots$$

и пусть конкретизируемое выражение есть

$$K\Psi\alpha(\ldots)\ \underbrace{\ldots\ldots}_{100}\ \underbrace{\ldots}_{1000}$$

Для связи символов K, стоящих в правой части будут просматриваться только символы $K\alpha\ K\beta\ K\gamma$, а содержимое $S1$, равное α, $W2$, содержащее 100 символов, и $E3$,

содержащее 1000 символов, просматриваться не будут.

После того, как в списке не останется ни одного символа $\underline{K}$, 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 составляющих его подвыражений.

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

Например, скобочная структура

$$\alpha(\underline{E}1 + \underline{E}2) + \underline{E}3$$

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. 1.1. уже говорилось о том, что в примерах 1 и 2 для набора содержимых $\underline{E}2$ и $\underline{E}4$ нет необходимости просматривать содержимое поле зрения. Что касается переменной $\underline{E}2$ в примере 1, то осуществить такой алгоритм весьма просто. В примере 2 переменная $\underline{E}4$ 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 I.

Если на нулевом уровне скобочной структуры нет ни одной переменной $\underline{E}$ либо она всего одна, то такая скобочная структура называется закрытой. И переменную $\underline{E}$ в такой структуре (если она присутствует) мы также будем называть закрытой. Если таких переменных несколько, то скобочная структура называется открытой. И все переменные $\underline{E}$, кроме последней (правой при отождествлении слева направо и левой - справа налево) мы будем называть открытыми. Последняя переменная $\underline{E}$ называется закрытой.

В примере I левая часть является открытой скобочной структурой (напомним, что символ $\underline{K}$ 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:

$$\S\ \underline{K}\varphi\underline{E}1\ (\ w\ z^{n}+\ (\underline{E}3)_{2}\ \underline{E}4\ )_{1}\ \underline{E}5\ (\ a+b\ )_{3}\ \underline{E}6\sim\ldots \quad (\text{пр.}3)$$

В данном примере самая внешняя скобочная структура - открытая, $(\ldots)_{1}$ - закрытая, $(\ldots)_{2}$ - закрытая, $(\ldots)_{3}$ - закрытая. Открытыми переменными являются $\underline{E}1$ $\underline{E}5$, закрытыми - $\underline{E}3$, $\underline{E}4$, $\underline{E}6$.

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

закрывающей соответствующую скобочную структуру, меняет направление отождествления на противоположное и после того, как в левой части предложения вновь дойдет до закрытой переменной $\underline{E}$, оставшуюся часть поля зрения относит к содержимому указанной переменной. Просмотра конкретизируемого выражения для набора открытой переменной $\underline{E}$ не избежать. Однако, при этом все встретившиеся скобочные структуры конкретизируемого выражения обходятся за один шаг с помощью адресов связи у парных скобок (см. п2I.4).

Приведем пример. Пусть имеется предложение

$$\S\ \underline{K}\ \Psi\ \underline{E}1 + \underline{E}2\ \underline{S}3\ \underline{W}4 \sim \underline{K}\ \Psi\ \underline{E}2\ \underline{W}4\ \underline{E}1 .$$

и конкретизируемое выражение

$$\underline{K}_1 (\Psi ABC + \underset{2}{(} \ldots \underset{2}{)} \underset{3}{(} \ldots \underset{3}{)} AD \underset{4}{(} \ldots \underset{4}{)} \underset{1}{)}$$

Отождествление данного выражения будет проводиться следующим образом.

Сначала при движении слева направо будет определено, что содержимым $\underline{E}1$ является ABC. После этого интерпретатор, определив, что $\underline{E}2$ является закрытой переменной, обратиться к правой заключительной скобке $)_1$ и начнет отождествление справа налево. К переменной $\underline{W}4$ отойдет содержимое скобок $\underset{4}{(} \ldots \underset{4}{)}$, переменная $\underline{S}3$ примет значение D, а оставшееся выражение $\underset{2}{(} \ldots \underset{2}{)} \underset{3}{(} \ldots \underset{3}{)} A$ без просмотра будет отнесено к содержимому $\underline{E}2$.

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 конкретизируемого 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 структур.

Отождествление открытых скобочных структур начинается после того, как будут отождествлены все закрытые скобочные структуры, не находящиеся внутри открытых. При этом в зависимости от направления отождествления в первую очередь отождествляется либо левая открытая скобочная структура (при прямом отождествлении), либо правая (при обратном отождествлении)

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

$$\S\ \underline{K}\langle МЕН\rangle\ \underline{E}1\ (\underline{E}2 + \underline{E}3)_1\ \underline{E}4\ (\underline{S}5\ (\underline{E}6 + (\underline{E}7)_4\ \underline{E}8)_3\ \underline{W}9)_2 \sim$$

И пусть конкретизируемое выражение есть

$$\underline{K}\langle МЕН\rangle\ A\beta\ (A+B+A)_1\ \gamma\ (\ )_2\ (\alpha\ (A-B+(AB)_4)_3\ (A)_2 \dot{-}$$

Порядок 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 E_1 примет значение A, а переменная E_4 – B. При обходе скобочных структур мы сопоставили скобкам левой части предложения скобки конкретизируемого выражения, и этот факт в примере отмечен одинаковыми индексами у соответствующих скобок.

Из неотождествленных скобочных структур (см. левую часть предложения) $(_1 \ldots)_1$ является открытой, а $(_4 \ldots)_4$ – закрытой. Первой будет отождествляться $(_2 \ldots)_2$. При этом E_5 примет значение α, а E_9 – (A). Скобочная структура $(_3 \ldots)_3$ будет пропущена, однако ей сопоставится скобочная структура $(_5 \ldots)_5$ конкретизируемого выражения.

Теперь из неотождествленных остались только одни открытые скобочные структуры: $(_1 \ldots)_1$ и $(_3 \ldots)_3$. Первой будет отождествляться скобочная структура $(_1 \ldots)_1$. В результате E_1 примет значение A, а E_2 – значение $B+A$. После отождествления данной скобочной структуры интерпретатор будет искать следующую ближайшую слева открытую скобочную структуру. Ей окажется скобочная структура $(_3 \ldots)_3$. После ее отождествления E_6 примет значение $A-B$, а E_8 окажется пустым. При этом будет пропущена скобочная структура $(_4 \ldots)_4$. Она будет отождествляться в последнюю очередь, и переменная E_7 примет значение AB.

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 $\underline{E}$. А каждое удлинение, как правило, вызывает дополнительные просмотры поля зрения. Закрытые же скобочные структуры отождествляются без просмотра.

Приоритет, отдаваемый при отождествлении закрытым скобочным структурам перед открытыми, открывает дополнительные возможности избежать многократного удлинения. Это происходит в тех случаях, когда одна и та же переменная типа $\underline{E}$ 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 отождествлении открытая переменная $\underline{E}$ набирается за один просмотр путем сравнения с последовательностью символов, относящихся к идентичной закрытой переменной $\underline{E}$.

Например, пусть предложение, левая часть которого имеет вид

$$\underline{K}\langle\langle\rho\rangle\rangle\,\underline{E}1+(\underline{E}2\,\alpha\,\underline{E}3)\,\underline{E}'(\_2)\sim$$

является подходящим для конкретизируемого выражения

$$\underline{K}\langle\langle\rho\rangle\rangle(\,)+(\alpha\alpha\beta\alpha(\,)\delta)+A(\alpha\alpha\beta)\_$$

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

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

В случае неудачного отождествления какой-либо скобочной структуры отождествление всего предложения считается неуспешным, если только данная структура не находится внутри какой-либо открытой скобочной структуры. В последнем случае управление передается на удлинение соответствующей открытой переменной $\underline{E}$ [3].

При записи левой части предложения в память машины происходит анализ скобочных структур и разделение на типы переменных $\underline{E}$. 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 вышеизложенными принципами. Например, левая часть предложения в примере будет записана в общую память интерпретатора в следующем виде:

$$\underline{K}\langle cp\rangle\,\underline{E}1+(\,)\,\underline{E}4(\,),\ \underline{E}2,\ \underline{E}2\propto\underline{E}3\sim$$

В результате такой записи процессор в дальнейшем отождествляет скобочные структуры последовательно слева направо и при этом правила отождествления скобочных структур будут соблюдены.

При наличии в комментарии знака направления отождествления ($\underline{R}$) 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 конца ее содержимого. Соответствующая ячейка таблицы регистров выбирается с помощью идентификатора переменной. РЕФАЛ-интерпретатор обращается к таблице регистров, находит адрес начала и конца соответствующей переменной и при замене изменяет адреса связи начала и конца ($A1$ - начала и $A3$ - конца). Само тело переменной остается на прежних участках памяти. И весь процесс замены сводится фактически к перекоммутации адресов связи начала и конца используемых при этом переменных.

Например, пусть мы имеем предложение

$$\S\ \underline{K}\alpha\ \underline{E}1\beta\ \underline{E}2 \sim \underline{K}\gamma\ \underline{E}2\beta\ \underline{E}1 .$$

Покажем в таблице на какие символы указывают адреса A1 начала и A3 конца переменных $\underline{E1}$ и $\underline{E2}$ до и после замены:

	до замены	после замены
A1 начала E1	α	β
A3 конца E1	β	)
A1 начала E2	β	γ
A3 конца E2	)	β

Время, необходимое для замены, теперь совершенно не зависит от числа символов, входящих в переменные. 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

$$\S\ \underline{K}\ \Psi\ \underline{E1}\ \underline{S2}\ \underline{W3} \sim \underline{K}\ \alpha\ \underline{E1}\ \underline{S2}\ \underline{E1}\ \underline{W3}\ \underline{E1}\ .$$

переменная $\underline{E1}$ в правой части встречается трижды, в то время как левой части она встречается только один раз. В этом случае $\underline{E1}$, 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 [3].

Определение 2:

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

Например:

Предложение в примере (I) имеет детерминатив, состоящий из одного символа α. Предложение

$$\S\,\underline{K}\,\varphi\alpha(\underline{E}1)-\underline{E}2\sim\beta(\underline{E}1)+\underline{K}\,\psi\underline{E}2.$$

имеет детерминатив, состоящий из трех символов $\varphi\alpha($.
Предложение $\S\,\underline{K}\,\underline{W}1\,\&\,\underline{W}2\sim\underline{W}2$ имеет пустой детерминатив.

Введение понятия детерминатива и анализ символов с которых начинается конкретизируемое выражение, дает возможность избежать просмотра тех предложений, которые заведомо являются неприменимыми. 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, детерминатив 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):

§ 3.I $\underline{K}\,\alpha\alpha\,\underline{W}1\,\sim$

§ 3.2 $\underline{K}\,\alpha\gamma\,\underline{E}1(\underline{E}2)\underline{E}3\,\sim$ (пр 4)

§ 3.3 $\underline{K}\,\alpha\beta\,\underline{S}1\,\underline{W}2\,\sim$

§ 3.4 $\underline{K}\,\alpha\,\underline{E}1\,\sim$

и пусть конкретизируемым выражением является

$$\underline{K}\,\alpha\alpha\,(\,)\,\alpha\beta\,.$$

Первым будет отождествляться предложение 3.I. Отождествление будет неудачным. После этого бессмысленно, конечно, пытаться 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.I 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.I, 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 пока введем несколько определений, необходимых для формального описания такого алгоритма. Заметим только, что предложение вводится в память машины в порядке, обратном направлению прос-

мотра 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 мы имеем последовательность предложений: $A_1, A_2, \ldots, A_n$. Назовем предложение A_i более общим, чем A_j, а предложение A_j менее общим, чем A_i, если выполняется следующее равенство:

$$Det(A_j) = Det(A_i)C \ ,$$

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 последовательность предложений (в примере в левых частях указываются только детерминативы)

§ 1 $\underline{K}\ a \ldots \sim$
§ 2 $\underline{K}\ a\,b \ldots \sim$
§ 3 $\underline{K}\ a\,a \ldots \sim$
§ 4 $\underline{K}\ a\,b\,c \ldots \sim$
§ 5 $\underline{K}\ a \ldots \sim$ (пр. 5)
§ 6 $\underline{K}\ a\,a\,a \ldots \sim$
§ 7 $\underline{K}\ a\,b\,c \ldots \sim$
§ 8 $\underline{K}\ a\,a \ldots \sim$
§ 9 $\underline{K}\ a\,a \ldots \sim$
§ 10 $\underline{K}\ a\,b\,c \ldots \sim$
§ 11 $\underline{K}\ a\,b\,d \ldots \sim$

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 I,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 предложениями являются предложения 3,8 и 9, т.к. они имеют один и тот же детерминатив (*аа*).

Определение 4.

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

В примере (5) мы имеем три монотонных последовательности: первая включает предложения I,2,3, 4; вторая - 5,6,7; третья - 8,9,I0,II. В самом деле ни одно из следующих предложений в последовательности предложений I,2,3,4 не является более общим, чем предыдущее. Предложение же 5 является более общим, чем предыдущее 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 узлы 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 иерархии), второй символ (б) - в узле предложения 2 (на втором уровне иерархии) и только третий символ будет располагаться в узле, соответствующем предложению 4 (на третьем уровне иерархии).

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.

Узлы 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 I,2,3,4. Покажем такую

иерархию на рисунке.

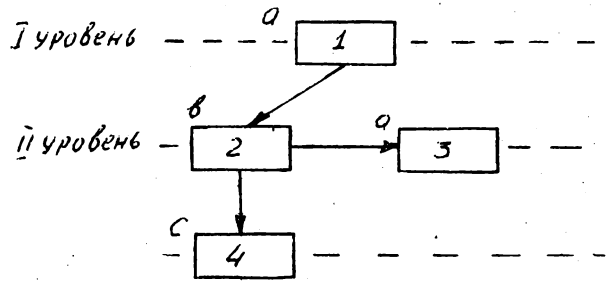


Рис 3

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

Узел, расположенный на I уровне иерархии, называется вершиной иерархии. Его адрес указывается в специальной таблице, так называемой таблице детерминативов (TD) .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 иерархии, 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 TD и определяем узел, соответствующий вершине иерархии. А затем ,сравнивая по порядку следующие символы

конкретизируемого выражения с символами, хранящимися 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.

ЦПП 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 отыскании 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.I. И поэтому ЦПП, имеющая началом, скажем, предложение 3.4, следующим элементом будет иметь предложение 3.I, так как только оно является более общим, чем предложение 3.4.

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

Например, если началом ЦПП внутри иерархии, изображенной на рисунке стр. 25 , является предложение 3, то ЦПП от этого предложения будет указывать сразу на предложение I, минуя предложение 2.

В целях экономии памяти для равнообщих предложений заводится только один узел - для самого последнего из них. Все остальные связываются в ЦПП.

Связь предложений в ЦПП осуществляется с помощью ячеек ЦПП, 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 вершины последней из них. В ячейке ЦПП, соответствующей такой вершине, указывается адрес вершины той иерархии, нарушением монотонности которой явилось данное предложение.

Таким образом, в общем случае, все предложения, детерминатив которых начинается с одного и того же символа, могут быть объединены в целую серию иерархий. Поиск начала ЦПП 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 ЦПП, происходит поиск подходящего предложения внутри соответствующей иерархии. При каждом неудачном отождествлении движение происходит по ЦПП. Если внутри данной иерархии подходящее предложение не было найдено, движение по ЦПП временно приостанавливается и происходит

обращение к предыдущей иерархии. В ней так же, как и в последней иерархии, согласно конкретизируемому выражению ищется начало ЦПП внутри данной иерархии, которое фактически является продолжением ЦПП, относящейся к предыдущей иерархии. После этого движение вновь осуществляется по ЦПП 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 ЦПП также с помощью адресов, расположенных в ячейках ЦПП. 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 диктуется требованием абстрактной РЕФАЛ-машины (опять же с учетом обратного ввода предложений).

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

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

Узел второго уровня иерархии должен содержать:

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

3) 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 :

1) код, соответствующий третьему символу детерминатива;

2) адрес начала левой части самого последнего предложения из числа равнообщих данному внутри данной иерархии;

3) адрес узла предложения, детерминатив которого отличается от детерминатива данного предложения только последним символом (адрес следующего узла данного уровня).

Ячейка ЦПП должна содержать признаковую часть $\mathcal{A}$ и адресное поле.

Если $\mathcal{A}=0$, то адресное поле содержит адрес, указывающий начало левой части того предложения, которое необходимо отождествлять после неудачного отождествления данного предложения внутри данной иерархии. С помощью этих адресов происходит движение по цепочке просматриваемых предложений.

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

Объединение предложений в иерархии и в ЦПП позволяет существенным образом сократить время нахождения подходящего предложения для данного конкретизируемого выражения. 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 левое конкретизируемое выражение подвигается к СПП. И если ни одна из переменных, находящихся 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 поднятого к СПП массива содержимое такой переменной выкидывается. Это достигается тем, что символ, предшествующий содержимому такой переменной, и символ следующий за ним, связываются между собой с помощью адресов связи. Такая процедура применяется к каждой переменной, используемой при замене левой части

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

Например, пусть имеется предложение

$$\S\ \underline{K}\alpha\ \underline{W}1 + \underline{W}2\beta \sim \underline{K}\Psi\ \underline{W}2 \underline{.}$$

и конкретизируемое выражение

$$\underline{K}\alpha\ (_1 \ldots)_1 + (_2 \ldots)_2\ \beta \underline{.}$$

Так как при замене используется только переменная $\underline{W}2$, ее содержимое выкидывается из содержимого подшитого к СПП конкретизируемого выражения. Для этого символы $+$ и β связываются между собой. И к СПП будет подшито выражение

$$\underline{K}\alpha\ (_1 \ldots)_1 + \beta \underline{.}$$

Наличие программы "чистильщик" позволяет полностью использовать память машины.

3. РЕФАЛ-интерпретатор для машины БЭСМ-6

В настоящей главе описывается общая схема и основные блоки РЕФАЛ-интерпретатора [5].

Работа блоков иллюстрируется с помощью блок-схем. Для описания блок-схем используется специальный полуформализованный 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 [7].

На рис. 4 изображена общая схема интерпретатора. Интерпретатор состоит из четырех 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 блоков 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 разрядным кодом. Содержимое самой цепочки


вход
БСК
БПИ
Препроцессор
Процессор
выход

рис. 4

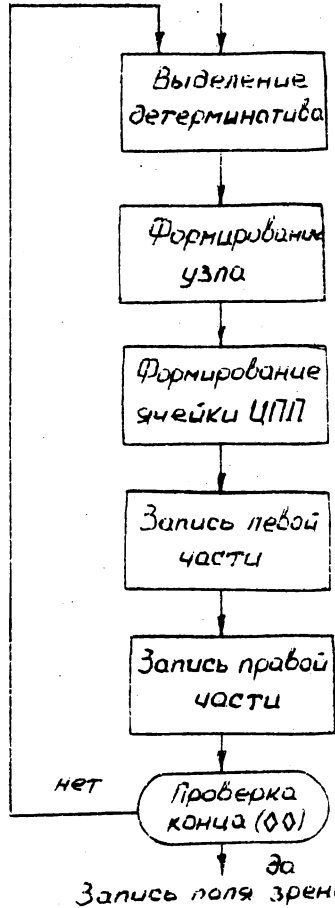

Выделение детерминатива
Формирование узла
Формирование ячейки ЦПП
Запись левой части
Запись правой части
нет
Проверка конца (00)
да
Запись поля зрения

рис. 5

запоминается в таблице *T1* и в дальнейшем используется при печати промежуточных или окончательных результатов. Таблица ограничивается 2400_8 ячейками, а это приводит к тому, что максимальное число символов, расположенных в кавычках, не должно превышать I2. Если после записи всех символов таблица *T1* использовалась не полностью, оставшаяся часть интерпретатором относится к СШ.

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

Управляющий символ	Код
≤	700
≡	600
≥	500
≠	400
)	300
(	200
←	I00

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

При перекодировке все буквенные идентификаторы переменных заменяются цифрами.

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

При проверке предложений общей памяти анализируется каждое предложение отдельно. При обнаружении ошибки БСК 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 БСК может обнаружить максимум одну ошибку. Остальные ошибки можно обнаружить, анализируя визуально напечатанный текст предложения, либо при повторном контроле с помощью БСК.

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

1) отсутствие § в начале предложения;
2) отсутствие символа $\underline{K}$ в начале левой части;
3) отсутствие знака $\sim$ разделяющего левую и правую часть предложения;
4) подчеркивание символа, отличного от S, W, E и $\cdot$;
5) 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;
9) 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 $\underline{\cdot}$, относящейся к соответствующему символу $\underline{<}$, или наоборот, наличие лишней $\underline{\cdot}$, не сбалансированной соответствующим символом $\underline{<}$.

При синтаксическом контроле поля зрения БСК обнаруживает несбалансированные скобки, а также проверяется соответствие между символами $\underline{<}$ и $\underline{\cdot}$.

3.I. Язык 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 ограничителей металингвистических переменных в нормальной бэкусовской форме используются угловые скобки).

I.I. Адрес

I.I.I. Синтаксис

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

<цифра> ::= 0|1|2|3|4|5|6|7|8|9

<целое десятичное> ::= <цифра> | <целое десятичное> <цифра>

<идентификатор> ::= <буква> | <идентификатор> <буква> | <идентификатор> <цифра>

<адрес> :: = <целое десятичное> | <идентификатор> |
<адрес> + <адрес> | <адрес> - <адрес>

I.I.2. Примеры

α A25 15-4
γ δ7 A1+α-12
141 αβγ

I.I.3. Семантика

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

I.2. Переменная

I.2.I. Синтаксис

<переменная> ::= <простая переменная> |
<косвенная переменная> |
<содержимое таблицы> | <адрес>

<простая переменная> ::= <идентификатор> |
<адрес>
$<адрес> | '<самоопределенная величина>'

<косвенная переменная> ::= <идентификатор> [<выражение>]

<содержимое таблицы> ::= <идентификатор> [T <целое десятичное> (<выражение>)]

<самоопределенная величина> ::= любая последовательность знаков

<выражение> ::= <переменная> |
<переменная> + <целое десятичное> |
<переменная> - <целое десятичное>

I.2.2. Примеры

Простая переменная $И4$

$М1_{А1+2}$

S_{α}

$'('$

Косвенная переменная $А1[В3_{\alpha}]$

$И2[А2[\gamma 1_{\alpha-1}]-3]$

Содержимое таблицы $М1[Т5(Sp_i)]$

$\beta[Т1(\alpha[Т2(А3_{\gamma})])]$

I.2.3. Семантика.

Простая 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 α выделяемая часть АI расположена в восьми младших разрядах, для обращения к этой части достаточно написать $А1_{\alpha}$. Для полной формализации записи алгоритма в языке необходимо иметь средства, позволяющие формально задавать идентификацию различных частей ячейки. Однако, такого средства мы не даем, а в дальнейшем, в необходимых случаях, будем содержательно описывать такую идентификацию. Символ S при описании простой переменной имеет служебное значение. Он применяется в тех случаях, когда несущественно с точки зрения описываемого алгоритма к какой части ячейки в данный момент идет обращение.

Задание простой переменной в виде самоопределенной величины 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 $B3$ ячейки, имеющей адрес α, хранится адрес β, то справедливо следующее равенство:

$$A1[B3\alpha] = A1\beta$$

С помощью переменной < содержимое таблицы > можно обращаться к соответствующей ячейке таблицы, начало которой обозначается с помощью служебного символа T 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.

I.3. Операторы

I.3.I. Синтаксис

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

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

< операция сравнения > ::= < | ≤ | = | ≥ | > | ≠

I.3.2. Примеры

$$A1\alpha := B2\beta$$
$$M1[T2(S\alpha)] := M2[A1\gamma + 1]$$
$$M4\alpha := '('$$
$$A1[C1_{СТРК}] := 'K'$$

I.3.3. Семантика.

Оператор присваивания имеет свой обычный смысл. Операторы сравнения аналогичны командам машины, осуществляющим условную

передачу управления, и предназначены для разветвления программы.

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

3.2. Препроцессор

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

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 памяти. Если после данного предложения стоят подряд два параграфа (§ §), запись предложений общей памяти заканчивается, и управление передается на запись поля зрения.

Рассмотрим более подробно работу каждого из этих блоков.

3.2.1. Выделение детерминатива (ВД)

Блок ВД выделяет детерминатив предложения в соответствии с его формальным определением (см.п.4 гл.II). Практически это осуществляется просмотром первых трех символов, стоящих в левой части предложения сразу за символом $\leftarrow$. Появление среди них какой-нибудь свободной переменной ограничивает 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 переписывается на магнитный барабан (МБ). Препроцессор переписывает ее с МБ квантами по 256 слов, для чего отводится соответствующий буферный массив ячеек МОЗУ. Обозначим его через α .

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

Если после символа $\underline{K}$ идет $\sim$, это значит, что детерминатив пустой. Если не $\sim$, 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 PI, берется следующий символ и производится аналогичная проверка.

Из данного блока существует четыре выхода: пустой детерминатив, детерминатив, состоящий из одного символа (запоминается в PI), детерминатив, состоящий из двух символов (запоминается в PI и P2) и детерминатив, состоящий из трех символов (запоминается в PI, P2 и P3).

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

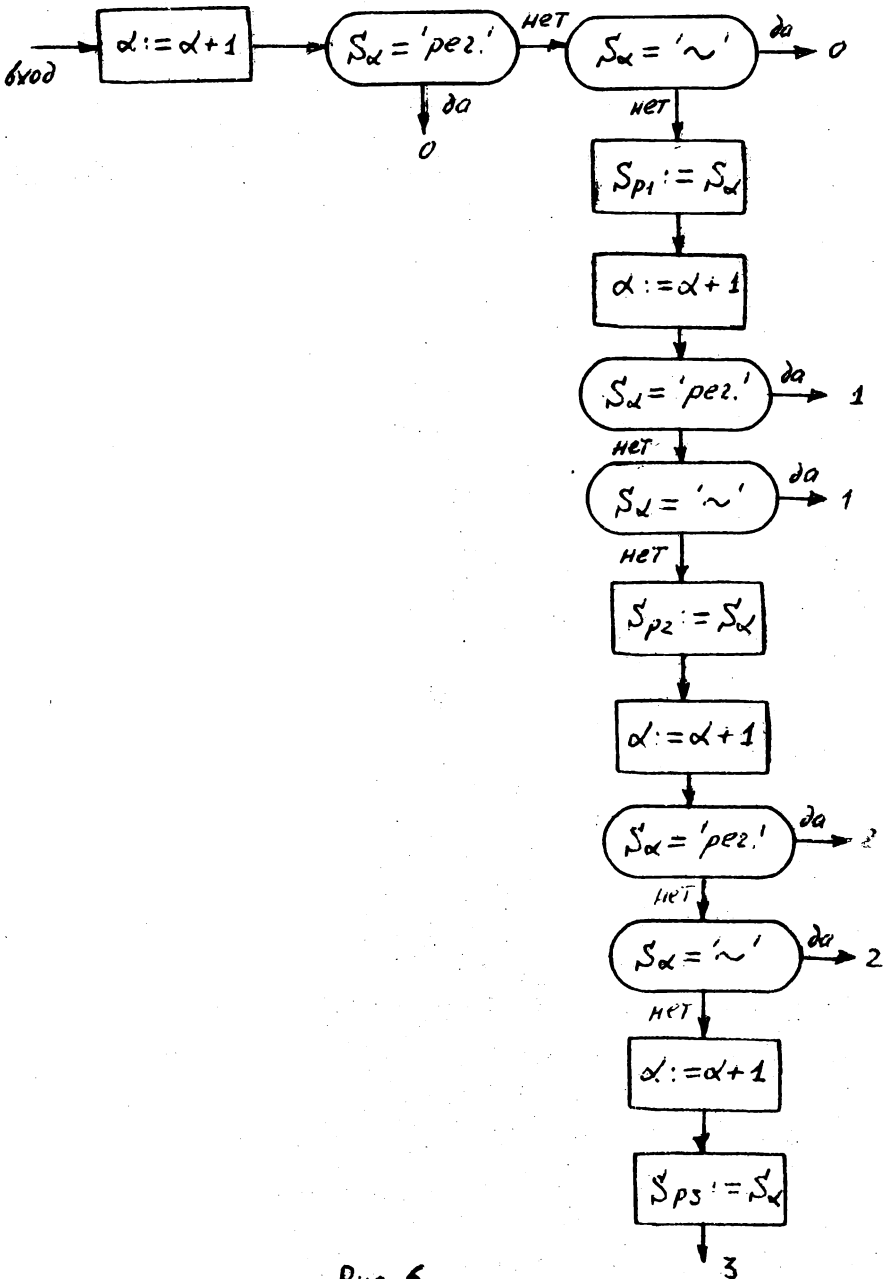

вход
α := α + 1
Sα = 'рез.'
нет
да
0
Sα = '~'
да
0
нет
Sр1 := Sα
α := α + 1
Sα = 'рез.'
да
1
нет
Sα = '~'
да
1
нет
Sр2 := Sα
α := α + 1
Sα = 'рез.'
да
нет
Sα = '~'
да
2
нет
α := α + 1
Sр3 := Sα
3

Рис. 6

левая часть предложения имеет вид

§ 2.I $\underline{K}(((\underline{E}1)+\underline{E}2)+\underline{W}3)\sim$

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. Включение в цепочку просматриваемых предложений (ЦПП)

Исходной информацией для данного блока является: начало таблицы детерминативов (обозначим его через Т2) и детерминатив данного предложения, первый символ которого находится в рабочей ячейке РI, а второй - в Р2.

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 связывается с узлами второго уровня иерархии, и ячейка ЦПП, в которой указывается адрес ближайшего более общего предложения, относящегося к данной иерархии. Узлу второго уровня иерархии отводятся две ячейки памяти машины БЭСМ-6.

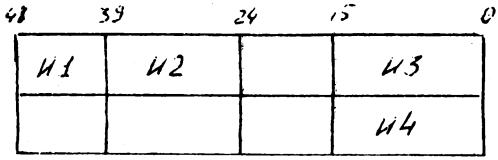


Рис. 7

где:

И1 - код, соответствующий второму символу детерминатива;

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

И3 - адрес узла менее общего предложения;

И4 - адрес следующего узла второго уровня иерархии.

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

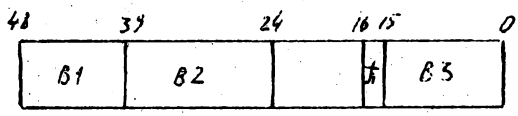


Рис. 8

где:

В3 содержит адрес связи предложений в ЦПП (см.п.4 гл.II); В2 содержит адрес начала правой части предложения, а поскольку предложения общей памяти хранятся в упакованном виде, то в В1 находится константа сдвига, указывающая число разрядов, на которое нужно сдвинуть первый символ правой части для того, чтобы 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.

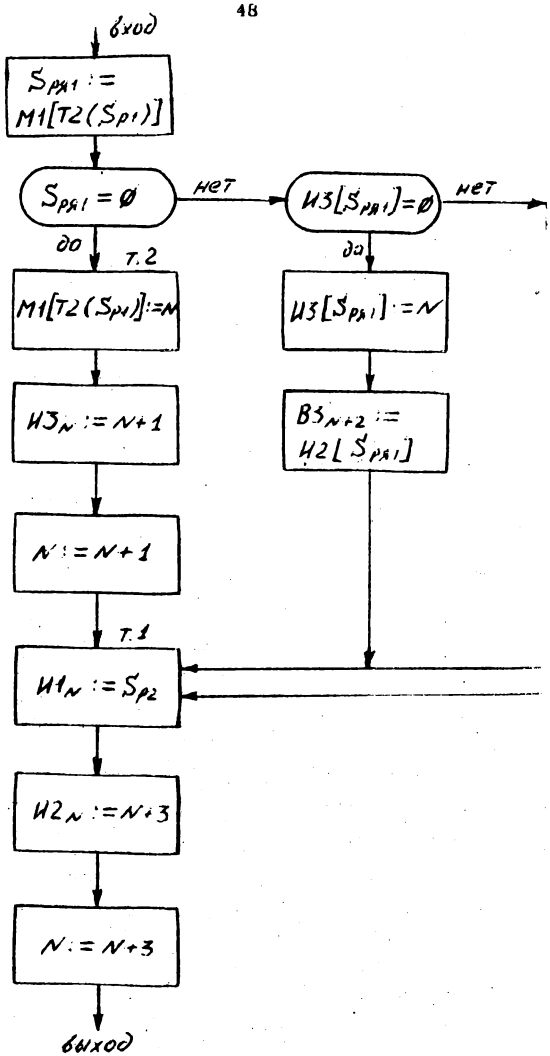
На рис. 9 изображена блок-схема алгоритма включения очередного предложения в ЦПП.

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

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

МІ будет пустым в том случае, когда ни одного более общего предложения не было записано в память машины. В этом случае узел данного предложения становится вершиной новой иерархии. Для этого в Т2 помещается адрес первой ячейки данного узла $M1[T2(S_{p1})] := N$. В дальнейшем заполняются И1, И2, И3 и И4 первой и второй ячейки узла. В ячейке ЦПП $\neq$ и И3 остаются равными 0. Для процессора это означает, что при движении по ЦПП после неудачного отождествления следует обратиться к ЦПП 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 И3 данной ячейки равно 0, это значит, что на втором уровне иерархии до этого не было записано ни одного предложения. В таком случае узел, находящийся в вершине иерархии,


вход
$S_{рд1} :=$
$M1[T2(S_{р1})]$
$S_{рд1} = \emptyset$
нет
$И3[S_{рд1}] = \emptyset$
нет
да
т.2
да
$M1[T2(S_{р1})] := N$
$И3[S_{рд1}] := N$
$И3_N := N+1$
$В3_{N+2} :=$
$И2[S_{рд1}]$
$N := N+1$
т.1
$И1_N := S_{р2}$
$И2_N := N+3$
$N := N+3$
выход

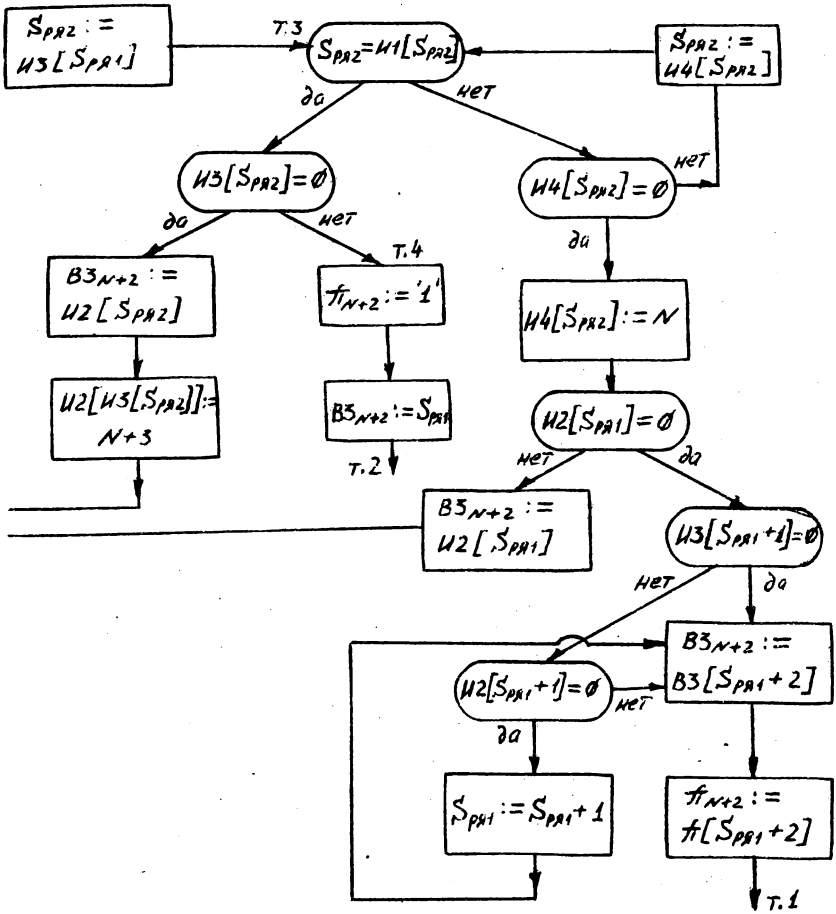

$S_{РЯ2} := И3[S_{РЯ1}]$
т.3
$S_{РЯ2} = И1[S_{РЯ2}]$
$S_{РЯ2} := И4[S_{РЯ2}]$
да
нет
$И3[S_{РЯ2}] = \emptyset$
$И4[S_{РЯ2}] = \emptyset$
нет
да
да
нет
$ВЗ_{N+2} := И2[S_{РЯ2}]$
т.4
$ff_{N+2} := '1'$
$И4[S_{РЯ2}] := N$
$И2[И3[S_{РЯ2}]] := N+3$
$ВЗ_{N+2} := S_{РЯ1}$
$И2[S_{РЯ1}] = \emptyset$
т.2
нет
да
$ВЗ_{N+2} := И2[S_{РЯ1}]$
$И3[S_{РЯ1}+1] = \emptyset$
нет
да
$ВЗ_{N+2} := ВЗ[S_{РЯ1}+2]$
$И2[S_{РЯ1}+1] = \emptyset$
нет
да
$S_{РЯ1} := S_{РЯ1}+1$
$ff_{N+2} := ff[S_{РЯ1}+2]$
т.1

Рис. 9

должен указывать на узел данного предложения $ИЗ[S_{рз1}] = N$
В ячейку ЦПП посылается адрес начала предложения, стоящего в вершине иерархии $ВЗ_{N+2} := И2[S_{рз1}]$. Остальные ячейки вновь образуемого узла формируются как и в предыдущем случае, для чего управление передается в τ 1 .

Если $ИЗ[S_{рз1}]$ 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 И1 каждого узла.

При удачном поиске проверяется, не содержит ли найденный узел адрес-ссылку на следующий уровень иерархии. Если нет, т.е. $ИЗ[S_{рз2}] \neq 0$, 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 начала левой части последнего из них $И2[ИЗ[S_{рз2}]] := N+3$
Ячейка ЦПП последнего предложения должна содержать адрес начала левой части предыдущего равнообщего предложения
$ВЗ_{N+2} := И2[S_{рз2}]$. Например, если мы имеем последовательность равнообщих предложений

§ 2.2.1 $\underline{K}\ \alpha\ \beta\ \ldots\ \sim$
§ 2.2.2. $\underline{K}\ \alpha\ \beta\ \ldots\ \sim$
§ 2.2.3. $\underline{K}\ \alpha\ \beta\ \ldots\ \sim$
§ 2.2.4. $\underline{K}\ \alpha\ \beta\ \ldots\ \sim$

то по мере записи этих предложений в И2 узла, относящегося

к предложению 2.2.I будет находиться адрес начала левой части предложений: сначала 2.2.I, затем 2.2.2, 2.2.3 и наконец 2.2.4. Так что, хотя формально узел стоит перед предложением 2.2.I, 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 II образуется вершина новой иерархии. Для этого в ячейке ЦПП данного предложения (см. $\tau 4$ на блок-схеме) принимается значение I, а в ВЗ посылается адрес вершины той иерархии, нарушением монотонности которой явилось данное предложение $B5_{N+2} := S_{PA1}$ В дальнейшем управление передается в $\tau 2$ для формирования узла данного предложения.

Если при движении по цепочке узлов второго уровня не будет найден узел предложения, равнообщего данному, рано или поздно мы дойдем до узла, у которого $И4 = 0$. В этом случае образуется новый узел данного уровня иерархии. В И4 последнего найденного узла посылается адрес вновь образованного узла $И4[S_{PA2}] := N$. В дальнейшем проверяется, не стоит ли в вершине последней иерархии предложение, детерминатив которого состоит из двух символов. Если это так, т.е. $И2[S_{PA1}] = 0$ и $И3[S_{PA1}+1] = 0$, то в ячейку ЦПП записываемого предложения, посылается содержимое $\#$ и ВЗ вершины иерархии (см. $\tau 6$). Например, пусть мы имеем последовательность вводимых в память машины предложений

§ 2.2.5 $\underline{\kappa}\, a \ldots \sim$

§ 2.2.6 $\underline{\kappa}\, abc \ldots \sim$

§ 2.2.7 $\underline{\kappa}\, ab \ldots \sim$

§ 2.2.8 $\underline{\kappa}\, ad \ldots \sim$

и пусть в данный момент записывается предложение 2.2.8. В его ячейку ЦПП перенесется содержимое $\not\tau$ и 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 $\mu 5[S_{p\text{я}1}+1]$ 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 ЦПП заносится значение $\not\tau$ и ВЗ вершины иерархии. Например, пусть мы имеем последовательность предложений

§ 2.2.9 $\underline{\kappa}\, ab \ldots \sim$

§ 2.I.I0 $\underline{\kappa}\, ab \ldots \sim$

§ I.2.II $\underline{\kappa}\, abc \ldots \sim$

§ 2.2.I2 $\underline{\kappa}\, ad \ldots \sim$

и пусть предложение 2.2.9 находится в вершине последней иерар-

хии. В ячейку ЦПП предложения 2.2.I2 будет записана ссылка на вершину предыдущей иерархии, так как ни одно из предложений 2.2.9 + 2.2.II не может оказаться подходящим, если в качестве первого из проверяемых было выбрано предложение 2.2.I2

И наконец, когда $u2[S_{рз,1}]$ отлично от нуля, в этом случае в ячейку ЦПП записываемого предложения посылается адрес, указывающий начало левой части более общего предложения $BS_{N+2} := u2[S_{рз,1}]$. В дальнейшем управление передается в 7.1 (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 левой части предложения

Запись левой 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 памяти.

На рис. 11 изображена блок-схема алгоритма переписи левой части предложения в развернутом виде. Под массив ТЗ отводится I00 ячеек. Это приводит к тому, что в левой части предложения число символов не должно превосходить I00. Практически это не накладывает на язык существенных ограничений, т.к. практика

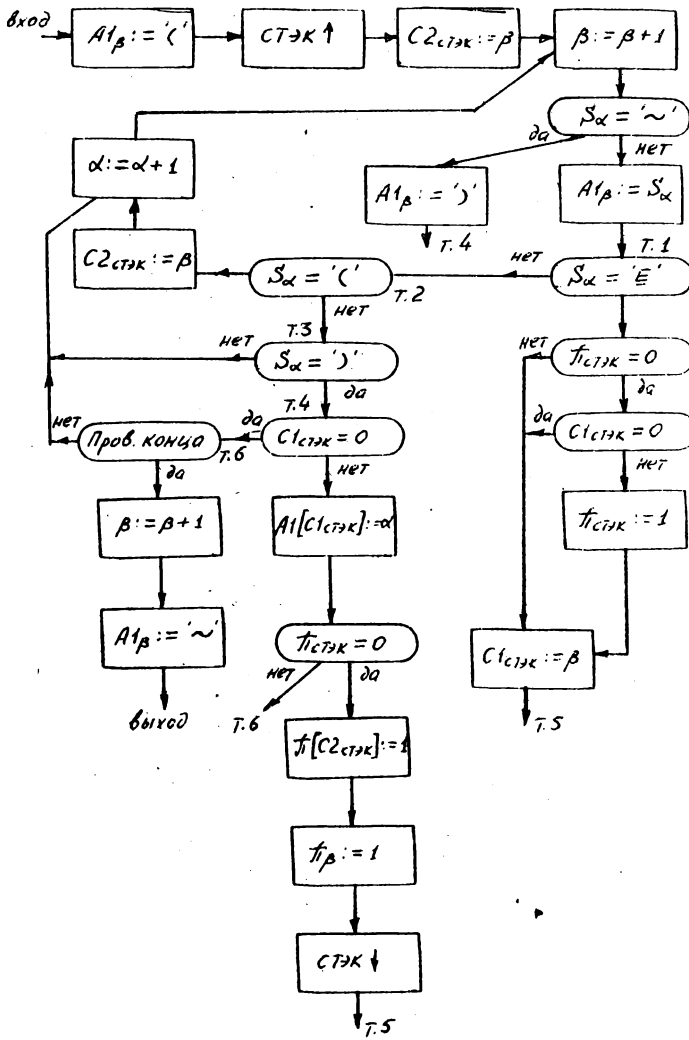

вход
A1β := '('
СТЭК ↑
C2стэк := β
β := β + 1
Sα = '~'
да
нет
α := α + 1
A1β := ')'
A1β := Sα
т. 4
т. 1
C2стэк := β
Sα = '('
нет
Sα = 'Е'
т. 2
нет
т. 3
нет
Sα = ')'
нет
fiстэк = 0
да
т. 4
да
нет
Пров. конец
да
C1стэк = 0
да
C1стэк = 0
т. 6
нет
нет
да
β := β + 1
A1[C1стэк] := α
fiстэк := 1
A1β := '~'
fiстэк = 0
C1стэк := β
нет
да
выход
т. 6
т. 5
fi[C2стэк] := 1
fiβ := 1
СТЭК ↓
т. 5

Рис. 11

использования РЕФАЛа [8, 9, 10, 11], 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 $\underline{E}$ в соответствии с определениями, данными в п.2 гл.П.

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

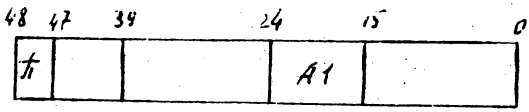


Рис. 10

A1 содержит код самого символа (разряды с 16 по 24), $\not{n}$ – признак для скобки ($\not{n}=0$ – закрытая скобочная структура, $\not{n}=1$ –открытая). Для открытой переменной $\underline{E}$ в A1 записывается код 600, для закрытой 700.

При анализе скобочных структур заводится стэк. На рис. 12 изображена ячейка такого стэка:

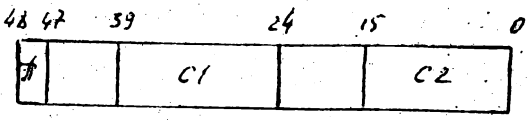


Рис. 12

C1 – адрес закрытой переменной $\underline{E}$ (если она есть в скобочной структуре),

C2 – адрес соответствующей левой скобки,

$\not{n}$ – признак, который равен 0 для закрытой скобочной структуры и 1 в случае открытой скобочной структуры.

Обратимся вновь к рис. 11 Текущий адрес массива ТЗ обозначается через β , переписываемого массива через α . Образование новой ячейки стэка обозначается СТЭК $\uparrow$, ликвидация последней строки – СТЭК $\downarrow$.

Перед входом в блок массив ТЗ и стэк очищаются, β принимает значение адреса начала массива ТЗ, α – адреса начала левой части предложения.

Работа блока заключается в посимвольной переписи содержимого массива α в массив ТЗ. При этом выделяются случаи появления $\underline{E}$, (или) (см.точки 1,2 и 3 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 π данной левой скобки.

При появлении $\underline{E}$ анализируется содержимое последней строки стэка.

При первом появлении $\underline{E}$ на нулевом уровне скобочной структуры значение π остается равным 1, 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 структуры. При этом происходит окончательный анализ данного уровня. Для открытых скобок π принимает значение 1, для закрытых 0. Закрытая переменная $\underline{E}$ принимает код 700. Для этого используется 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).

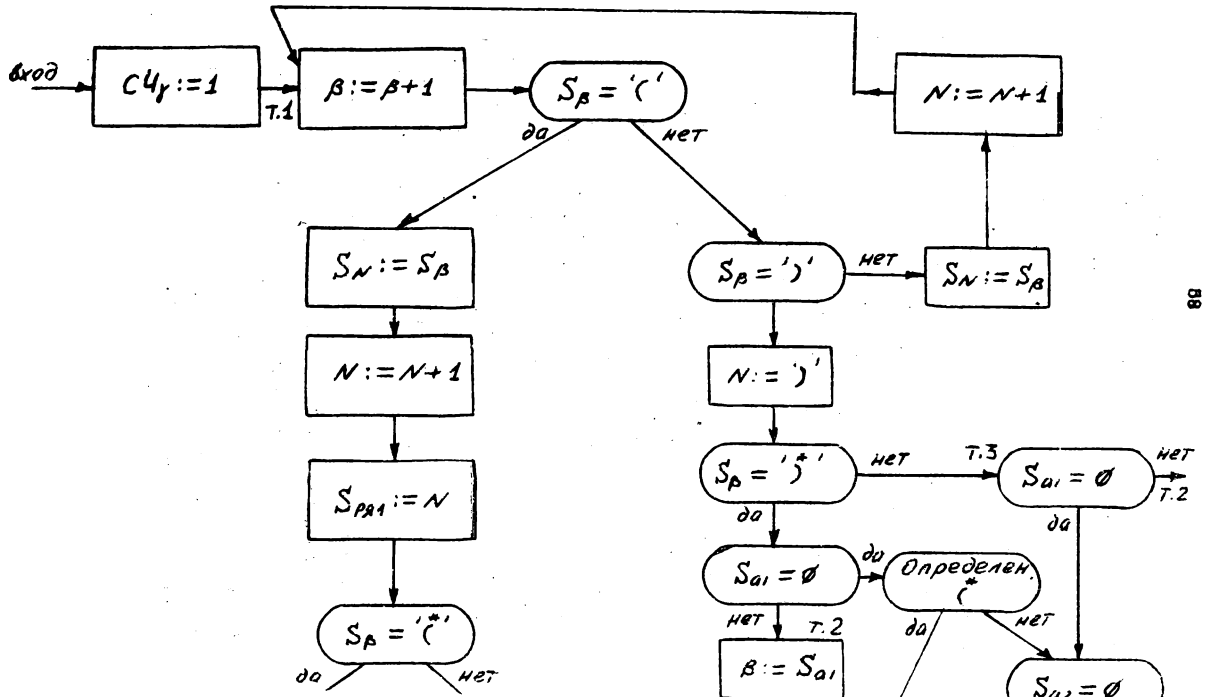
Если присутствует указатель направления синтаксического отождествления, то левая часть предложения записывается в массив ТЗ в обратном порядке. При этом левая скобка заменяется на правую, а правая - на левую; указатель переменной и ее идентификатор меняются местами.

Запись левой части предложения начинается сразу после ячейки ЦПП. Массив памяти машины, в который записывается предложение, мы как и раньше обозначаем через N . На блок-схеме не показывается плотная упаковка символов в ячейке, но подразумевается, что, например, $S_N = S_\beta$ означает включение подпрограммы, записывающей символ в первую свободную позицию ячейки памяти (напомним, что каждая ячейка имеет пять таких позиций).

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

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

Работа блока начинается с того, что заводится счетчик C_4 , которому вначале присваивается I. Этот счетчик необходим для того, чтобы каждой скобочной структуре присвоить соответствующий номер. Левая скобка самой внешней скобочной


вход
СЧ$_У$:= 1
т.1
β := β+1
$S_β$ = '('
да
нет
N := N+1
S_N := $S_β$
N := N+1
$S_{РД1}$:= N
$S_β$ = '(*'
да
нет
$S_β$ = ')'
нет
S_N := $S_β$
N := ')'
$S_β$ = ')*'
нет
т.3
S_{a1} = ∅
нет
т.2
да
да
S_{a1} = ∅
да
Определен. (*
нет
нет
т.2
да
β := S_{a1}
S_{a2} = ∅

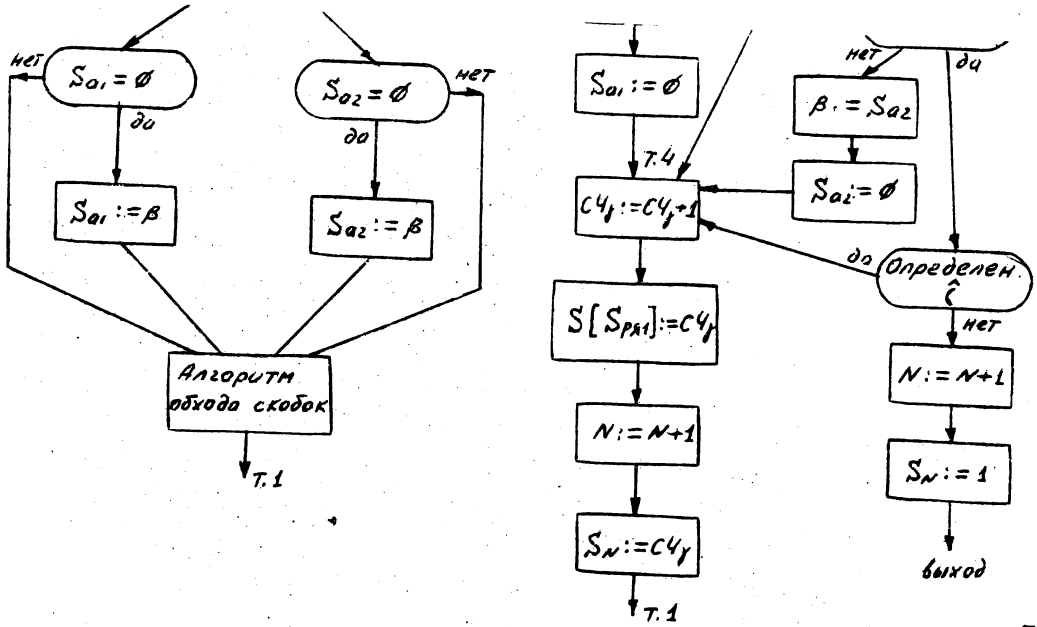

нет
$S_{a1} = \emptyset$
да
$S_{a1} := \beta$
$S_{a2} = \emptyset$
нет
да
$S_{a2} := \beta$
Алгоритм обхода скобок
Т. 1
$S_{a1} := \emptyset$
Т. 4
$СЧ_y := СЧ_y + 1$
$S[S_{PA1}] := СЧ_y$
$N := N + 1$
$S_N := СЧ_y$
Т. 1
нет
да
$\beta := S_{a2}$
$S_{a2} := \emptyset$
до
Определен. ĉ
нет
$N := N + 1$
$S_N := 1$
выход

Рис. 13

структуры в массив N не записывается, поэтому из массива ТЗ берется следующий символ $\beta := \beta + 1$. 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 $T1$.

Если очередной символ оказался левой скобкой, то она также записывается в массив N, после нее резервируется место для номера $N := N+1$. Полный адрес номера (адрес ячейки и номер позиции) запоминаются в рабочей ячейке РЯ1. После этого проверяется не является ли скобка закрытой $S_\beta = ')'$ (На блок-схеме закрытая скобка обозначается $)^*$, а открытая $\hat{)}$).

Если скобка оказалась закрытой, то проверяется содержимое рабочей ячейки $Q1$. В $Q1$ хранится адрес начала самой левой закрытой скобочной структуры. Если $S_{Q1} = 0$, это значит, что при движении по внешней скобочной структуре данная закрытая скобочная структура встретилась впервые. В этом случае в $Q1$ заносится адрес β левой скобки указанной скобочной структуры, а сама скобочная структура при этом пропускается. Это достигается включением алгоритма обхода скобок. Если $S_{Q1} \neq 0$, содержимое $Q1$ не меняется, а управление передается на обход встретившейся скобочной структуры.

В том случае, когда скобка оказалась открытой, проверяется содержимое рабочей ячейки $Q2$, которая предназначена для хранения адреса начала самой левой открытой скобочной структуры. Алгоритм проверки точно такой же, как и в случае закрытой скобочной структуры.

Если очередным символом массива ТЗ оказалась правая скобка,

то прежде всего пишется символ конца скобочной структуры . После этого анализируется сама скобка. Если скобка оказалась закрытой, то проверяется содержимое $a1$.

Если $S_{a1} \neq 0$, то β становится равным адресу, расположенному в $a1$, S_{a1} становится равным 0, счетчик $C4_y$ увеличивается на I и его содержимое записывается по полному адресу, расположенному в РЯП. После этого полный адрес N сдвигается на следующую позицию, куда и записывается последнее содержимое $C4_y$, управление передается в $\gamma 1$.

Если $S_{a1} = 0$ 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 скобочной 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 $a1$ (см. $\gamma 3$ блок-схемы).

Если $S_{a1} \neq 0$, то управление передается в $\gamma 2$. Если $S_{a1} = 0$ это значит, что ни внутри ни во вне данной скобочной структуры закрытых скобочных структур нет (в расчет не принимаются согласно п.2 гл.П закрытые скобочные структуры, находящиеся внутри открытых). Поэтому блок определения закрытых скобочных структур не включается, а проверяется содержимое ячейки $a2$.

Если $S_{a2} \neq 0$, то β принимает адрес начала соответствующей открытой скобочной структуры $\beta = S_{a2}$, $a2$ стано-

витоя равным 0 и управление передается в 7.4 . Если $Sa_2 = 0$, 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 T2 проверяется не является ли он ~ . Если да, это означает конец просмотра. До тех пор, пока такой символ не появился, выделяются 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 β как раз равен адресу начала скобочной структуры.Если левая скобка оказалась открытой, то такая скобочная структура пропускается, для чего включается алгоритм обхода скобок. Если появляется правая открытая скобка, это также означает конец просмотра, т.к. вне данной скобочной структуры уже не может быть закрытых скобочнных структур, которые бы не находились внутри открытых скобочных структур. А такие 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 включаетоя только тогда, когда в массиве T3 остались неперепи-саннными только открытые скобочные структуры, которые внутри себя в свою очередь могут содержать закрытые скобочные структуры. Алгоритм выделяет ближайшую левую открытую скобочную структуру. При этом встречающиеся закрытые скобочные структуры не пропускаются, а происходит вхождение внутрь их, т.к. внутри себя они могут содержать неперепиcанные открытые скобочные

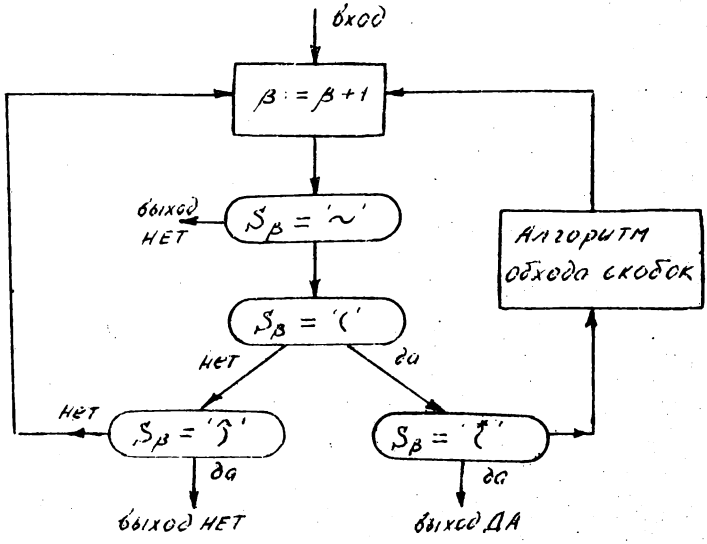

вход
β := β + 1
выход
НЕТ
S_β = '~'
Алгоритм обхода скобок
S_β = '('
нет
да
нет
S_β = ')'
S_β = '['
да
да
выход НЕТ
выход ДА

Рис 14

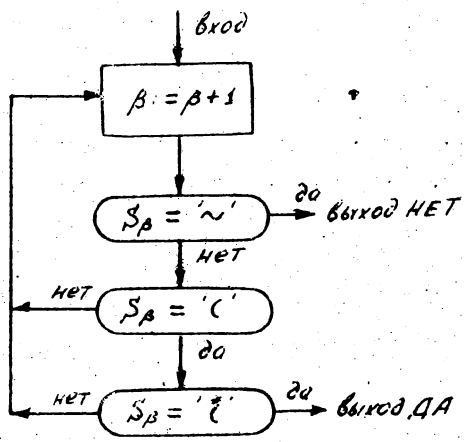

вход
β := β + 1
S_β = '~'
да
выход НЕТ
нет
нет
S_β = '('
да
нет
S_β = '['
да
выход ДА

Рис 15

структуры. Когда находится адрес открытой левой скобки, управление передается в 7.4 блок-схемы на рис. 15. Если встречается символ ~, это означает конец работы всего блока переписи левой части и управление передается на запись правой части предложения.

3.2.4. Запись правой части предложения

Правая часть предложения так же, как и левая часть, записывается 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 части. Мы вновь здесь обращаемся к массиву α . В данном блоке посимвольно переписывается содержимое массива α в массив ν . Когда встречается символ $\underline{K}$, после него записывается левая скобка. Перед . записывается 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 записывает информацию в память машины в соответствии с принципами, описанными в п.I гл.II. Каждому символу поля зрения отводится отдельная ячейка рпамяти машины БЭСМ-6.

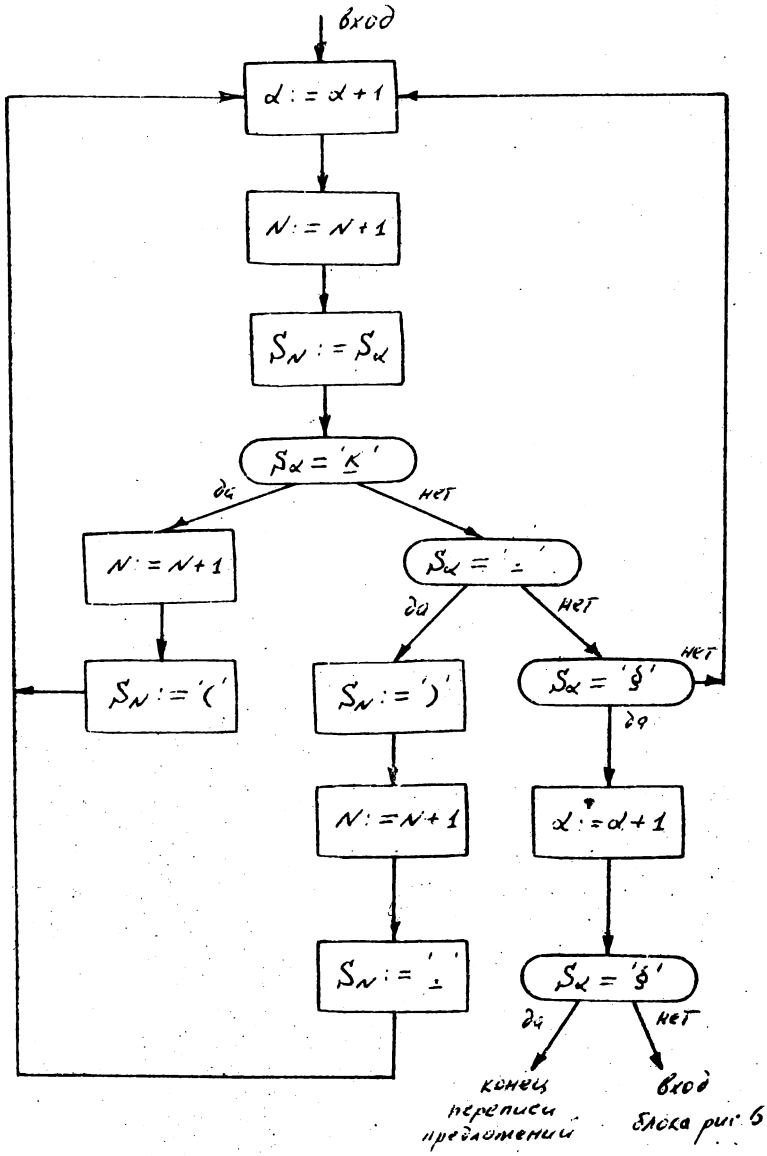

вход
α := α + 1
N := N + 1
S_N := S_α
S_α = '<'
да
нет
N := N + 1
S_α = '='
да
нет
нет
S_N := '('
S_N := ')'
S_α = '$'
да
N := N + 1
α := α + 1
S_N := ''
S_α = '$'
да
нет
конец переписи предложений
вход блока рис 6

Рис 16

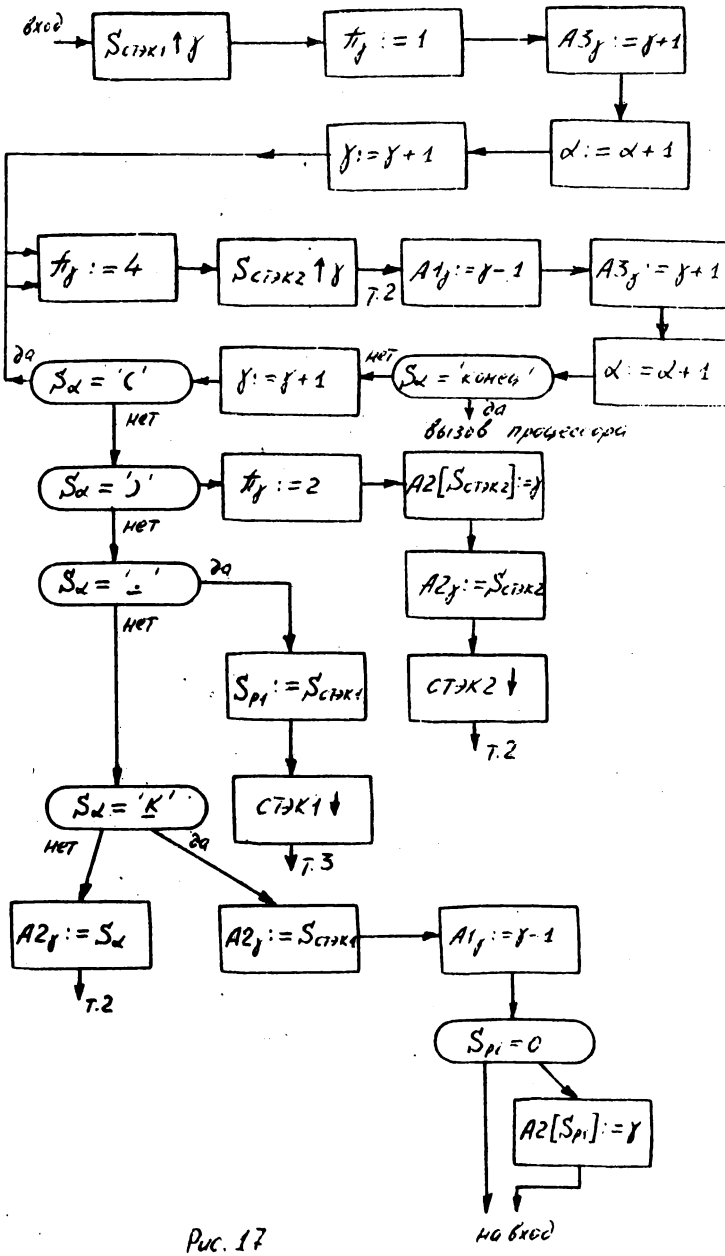

вход
$S_{стэк1}\uparrow\gamma$
$tt_\gamma := 1$
$A3_\gamma := \gamma+1$
$\alpha := \alpha+1$
$\gamma := \gamma+1$
$tt_\gamma := 4$
$S_{стэк2}\uparrow\gamma$
Т.2
$A1_\gamma := \gamma-1$
$A3_\gamma := \gamma+1$
$\alpha := \alpha+1$
$S_\alpha = 'конец'$
да
вызов процессора
нет
$\gamma := \gamma+1$
да
$S_\alpha = 'C'$
нет
$S_\alpha = 'D'$
$tt_\gamma := 2$
$A2[S_{стэк2}] := \gamma$
$A2_\gamma := S_{стэк2}$
СТЭК2 ↓
Т.2
$S_\alpha = '.'$
да
нет
$S_{p1} := S_{стэк1}$
СТЭК1 ↓
Т.3
$S_\alpha = 'K'$
нет
да
$A2_\gamma := S_\alpha$
Т.2
$A2_\gamma := S_{стэк1}$
$A1_\gamma := \gamma-1$
$S_{p1} = 0$
$A2[S_{p1}] := \gamma$
на вход

Рис. 17

Каждое адресное поле (А1, А2 и А3 (см.рис. 1)) составляет 15 разрядов в соответствии с адресностью машины БЭСМ-6.

В блок-схеме текущий адрес массива поля зрения обозначается через j . Переписываемый массив по-прежнему обозначается через α . Обозначение частей отдельной ячейки совпадает с обозначениями п.1 гл.П (см.рис. 1).

Для установления связи между скобками и символами заводятся два стэка:

стэк 1 - для символов $\underline{K}$,

стэк 2 - для скобок.

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

Когда встречается левая скобка, ее адрес посылается в стэк 2. При появлении правой скобки ее адрес посылается в $A2$ для данной скобки. Кроме этого по адресу, расположенному в стэк 2, заносится текущий адрес, соответствующий правой скобке. Таким образом связываются парные скобки. Таким образом связываются парные скобки. Связь с предыдущими и последующими символами устанавливаются так же, как и в случае значащего символа.

Алгоритм установления связи у символов $\underline{K}$ следующий. Кроме стэка 1, заводится еще рабочая ячейка $P1$. При появлении символа $\underline{K}$ в стэк 1 заносится его адрес в поле зрения. Если этот символ первый, то производится только связь его с последующим символом. Если такой символ уже был записан, то в $A2$ текущей ячейки заносится адрес, находящийся в последней строке стэка 1. Проверяется также содержимое ячейки $P1$. Если она не пуста, то в $A2$ ячейки, имеющей адрес, находящий-

ся в *P1* , заносится текущее значение адреса поля зрения.

В ячейку *P1* каждый раз, как встретится символ $\dot{}$, 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 $\underline{K}$, которому соответствует данная подчеркнутая точка. При появлении $\dot{}$ каждый раз последняя строка стэка I уничтожается.

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

На рис. *18* приведен пример поля зрения, содержащего только символы $\underline{K}$ и $\dot{}$, и последовательная работа вышеописанного алгоритма. Связи между символами $\underline{K}$ указываются с помощью стрелок. Из примера нетрудно видеть, что алгоритм установления связей между символами $\underline{K}$ полностью соответствует требованиям п.I. гл.II.

На рис. *19* приведено поле зрения содержащее значащие символы, скобки, символы $\underline{K}$ и $\dot{}$. В данном примере также показан окончательный результат работы блока записи поля зрения в память машины. Адреса ячеек указываются слева.

Описанием данного блока мы заканчиваем описание препроцессора.

3.3. Процессор

Процессор интерпретирует шаг РЕФАЛ-машины [*3*]. На рис. *20* изображена общая схема шага. Каждый шаг начинается с отыскания ведущего символа конкретизации. Если первый символ конкретного выражения (так в дальнейшем мы будем называть область

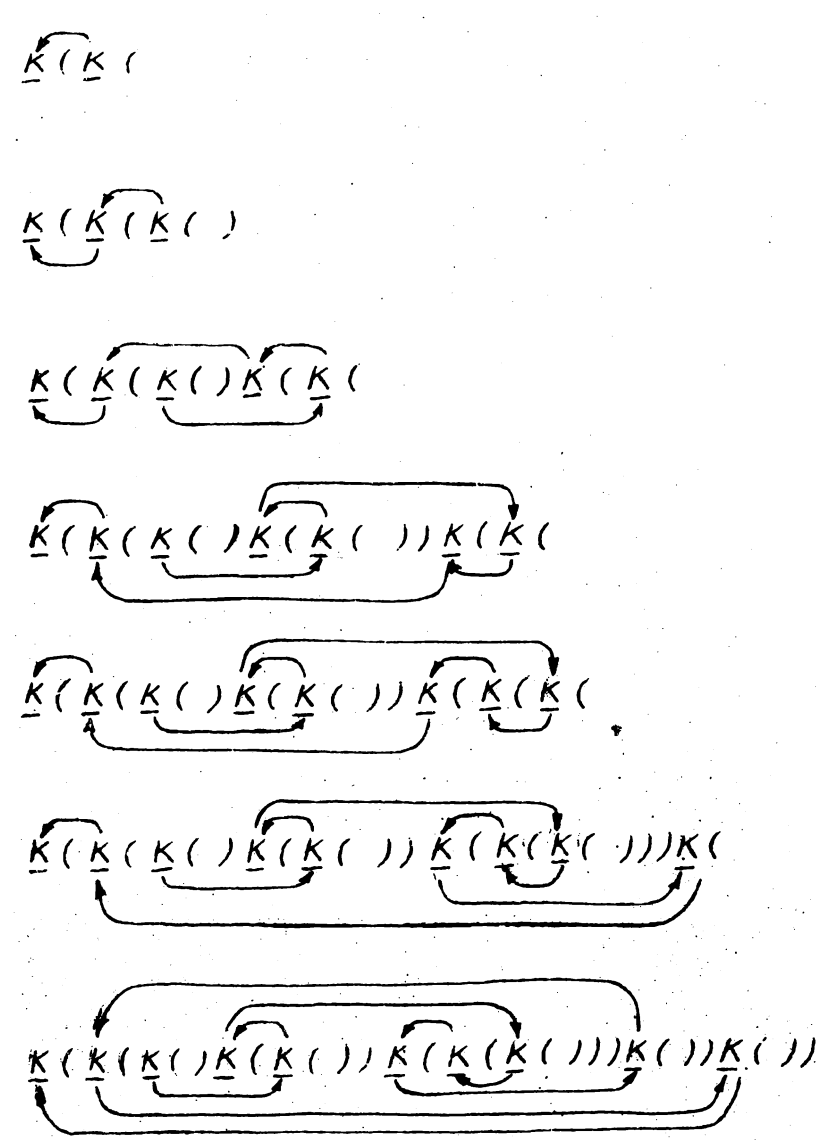


Рис. 18

$\underline{K}(\alpha\beta+(\alpha\beta-\underline{K}(\alpha\delta)+\gamma))\underline{K}(\varepsilon f)$

K 20	1	0	38	21
(21	4	20	37	22
22	0	21	α	23
23	0	22	β	24
24	0	23	+	25
(25	4	24	36	26
26	0	25	α	27
27	0	26	β	28
28	0	27	−	29
K 29	1	28	20	30
(30	4	29	33	31
31	0	30	α	32
32	0	31	δ	33
) 33	2	32	30	34
34	0	33	+	35
35	0	34	γ	36
) 36	2	35	25	37
) 37	2	36	21	38
K 38	1	37	0	39
(39	4	38	42	40
40	0	39	ε	41
41	0	40	f	42
) 42	2	41	39	0

Рис. 19

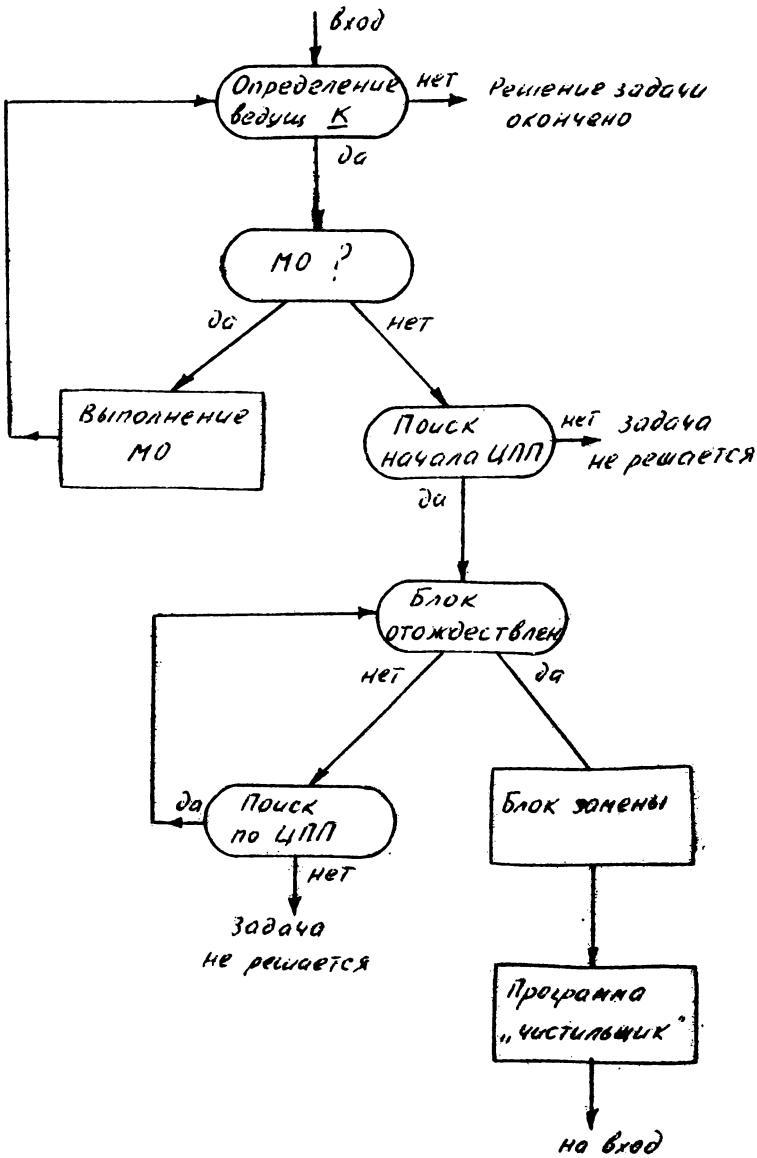

Вход
Определение ведущ. К
нет
Решение задачи окончено
да
МО ?
да
нет
Выполнение МО
Поиск начала ЦПП
нет
задача не решается
да
Блок отождествлен
нет
да
Поиск по ЦПП
да
нет
Задача не решается
Блок замены
Программа „чистильщик"
на вход

Рис. 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 производится замена левой части предложения на правую. После этого включается программа "чистильщик" и управление передается на начало шага.

Работа интерпретатора заканчивается, когда в поле зрения не останется ни одного символа $\underline{K}$.

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

При описании процессора мы подробно остановимся на работе двух основных блоков: блока отождествления и блока замены левой части предложения на правую. Блока определения ведущего символа $\underline{K}$ фактически не существует. В самом начале работы процессора адрес ведущего символа $\underline{K}$ указывается в специальной ячейке. В начале работы очередного шага этот адрес определяется с помощью адресов связи у символов $\underline{K}$ (см. п.I гл.II).

Что касается блока выполнения машинных операций, то в указываются те минимальные требования, которые предъявляет РЕФАЛ-интерпретатор к машинным операциям, вводимым самим

пользователем.

Мы также не будем описывать блоки: поиск начала ЦПП, движение по ЦПП и программу "чистильщик", так как о работе этих блоков достаточно подробно говорилось и в главе II и предыдущем разделе настоящей главы.

3.3.I. Блок отождествления (БО)

Блок отождествления осуществляет алгоритм синтакоического 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 прямой и обратный. Реализация этих алгоритмов, по существу, различается лишь направлением движения по конкретному выражению. Движение по 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 память машины в обратном порядке. Поэтому в настоящем разделе мы опишем только прямой алгоритм синтаксического отождествления.

На рис. *21* изображена общая схема БО. Работа БО начинается с развертки левой части. Напомним, что левая часть предложения хранится в памяти машины в плотно упакованном виде (по 5 символов в ячейке). Такая сжатая запись при отождествлении оказывается 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-

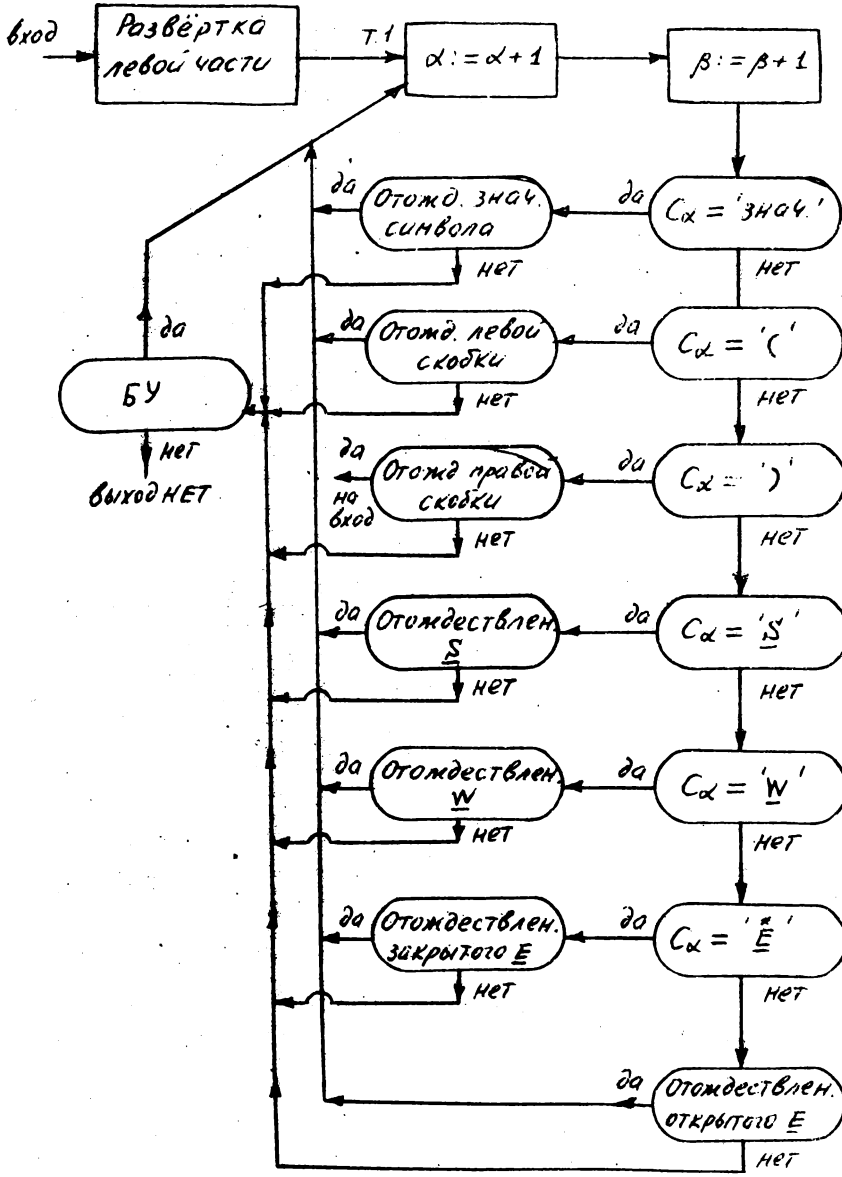

вход
Развёртка левой части
т.1
α := α + 1
β := β + 1
Cα = 'знач.'
да
нет
Отожд. знач. символа
Cα = '('
Отожд. левой скобки
Cα = ')'
Отожд правой скобки
на вход
Cα = 'S'
Отождествлен. S
Cα = 'W'
Отождествлен. W
Cα = 'E'
Отождествлен. закрытого E
Отождествлен. открытого E
БУ
выход нет

Рис. 21

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

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

Развертка левой части осуществляется в специальном массиве памяти. Обозначим текущий адрес такого массива через α .

При синтаксическом отождествлении происходит движение по абстрактному и конкретному выражению слева направо (текущий адрес конкретного выражения на схеме обозначен буквой β). Каждый символ абстрактного выражения проверяется и при этом выделяются семь частных случаев: очередной символ может быть значащим символом, левой скобкой, правой скобкой, переменной $\underline{S}$, переменной $\underline{W}$, закрытой переменной выражения $\underline{E}$, открытой переменной выражения $\underline{\mathcal{E}}$. 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 следующая скобочная структура. Если перед этим была отождествлена последняя скобочная структура, это означает конец работы блока отождествления, 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 $\underline{E}$. В первую очередь удлиняется самая последняя из них. Если до данного момента не было отождествлено ни одной открытой переменной $\underline{E}$, удлинение невозможно. В этом случае управление передается на поиск следующего предположения в ЦПП.

Остановимся более подробно на описании отдельных подблоков блока синтаксического отождествления.

3.3.I.I. 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 γ . Текущий адрес массива, на который переписывается левая часть, обозначается через α .

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

С - содержит код символа,

А1 -содержит информацию о конкретном выражении для опорного символа и правой заключительной скобки,

А2 - для открытой переменной $\underline{E}$ содержит адрес опорного символа, для закрытой $\underline{E}$ адрес правой заключительной скобки данной скобочной структуры, для заключительной скобки А2 содержит адрес левой скобки, за которой следует такой же номер, что и за данной заключительной скобкой.

$\mathcal{I}$ - идентификатор переменной.

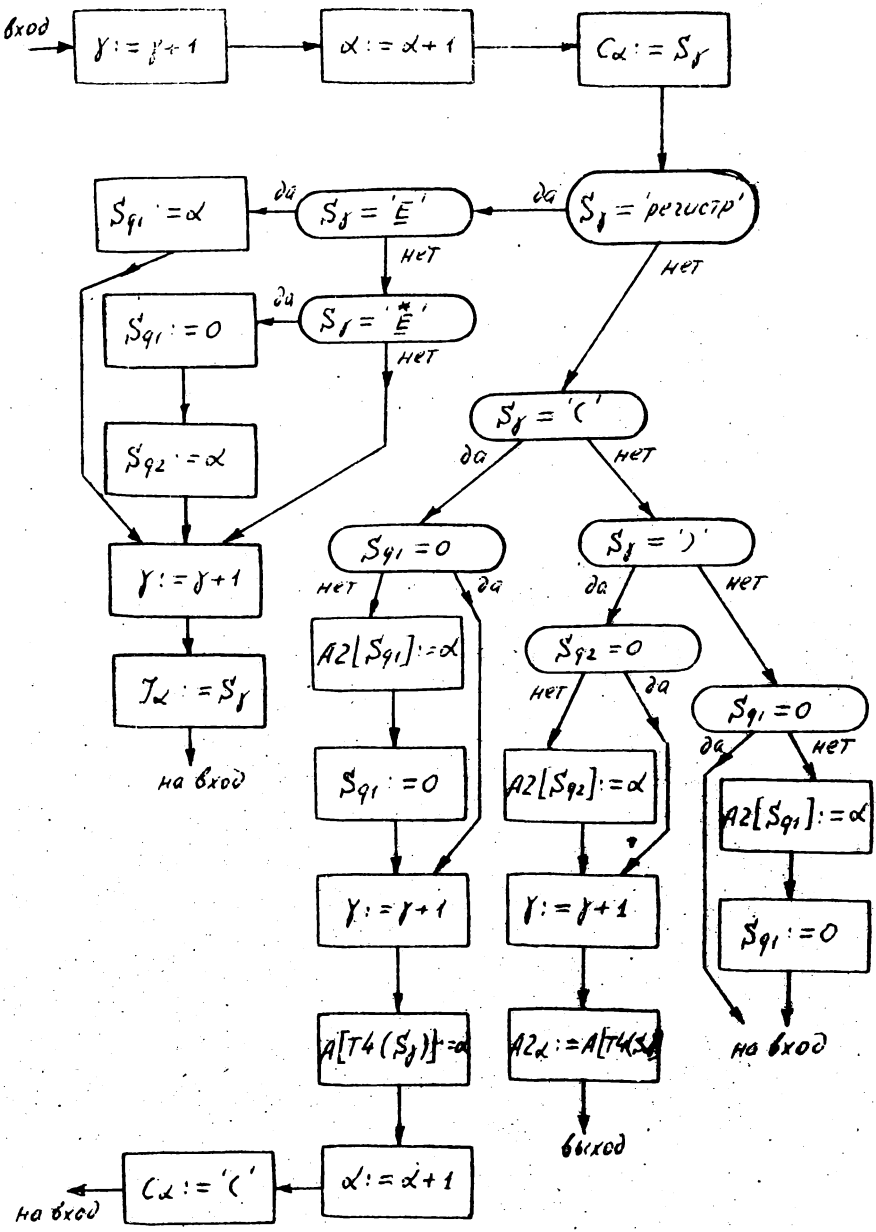

вход
γ := γ+1
α := α+1
Cα := Sγ
Sγ = 'регистр'
да
Sγ = 'Ē'
да
Sq1 := α
да
Sγ = 'Ē*'
нет
Sq1 := 0
нет
Sq2 := α
нет
Sγ = '('
да
нет
γ := γ+1
Sq1 = 0
нет
да
Sγ = ')'
да
нет
Tα := Sγ
на вход
A2[Sq1] := α
Sq2 = 0
нет
да
Sq1 := 0
A2[Sq2] := α
Sq1 = 0
да
нет
γ := γ+1
γ := γ+1
A2[Sq1] := α
A[T4(Sγ)] := α
A2α := A[T4(Sγ)]
Sq1 := 0
выход
на вход
α := α+1
Cα := '('
на вход

Рис. 22

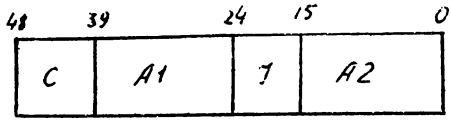


Рис 23

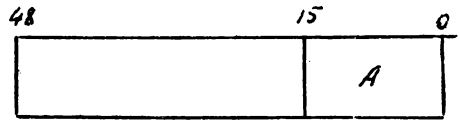


Рис 24

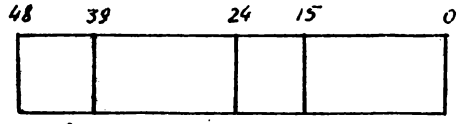


Рис. 25

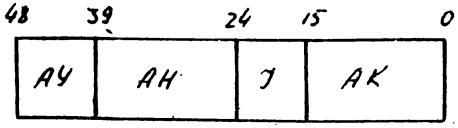


Рис. 26

В процессе переписи происходит заполнение указанных частей ячеек массива α. Для этого вводятся две рабочие ячейки $q1$ и $q2$, а также таблица Т4.

Адрес каждой открытой переменной $\underline{E}$ запоминается в ячейке $q1$. И при дальнейшей переписи адрес первого встретившегося символа, отличного от переменной и правой скобки, посылается в А2 ячейки, адрес которой находится в $q1$. Такой символ в дальнейшем мы и будем называть опорным символом.

Точно так же посылается адрес правой скобки в ячейку, соответствующую закрытой переменной $\underline{E}$. Только в этом случае адрес $\underline{E}$ запоминается в ячейке $q2$.

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 записывается адрес левой скобки, имеющей данный номер. При развертывании очередной скобочной структуры адреса всех левых скобок, содержащихся внутри нее, будут занесены в таблицу Т4. Следует также сказать, что при развертке левой части после каждой левой скобки записывается соответствующая правая скобка.

Работа блока заканчивается после записи заключительной скобки данной скобочной структуры.

3.3.I.2. Отождествление значащих символов.

Когда в абстрактном выражении встречается значащий символ, то он просто сравнивается с очередным символом конкретного выражения. Если символы оказались одинаковыми, управление пе-

редается на продолжение движения по абстрактному и конкретному выражению (т.I на блок-схеме рис. 21). При несовпадении управление передается в БУ.

3.3.I.3. Отождествление левых скобок.

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

При несовпадении символов управление передается в БУ. При совпадении в левую скобку абстрактного выражения (в часть А2) посылается адрес соответствующей ей левой скобки конкретного выражения. Это делается для того, чтобы в дальнейшем 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 структуры.

Таким образом, после отождествления очередной скобочной структуры все встретившиеся левые скобки с помощью адресов, расположенных в А2, оказываются связанными с левыми скобками пропущенных скобочных структур конкретного выражения.

3.3.I.4. Отождествление правых 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.

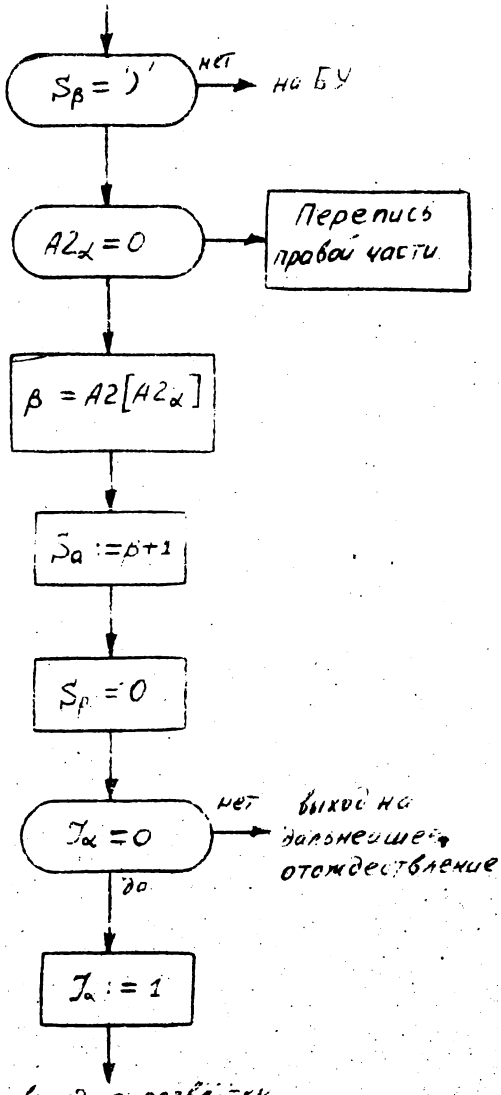

$S_\beta = 1$
нет
на БУ
$A2_\alpha = 0$
Перепись правой части
$\beta = A2[A2_\alpha]$
$P_a := p+1$
$S_p = 0$
$J_\alpha = 0$
нет
выход на дальнейшее отождествление
да
$J_\alpha := 1$
выход на развёртку левой части

рис. 27

Если $A2_{\alpha}=0$, это значит, что левая часть предложения отождествлена полностью. В противном случае необходимо продолжить отождествление пропущенных скобочных структур. Для этого по адресу связи, находящемуся в А2 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 $\beta := A2[A2_{\alpha}]$.

Информация, заносимая в рабочие ячейки α и β , необходима для работы других блоков.

3.3.I.5. Отождествление переменных.

Кроме значащих символов и скобок в абстрактном выражении могут встретиться свободные переменные $\underline{S}$, $\underline{W}$ и $\underline{E}$ Интерпретатор среди переменных $\underline{E}$ различает закрытые и открытые.

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

Для запоминания содержимого отождествленных переменных заводится таблица TI, состоящая из I6 ячеек. На рис. 25 изображена такая ячейка. В АН заносится адрес начала содержимого переменной, а в АК – адрес конца.
Обращение к таблице TI осуществляется с помощью идентификатора переменной.

Например:

Пусть (... W3 ...) – абстрактное выражение, а

(... (...) ...) – конкретное выражение.
3 3

Тогда после отождествления переменной W3 в ячейку T1+3 в АН занесется адрес (, а в АК – адрес) .
3 3

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 называемое тождество одинаковых переменных. Тот факт, что данная переменная встретилась не впервый раз, определяется с помощью таблицы T2.

В таблице T2 хранятся счетчики числа отождествленных переменных с одним и тем же идентификатором. Одинаковым переменным в такой таблице соответствует одна ячейка. Число ячеек в таблице T2 равно числу возможных.

Перед отождествлением очередной переменной с помощью ее идентификатора проверяется содержимое соответствующего счетчика таблицы T2. 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 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 T2 отнимаются единички. Такой механизм счетчиков точно определяет когда нужно проводить тождество.

Для более эффективного осуществления алгоритма удлинение и замены вводится таблица T3 (см.рис. 26).

Ее точнее можно было бы назвать стэком переменных. Таблица Т3 содержит 32 ячейки. В нее по мере отождествления переменных заносятся их идентификаторы (часть J). Кроме этого, для некоторых переменных $\underline{W}$ и $\underline{E}$ в эту таблицу заносятся адреса начала и конца (соответственно, АН и АК). Для открытой переменной $\underline{E}$ в АУ таблицы Т3 заносится ее адрес в абстрактном выражении (АУ занимает 9 разрядов, что вполне достаточно для указания адреса в массиве α, так как он ограничен 100 ячейками и находится в постоянном месте памяти).

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

3.3.1.6. Отождествление переменной $\underline{W}$.

На рис. *28* изображена блок-схема алгоритма отождествления переменной $\underline{W}$.

Прежде всего проверяется содержимое соответствующего счетчика таблицы 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 Т1 заносится адрес конца и начала содержимого переменной: $AK[T1(J_\alpha)] := S_{\alpha 3}$ и $AH[T1(J_\alpha)] := \beta$. Затем заполняется очередная ячейка таблицы Т3 (текущий адрес обозначается буквой j): $AK_j := S_{\alpha 3}$, $AH_j := \beta$ и $J_j := J_\alpha$. Счетчик таблицы Т2, соответствующий

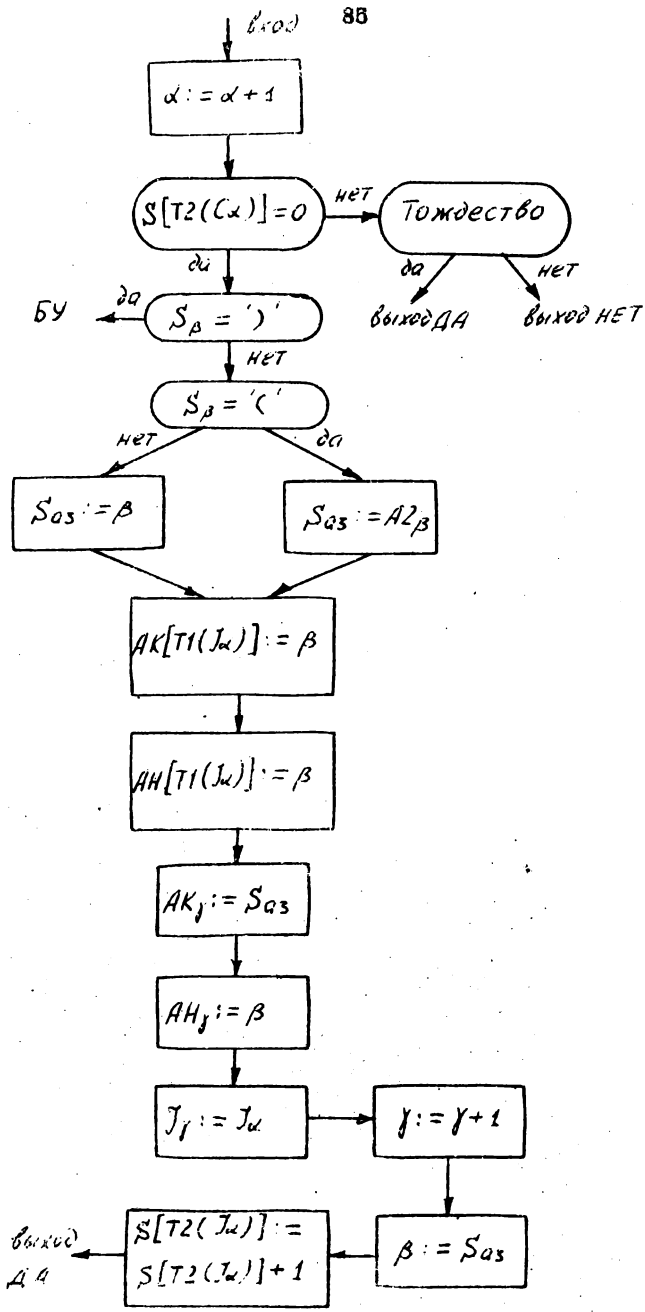

вход
α := α + 1
S[T2(Cα)] = 0
нет
тождество
да
нет
да
нет
БУ
да
Sβ = ')'
выход ДА
выход НЕТ
нет
Sβ = '('
нет
да
Sαз := β
Sαз := A2β
AK[T1(Jα)] := β
AH[T1(Jα)] := β
AKγ := Sαз
AHγ := β
Jγ := Jα
γ := γ + 1
β := Sαз
S[T2(Jα)] := S[T2(Jα)] + 1
выход ДА

Рис. 28

идентификатору переменной, увеличивается на I:

$S[T2(J_{\alpha})] := S[T2(J_{\alpha})] + 1$.

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

Алгоритм проверки тождества переменных одинаков для всех переменных и в общем-то достаточно прост. Поэтому специально на нем мы не останавливаемся. Укажем только, что после успешного отождествления содержимое таблицы TI не меняется, однако как всегда счетчик таблицы T2 увеличивается на I, а в таблицу T3 заносится информация об очередной отождествленной переменной. В случае неудачного отождествления управление передается на БУ.

3.3.I.7. Отождествление закрытой переменной $\underline{E}$.

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

Отождествление закрытой переменной $\underline{E}$ интерпретатор проводит следующим образом.

По адресу, хранящемуся в A2 ячейки переменной $\underline{E}$, 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

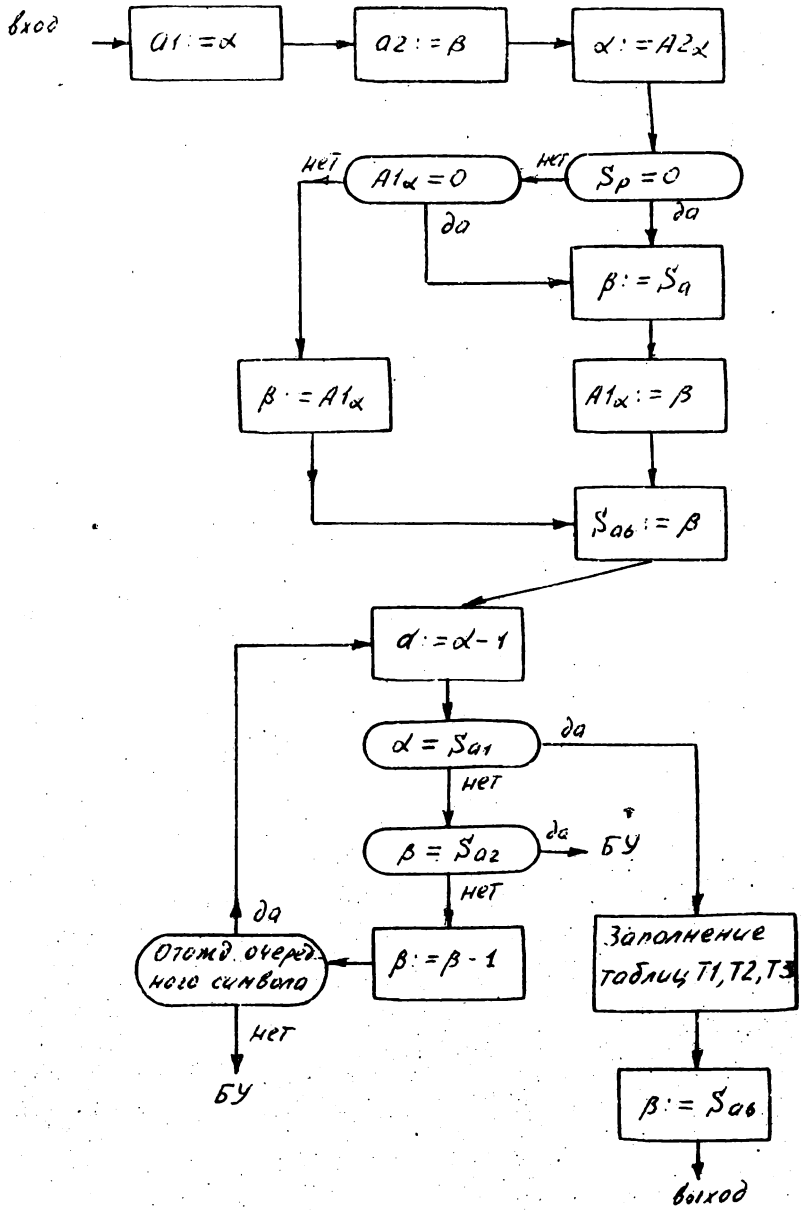

вход
A1 := α
A2 := β
α := A2α
Sр = 0
нет
A1α = 0
нет
да
да
β := Sа
β := A1α
A1α := β
Sаб := β
α := α - 1
α = Sа1
да
нет
β = Sа2
да
БУ
нет
да
Отожд. очеред. ного символа
β := β - 1
нет
БУ
Заполнение таблиц T1, T2, T3
β := Sаб
выход

Рис. 29

в AI ячейки абстрактного выражения, соответствующей правой заключительной скобке данной скобочной структуры. По какому именно адресу зависит от того, вошли ли мы в эту скобочную структуру обычным путем ($S_p = 0$) или же из БУ. Во втором случае AI 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 $S_p = 0$, адрес правой скобки конкретного выражения определяется по содержимому рабочей ячейки a .

После того, как найдены правые 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 $\underline{S}$ и $\underline{W}$ отождествляются обычным образом. Блок заканчивает свою работу после того, как в абстрактном выражении вновь встретится закрытая переменная $\underline{E}$. В этом случае оставшаяся часть конкретного выражения относится к содержимому данной переменной. Для этого заполняются таблицы T1, T2 и T3.

Отождествление будет неудачным в следующих двух случаях:

I) Когда при движении по конкретному выражению мы переходим границу начала содержимого переменной $\underline{E}$.

Например, пусть имеется абстрактное выражение

$$(\ldots\ \underline{E}4\underline{S}2\alpha\underline{W}1\ )$$

и конкретное выражение

$$(\ldots\alpha\underset{1}{(}\ldots\underset{1}{)}\ )$$

При обратном движении для отождествления $\underline{S}2$ нам бы

пришлось перейти через символ α ,стоящий в конкретном выражении. Это является недопустимым, и поэтому управление передается в БУ.

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

Например, если у нас имеется абстрактное выражение

$$(\ldots \ \underline{E} \alpha \ \underline{W1} \ )$$

и конкретное выражение

$$(\ldots \ (\ ) (\ ) \beta \ (\ \ldots \ )_1 \ )_1$$

то они не могут быть отождествлены друг с другом. В самом деле, при движении справа налево переменная $\underline{W1}$ примет значение $(\ \ldots \ )_1$, 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 ТI только адрес очередного символа конкретного выражения.

3.3.I.8 Отождествление открытой переменной $\underline{E}$.

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 символ в абстрактном выражении. Для этого используется адрес находящийся в А2 ячейки, соответствующей данной переменной $\underline{E}$. Затем включается блок поиска символа, равного опорному, в конкретном выражении.

После того как найдены опорные символы, дальнейшее

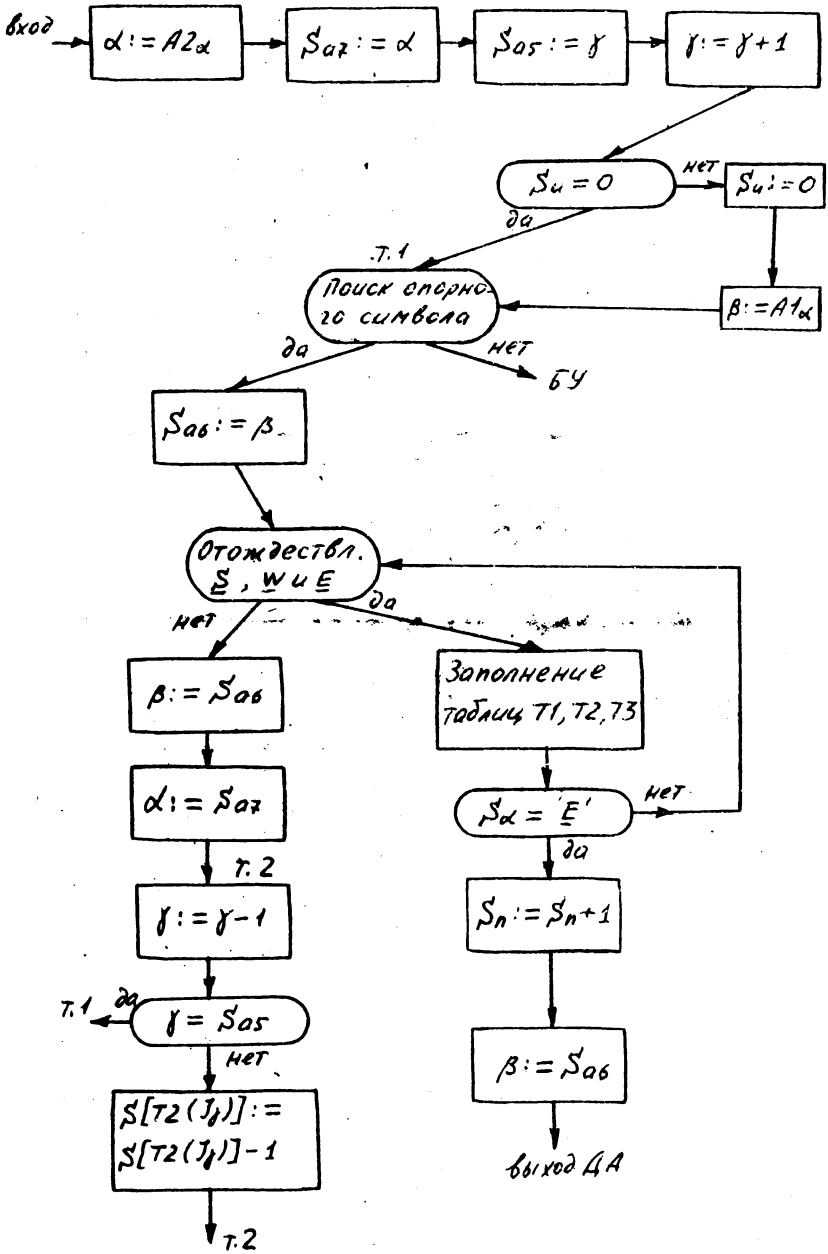

вход
α := A2α
S_a7 := α
S_a5 := γ
γ := γ+1
S_u = 0
нет
S_u := 0
да
т.1
Поиск опорного символа
β := A1α
да
нет
БУ
S_a6 := β
Отождествл. S, W и E
нет
да
β := S_a6
Заполнение таблиц T1, T2, T3
α := S_a7
S_α = E'
нет
да
т.2
γ := γ-1
S_n := S_n+1
т.1
да
γ = S_a5
нет
β := S_a6
S[T2(J_γ)] := S[T2(J_γ)] - 1
выход ДА
т.2

Рис. 30

отождествление - становится аналогичным отождествлению закрытой переменной $\underline{E}$. 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 j). При этом идентификаторы переменных, хранящихся в этой таблице, используются для обращения к таблице Т2.

Приведем пример местного удлинения. Пусть

$(\ldots \underline{E}1\underline{S}2\underline{S}2+\ldots)$ - абстрактное выражение, а

$(\ldots()+(A-B)+AB+AA+\ldots)$ - конкретное выражение.

В данном примере переменная $\underline{E}1$ 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 $\underline{S}2$ 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 неудачным. И только тогда, когда в качестве опорного символа будет взять четвертый знак +, отождествление будет удачным. Переменная $\underline{S}2$ примет значение A, а переменная EI станет равной $(\ )+(A-B)+AB+$.

На этом же примере можно показать необходимость вторичного движения к опорному символу и вычитания единицы из счетчиков таблицы T2. Действительно, при вторичном удлинении $\underline{E}$I опорным символом в конкретном выражении будет третий слева знак +. Отождествление в этом случае не произойдет из-за того, что не выполняется тождество одинаковых переменных $\underline{S}2$. Однако в этом случае правая $\underline{S}2$ все-таки примет значение B и в таблице T2 во второй ячейке будет записана I. Если бы мы при возврате к опорному символу не вычитали I из второй ячейки таблицы T2, то отождествление $\underline{S}2$ после третьего удлинения переменной $\underline{E}$I начиналось бы с проверки тождества. И отождествление бы не произошло.

При существующем алгоритме единица вычитается, и $\underline{S}2$ при новом отождествлении предстает как впервые встретившаяся переменная.

В рабочей ячейке n заводится счетчик числа отождествленных открытых переменных $\underline{E}$. Эта информация необходима для БУ.

3.3.I.9. Блок удлинения (БУ).

На рис. 31 изображена блок-схема алгоритма определения адреса последней из числа отождествленных открытой переменной $\underline{E}$. Для этого используется таблица T3 (на блок-схеме текущий адрес этой таблицы обозначен буквой γ).

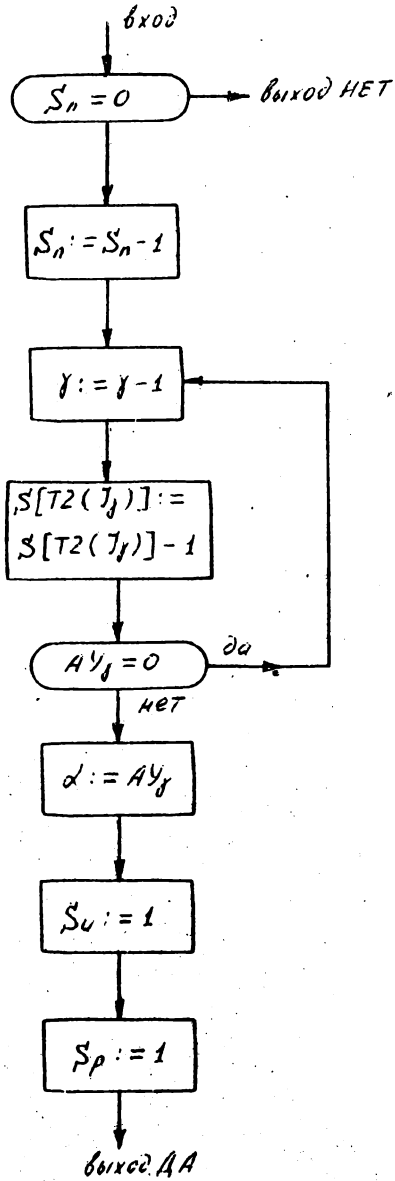

вход
$S_n = 0$
выход НЕТ
$S_n := S_n - 1$
$\gamma := \gamma - 1$
$S[T2(J_\gamma)] :=$
$S[T2(J_\gamma)] - 1$
$AY_\gamma = 0$
да
нет
$\alpha := AY_\gamma$
$S_u := 1$
$S_p := 1$
выход АА

Рис. 31

БУ идет по таблице Т3 в обратном направлении и использует идентификаторы переменных (J_j) для обращения к таблице Т2. Для всех переменных, за исключением открытых, АУ ячейки таблицы Т3 равно 0. Так что первая же встретившаяся переменная, у которой $AУ_j \neq 0$, объявляется удлиняемой.

Содержимое рабочих ячеек u и ρ используется для отождествления переменной $\underline{E}$ (см. 3.I.7 и 3.I.8).

После того, как найдена удлиняемая переменная $\underline{E}$, 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 блока является адрес начала СПП (свободного поля памяти), полный адрес начала правой части и содержимое таблицы Т2 и Т3.

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

Парные скобки связываются между собой с помощью адресов связи, помещаемых в А2. При этом работает точно такой же алгоритм, как и в препроцессоре (см.блок-схему рис. 14). Для этого заводится стек скобок, к которому производится

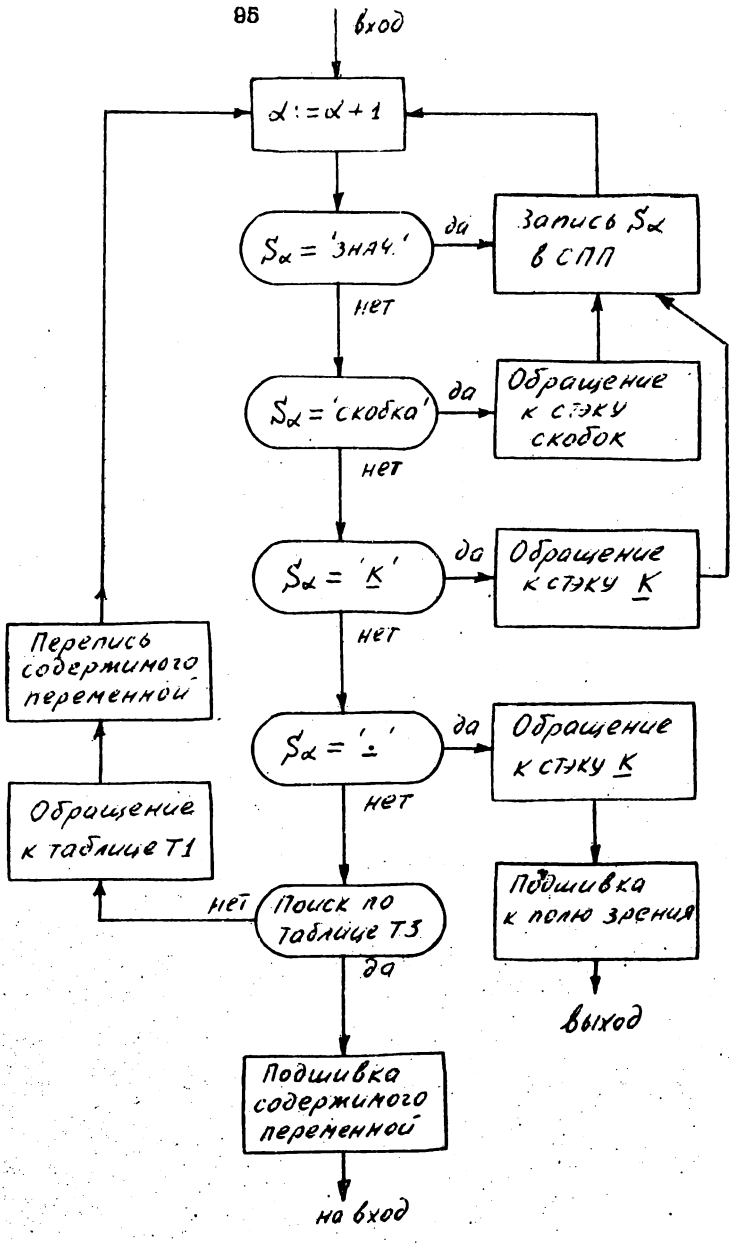

вход
α := α + 1
Sα = 'знач.'
да
Запись Sα в СПП
нет
Sα = 'скобка'
да
Обращение к стэку скобок
нет
Sα = 'К'
да
Обращение к стэку К
нет
Sα = '÷'
да
Обращение к стэку К
нет
Поиск по таблице Т3
нет
да
Обращение к таблице Т1
Перепись содержимого переменной
Подшивка к полю зрения
выход
Подшивка содержимого переменной
на вход

Рис. 32

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

Алгоритм связи символов $\underline{K}$ совпадает с аналогичным алгоритмом препроцессора. Для этого заводится второй стэк - стэк $\underline{K}$.

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. Параметры РЕФАЛ-интерпретатора для БЭСМ-6

Размеры основных блоков (см. рис. 4):

БСК - 600_8 ячеек ;

БПИ - 300_8 ячеек;

Препроцессор – 4000_8 ячеек;

Процессор – 8000_8 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.

4.1. 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 (САБСЕТ-АЛГОЛе), заключенный в "скобки" *К ТРАЛГ И* – , вводится в поле зрения РЕФАЛ-интерпретатора (ТРАЛГ -детерминатив процедуры "трансляция с АЛГОЛа"). Интерпретатор производит переработку текста на входном языке в текст на выходном языке (автокод БЕМШ) в соответствии с набором предложений в его памяти. Таким образом, процесс трансляции происходит в режиме интерпретации, этот процесс напоминает работу транслятора "управляемого синтаксисом". Результат-программа на автокоде БЕМШ – является, конечно, скомпилированной эффективной программой, то есть транслятор, описанный на РЕФАЛе, есть обычный транслятор-компилятор. В процессе трансляции интерпретатор выдает сообщения о синтаксических ошибках в АЛГОЛ-

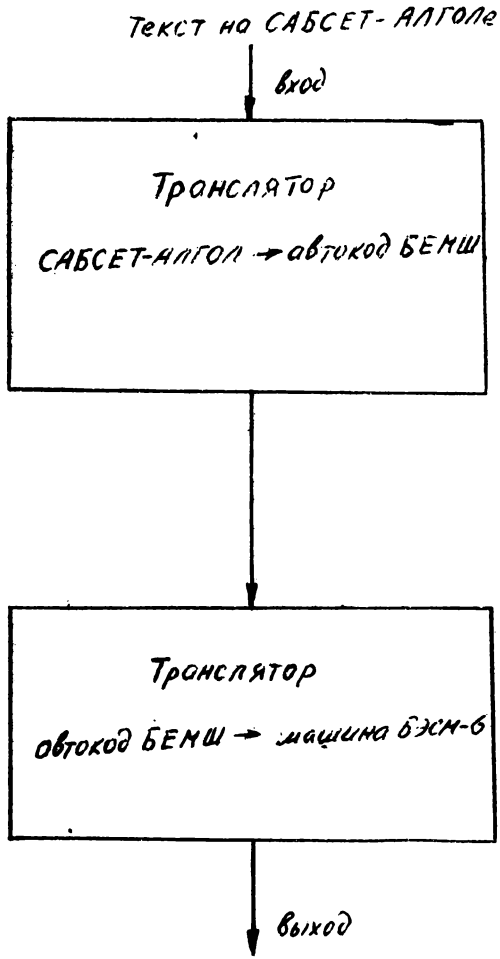

Текст на САБСЕТ-АЛГОЛе
вход
Транслятор
САБСЕТ-АЛГОЛ → автокод БЕМШ
Транслятор
автокод БЕМШ → машина БЭСМ-6
выход

рис. 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 рассчитан на перевод небольших программ (не более 2-3 тыс.команд автокода). Расширение возможностей транслятора в направлении увеличения объема программ может быть достигнуто путем введения машинных 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 (по 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 связан 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, написавшему блок синтаксического контроля и участвовавшему в обсуждении ряда вопросов.

Литература

I. В.Ф.Турчин, Метаязык для формального описания алгоритмических языков, в сб."Цифровая вычислительная 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. Bobrow D.G., Raphael B. A Comparison of List-Processing computer Languages. Comm ACM, 7, 4, April 1964.

5. С.Н.Флоренцев, В.Ю.Олюнин, В.Ф.Турчин, РЕФАЛ-интерпретатор в сб. "Труды 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 ВЦАН СССР, Москва 1967 г.

8. В.Ф.Турчин, В.И.Сердобольский, Язык РЕФАЛ и его использование для преобразования алгебраических 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 автоматизации обработки экономической информации. Москва, май 1967г.

IO. 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

Подписано к печати "12" VI 1969 г.

№ 89209 от "19" VI 1969 г. Заказ № 976 Тираж 300 экз.

Ордена Ленина институт прикладной математики
Москва, Миусская пл., 4